



FREE eBook

LEARNING batch-file

Free unaffiliated eBook created from
Stack Overflow contributors.

#batch-file

Table of Contents

About.....	1
Chapter 1: Getting started with batch-file.....	2
Remarks.....	2
Examples.....	2
Opening a Command Prompt.....	2
Editing and Viewing Batch Files.....	3
Getting Help.....	3
Chapter 2: Add delay to Batch file.....	5
Introduction.....	5
Examples.....	5
Timeout.....	5
Timeout.....	5
Pause.....	6
Ping.....	6
Ping.....	6
Sleep.....	7
Sleep.....	7
Chapter 3: Batch and JSCript hybrids.....	8
Introduction.....	8
Examples.....	8
Embedded JScript In a Batch File.....	8
Run JScript With Temporary Files.....	8
Chapter 4: Batch and VBS hybrids.....	10
Introduction.....	10
Examples.....	10
Run VBS with temporary file(s).....	10
Embed vbscript code into batch file without using temporary files.....	11
Chapter 5: Batch file command line arguments.....	12
Examples.....	12
Command line arguments supplied to batch files.....	12

Batch files with more than 9 arguments.....	12
Shifting arguments inside brackets.....	13
Chapter 6: Batch file macros.....	15
Introduction.....	15
Examples.....	15
Basic Macro.....	15
Comments.....	15
\$ Character Usages.....	15
Command separator.....	15
Command-line arguments.....	15
Macros In Batch Script.....	16
Chapter 7: Batch files and Powershell hybrids.....	17
Examples.....	17
Run Powershell with Temporary Files.....	17
Use POWERSHELL Command To Execute 1-line Powershell Command.....	17
Powershell/batch hybrid without temp files.....	18
Chapter 8: Best Practices.....	19
Introduction.....	19
Examples.....	19
Quotes.....	19
Examples and Solutions.....	19
Example A.....	19
Solution A.....	19
Example B.....	19
Solution B.....	20
Spaghetti Code.....	20
Examples and Solutions.....	20
Example A.....	20
Solution A.....	20
Example B.....	20
Chapter 9: Bugs in cmd.exe processor.....	22

Introduction.....	22
Remarks.....	22
Examples.....	22
Parentheses Confusion.....	22
Cause.....	22
Solution.....	22
Improper Escape Character.....	23
Cause.....	23
Solutions.....	23
Extra.....	23
DEL File Extension.....	23
Cause.....	23
Solution.....	24
Chapter 10: Bypass arithmetic limitations in batch files.....	25
Introduction.....	25
Examples.....	25
Using powershell.....	25
Using jscript.....	25
Emulating pen and paper calculations, math functions implementations.....	26
Chapter 11: Changing Directories and Listing their Contents.....	27
Syntax.....	27
Remarks.....	27
Examples.....	27
To display the current directory.....	27
To change the current directory (without changing drives).....	27
Navigating to a directory on a different drive.....	28
How to show all folders and in files in a directory.....	28
Changing drive without CD /D.....	28
To change the current directory to the root of the current drive.....	29
Chapter 12: Comments in Batch Files.....	30
Introduction.....	30

Syntax.....	30
Examples.....	30
Using REM for Comments.....	30
Using Labels as Comments.....	30
Using Variables as Comments.....	31
Block Comments.....	31
Comment on the code's line.....	31
Batch and WSF Hybrid Comment.....	32
Chapter 13: Creating Files using Batch.....	33
Introduction.....	33
Syntax.....	33
Remarks.....	33
Examples.....	33
Redirection.....	33
Echo to create files.....	34
Saving the output of many commands.....	35
Chapter 14: Deprecated batch commands and their replacements.....	37
Examples.....	37
DEBUG.....	37
APPEND.....	38
BITSADMIN.....	38
Chapter 15: Differences between Batch (Windows) and Terminal (Linux).....	39
Introduction.....	39
Remarks.....	39
Examples.....	39
Batch Commands and Their Bash Equivalents.....	39
Batch Variables and Their Bash Equivalent.....	42
Chapter 16: Directory Stack.....	43
Syntax.....	43
Parameters.....	43
Remarks.....	43
Examples.....	43

Delete Text Files.....	43
Print Directory Stack.....	43
Chapter 17: Echo.....	45
Introduction.....	45
Syntax.....	45
Parameters.....	45
Remarks.....	45
Examples.....	45
Displaying Messages.....	45
Echo Setting.....	46
Getting and Setting.....	46
Echo outputs everything literally.....	47
Echo output to file.....	47
@Echo off.....	49
Turning echo on inside brackets.....	49
Chapter 18: Elevated Privileges in Batch Files.....	50
Examples.....	50
Requesting Elevate Privileges in a Shortcut.....	50
Requesting Elevated Privileges at Runtime.....	51
Requesting runtime elevated privileges without UAC prompt.....	51
Chapter 19: Escaping special characters.....	54
Introduction.....	54
Examples.....	54
Escape using caret(^).....	54
Escaping the caret.....	54
Security issue.....	55
FIND and FINDSTR Special Characters.....	55
FIND.....	55
FINDSTR.....	55
FOR /F Special Characters.....	56
FOR /F.....	56

Extra Special Characters.....	.56
Escaping through the pipeline.....	.57
Chapter 20: File Handling in batch files.....	.58
Introduction.....	.58
Examples.....	.58
Creating a File in Batch.....	.58
How to Copy Files in Batch.....	.58
Moving Files.....	.58
Deleting Files.....	.59
Copy Files Without xcopy.....	.59
Editing Nth Line of a File.....	.60
Chapter 21: For Loops in Batch Files.....	.62
Syntax.....	.62
Remarks.....	.62
Examples.....	.62
Looping through each line in a files set.....	.62
Recursively Visit Directories in a Directory Tree.....	.63
Renaming all files in the current directory.....	.63
Iteration.....	.64
Chapter 22: Functions.....	.65
Remarks.....	.65
Examples.....	.65
Simple Function.....	.65
Function With Parameters.....	.66
Function Utilizing setlocal and endlocal.....	.66
Combining them all.....	.66
Anonymous functions in batch files.....	.67
Calling functions from another batch file.....	.67
Chapter 23: If statements.....	.69
Syntax.....	.69
Remarks.....	.69
1-Line Syntaxes.....	.69

Multiline Syntaxes	.69
Examples	.70
Comparing numbers with IF statement	.70
Comparing strings	.70
Comparing Errorlevel	.70
Check if file exists	.71
If variable exists / set	.71
Chapter 24: Input and output redirection	.72
Syntax	.72
Parameters	.72
Remarks	.72
Examples	.72
An Example	.72
Redirect special character with delayed expansion enabled	.73
Write to a file	.73
Chapter 25: Random In Batch Files	.75
Examples	.75
Random Numbers	.75
Generating Random Numbers Within Specific Range	.75
Generating Random Numbers larger than 32767	.75
Pseudorandom	.76
Random Alphabets	.76
Pseudorandom And Uniform Random In Batch	.76
Pseudorandom Distribution	.76
Uniform Distribution	.76
Chapter 26: Search strings in batch	.78
Examples	.78
Basic strings search	.78
Using search results	.78
Chapter 27: Using Goto	.79
Introduction	.79

Syntax.....	79
Parameters.....	79
Remarks.....	79
Examples.....	79
Example Programs.....	79
Goto with variable.....	80
Chapter 28: Variables in Batch Files.....	81
Examples.....	81
Declaration.....	81
Notes about quotation marks.....	81
Spaces in variables.....	81
Using quotation marks to eliminate spaces.....	81
Usage.....	82
Variable Substitution.....	82
Declare multiple variables.....	84
Using a Variable as an Array.....	84
Operations on Variables.....	85
Setting variables from an input.....	87
Credits.....	88

About

You can share this PDF with anyone you feel could benefit from it, download the latest version from: [batch-file](#)

It is an unofficial and free batch-file ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official batch-file.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with batch-file

Remarks

From Microsoft Technet:

With batch files, which are also called batch programs or scripts, you can simplify routine or repetitive tasks. A batch file is an unformatted text file that contains one or more commands and has a .bat or .cmd file name extension. When you type the filename at the command prompt, Cmd.exe runs the commands sequentially as they appear in the file.

Batch File Names and Extensions

Extension	Remarks
.bat	This extension runs with MS-DOS and all versions of Windows
.cmd	Used for batch files in Windows NT family
.btm	The extension used by 4DOS and 4NT

To understand the difference between `.cmd` and `.bat` please see [here](#).

Avoid names which are already the name of built-in commands. like `tracert`. There is a utility called `tracert.exe`. So, avoid naming a batch file `tracert.bat`

Running Batch File

The easiest way to run a batch file is simply double-clicking its icon. Or paste the file full path into a command prompt, or just its name if command Prompt was started from the batch file directory, then enter.

Example:

```
C:\Foo\Bar>test.bat  
C:\Foo\Bar>C:\Foo\Bar\Baz\test.bat
```

Examples

Opening a Command Prompt

The command prompt comes pre-installed on all Windows NT, Windows CE, OS/2 and eComStation operating systems, and exists as `cmd.exe`, typically located in `C:\Windows\system32\cmd.exe`

On Windows 7 the fastest ways to open the command prompt are:

- Press `Win` + `Space`, type "cmd" and then press `Enter`.
- Press `Win` + `R`, type "cmd" then then press `Enter`.
- If you have an explorer window open, type "cmd" in the address bar to open a prompt in the currently selected directory.
- Right-click a folder in Explorer while holding `Shift` and select "Open command window here".

It can also be opened by navigating to the executable and double-clicking on it.

In some cases you might need to run `cmd` with elevated permissions, in this case right click and select "Run as administrator". This can also be achieved by pressing `Control+Shift+Enter` instead of `Enter` when using way 1 of the points above.

Editing and Viewing Batch Files

Any ASCII editor can edit batch files. A list of editors that can syntax highlight batch syntax can be found [here](#). You can also use the default notepad shipped with windows to edit and view a batch file, although it does not offer syntax highlighting.

To open notepad:

- Press `Win` + `R`, type `notepad` and then press `Enter`.

Alternatively, the most "primitive" way to [create a batch file](#) is to redirect output from the command line to a file, eg.

```
echo echo hello world > first.bat
```

which writes `echo hello world` to the file `first.bat`.

You can edit a batch file by right clicking the file and selecting "Edit" from the context menu.

To view the contents of a batch file from within a command prompt, run the following command:

```
type first.bat
```

You can also start editing your batch file with notepad from the command prompt by typing

```
notepad first.bat
```

Getting Help

To get help on a batch file command you can use the built-in help.

Open a command prompt (whose executable is `cmd.exe`) and enter `help` to see all available

commands.

To get help for any of these commands, type `help` followed by the name of the command.

For example:

```
help help
```

Will display:

```
Provides help information for Windows commands.  
HELP [command]  
command - displays help information on that command.
```

Some commands will also display help if followed by `/?`.

Try:

```
help /?
```

Note:

`Help` will only display the help for **internal** commands.

Read Getting started with batch-file online: <https://riptutorial.com/batch-file/topic/1277/getting-started-with-batch-file>

Chapter 2: Add delay to Batch file

Introduction

This topic will teach you one of the many useful things to know in the scripting language, batch file; Adding a delay/pause/timeout to your batch file.

Examples

Timeout

Timeout

The simplest way to make a delay or pause for a certain amount of time, is with the standard command `TIMEOUT`. To make a timeout that lasts exactly one minute we type:

```
timeout /t 60
```

Now what is going on here?

First off we use the command `TIMEOUT` with the parameter `/T` (which simply means timeout) then we specify the amount of seconds to wait. In this case... 60 seconds.

Timeout with the parameter `/NOBREAK`

If we take the example from before and run that in a BATCH file: `timeout /t 60` then while waiting those 60 seconds, you are actually able to break the timeout by pressing any key on your keyboard. To prevent this we simply add the parameter `/NOBREAK` to the end of it.

```
timeout /t 60 /nobreak
```

By doing this it will timeout for 60 seconds, and if you want to break the timeout you will have to press (CTRL-C) on your keyboard.

Silent timeout

When it's doing a timeout it will display:

```
Waiting for X seconds, press a key to continue ...  
or  
Waiting for X seconds, press CTRL+C to quit ... [This is with the /NOBREAK parameter]
```

To hide the message use the `NUL` argument (For explanation of `NUL`: [Click Here](#))

```
timeout /t 60 > nul  
or  
timeout /t 60 /nobreak > nul
```

Pause

To make your script pause simply use the `PAUSE` command.

```
PAUSE
```

This will display the text `Press any key to continue . . .`, then add a newline on user input.

Let's say we want to create a "Hello World" program and after we click something on our keyboard, we want it to exit the program with the `EXIT` command.

```
echo Hello World  
pause  
exit
```

Here it uses the `ECHO command` to say "Hello World". Then we use the `PAUSE` command which displays `Press any key to continue . . .` and then we use the `EXIT` command to terminate the current BATCH script.

When it's pausing it will display:

```
Press any key to continue . . .
```

Hide the "Press any key to continue... prompt

To hide the message we redirect the output to a special device called `nul`. This isn't actually a real device, but whatever we send to it is thrown away.

```
pause > nul
```

Ping

Ping

One of the most used command to delay for a certain amount of time is `ping`.

Basic usage

```
PING -n 1 -w 1000 1.1.1.1  
  
REM the -n 1 flag means to send 1 ping request.  
REM the -w 1000 means when the IP(1.1.1.1) does not respond, go to the next command  
REM 1.1.1.1 is an non-existing IP so the -w flag can ping a delay and go to next command
```

This would output the following on your batch file/console:

```
C:\Foo\Bar\Baz>ping -n -w 1000 1.1.1.1

Pinging 1.1.1.1 (Using 32 bytes of data)
Request timed out

Ping statistics for 1.1.1.1
    Packets: Sent = 2, Received = 0, Lost = 1 (100% loss)
```

Hide the text echoed out

Just add `>nul` at the back of the command to redirect it to null.

```
ping -n w 1000 1.1.1.1 >nul
```

This would output nothing.

Sleep

Sleep

On older Windows system, `timeout` is not available. However, we can use the `sleep` command.

Usage

```
sleep 1
```

Very self-explanatory; sleep for 1 second. However, `sleep` is a deperacted command and should be replaced by `timeout`.

Availability

This command is available on old Windows system. Also `SLEEP.exe` is included in 2003 Resource Kit.

To use `sleep.exe`, put the executable file to `%Windir%\System32` folder. Then you can use it as normal command.

Read Add delay to Batch file online: <https://riptutorial.com/batch-file/topic/9123/add-delay-to-batch-file>

Chapter 3: Batch and JScript hybrids

Introduction

JScript is actually the superset of Javascript (it's 1.8.1 version - so some newer features are not available), and they can be embedded into a batch script for extending batch script's functions. Usually, techniques of embedding are using the JScript directives (not part of the official Javascript standard) in order to separate the batch and JScript code. JScript allows you to work with Com/ActiveX objects, as well as with WMI objects in addition to the standard Javascript.

Examples

Embedded JScript In a Batch File

This following example is created by user Michael Dillon from [this answer](#).

Consider the following script:

```
@set @junk=1 /*  
@echo off  
cscript //nologo //E:javascript %0 %*  
goto :eof  
*/  
  
//JScript aka Javascript here
```

This script snippet does:

- Execute the `cscript` command which calls itself with all the arguments provided.
- As the part after `@set @junk=1` is commented (`/*` and `*/` Are valid JScript comment),
- JScript will ignore them.
- **Note: We need the `@set @junk=1` part because the batch file does not recognize `/*` as a command, but a `set` statement will be a workaround. JScript will recognize `/*` as a comment so the other batch file will not be executed by JScript engine.**

You can add your JScript after `*/` and start extending your batch file scripting!

Run JScript With Temporary Files

As mentioned [here](#), the old-school method to run another script is by using temporary files. Simple `echo` it into a file and then run it (and remove it optionally).

Here's the basic concept:

```
@echo off
echo //A JS Comment > TempJS.js
echo //Add your code>>TempJS.js

cscript //nologo //e:cscript.exe TempJS.js

del /f /s /q TempJS.js
```

But this would require lots of `echo` statements to create a relatively large JScript. Here's a better method by Aacini.

```
@echo off
setlocal

rem Get the number of the "<resource>" line
for /F "delims=:" %%a in ('findstr /N "<resource>" "%~F0"') do set "start=%%a"

rem Skip such number of lines and show the rest of this file
(for /F "usebackq skip=%start% delims=" %%a in ("%~F0") do echo %%a) > TempJS.js

cscript //nologo //e:cscript.txt TempJS.js
del /f /s /q TempJS.js

goto :EOF

<resource>
JScript
JScript
JScript
```

Read Batch and JSCript hybrids online: <https://riptutorial.com/batch-file/topic/10578/batch-and-jscript-hybrids>

Chapter 4: Batch and VBS hybrids

Introduction

Batch are capable of running with VBS functionality further increasing their reliability. For example, VBS can deal with decimals, spaces, and some other advanced operations that cannot be done in batch. Also is capable of working with WMI and ActiveX objects.

Examples

Run VBS with temporary file(s)

The old-school method for running another script from batch is to echo the script into another location, and then run it.

This method can be represented like this:

```
@echo off
rem VBS below
echo your vbs > TempVBS.vbs
echo other vbs>>TempVBS.vbs
rem End of VBS

cscript //nologo TempVBS.vbs
del /f /s /q TempVBS.vbs
```

The method above would require lots of echo (vbs) >> TempVBS.vbs, so here's a way to shorten it. (*code by Aacini*)

```
@echo off
setlocal

rem Get the number of the "<resource>" line
for /F "delims=:" %%a in ('findstr /N "<resource>" "%~F0"') do set "start=%%a"

rem Skip such number of lines and show the rest of this file
(for /F "usebackq skip=%start% delims=" %%a in ("%~F0") do echo %%a) > Program.vbs

cscript //nologo Program.vbs
del /f /s /q Program.vbs
exit /b

<resource>
your vbs
another line of vbs
```

The last method is by using streams. A file can have a few streams. And every stream can contain different information.

```
@echo off
```

```

echo vbs >%0:stream1
rem This command redirect "vbs" into the stream1 of this script, then we can call it later

cscript %0:stream1 //nologo
rem if you like, you can clear the stream1 of this file by:
type nul>%0:stream1

```

Embed vbscript code into batch file without using temporary files

Here's an example with the technique(hack) invented by the [dostips](#) forums' user Liviu:

```

@echo off
echo Printed by CMD.EXE
cscript //nologo "%~f0?.wsf" //job:JS //job:VBS

exit /b %errorlevel%

----END OF BATCH CODE---
<package>
<job id="JS">
<script language="VBScript">

    WScript.Echo("Printed by VBScript"):

</script>
</job>
<job id="VBS">
<script language="JScript">

    WScript.Echo("Printed by JScript");

</script>
</job>
</package>

```

As running `wsf` file with [windows script host](#) is extension sensitive you can run a file with any extension by adding `?.wsf` at the end of the file (which is the core of the hack). While the Liviu's example is probably more robust the above code is more simplified version. As wsh does not care much about the things outside the `<package>` node you are not obligated to put everything in xml comments. Though it's to be careful with redirection symbols (`<` and `>`)

Read Batch and VBS hybrids online: <https://riptutorial.com/batch-file/topic/10061/batch-and-vbs-hybrids>

Chapter 5: Batch file command line arguments

Examples

Command line arguments supplied to batch files

Batch file command line arguments are parameter values submitted when starting the batch. They should be enclosed in quotes if they contain spaces. In a running batch file, the arguments are used for various purposes, i.e. redirection to :labels, setting variables, or running commands.

The arguments are referred to in the batch file using %1, %2, ..., %9.

```
@echo off
setlocal EnableDelayedExpansion
if not "%1"=="" (
    set "dir=%~1" & set "file=%~2"
    type !dir!\!file! | find /n /i "True" >nul^
        && echo Success! || echo Failure
)
exit /b

C:\Users\UserName> test.bat "C:\Temp\Test Results" "Latest.log"
Success!
```

Notes:

- In the above example, double quotes are removed by using the argument modifier %~1.
- Long strings are split to several lines using ^, and there is a space before the character on the next line.

Batch files with more than 9 arguments

When more than 9 arguments are supplied, the shift [/n] command can be used, where /n means start at the nth argument, n is between zero and eight.

Looping through arguments:

```
:args
set /a "i+=1"
set arg!i!=%~1
call echo arg!i! = %%arg!i%%%
shift
goto :args
```

Note, in the above example delayed expansion variable `i` is used to assign argument values to variables array. The `call` command allows to display such variable values inside the loop.

Counting arguments:

```
for %%i in (*.*) do (set /a ArgCount+=1)
echo %ArgCount%
```

Set a variable to n'th argument:

```
set i=5
call set "path%i%=%~i"
```

Shifting arguments inside brackets

Lets have the following `example.bat` and call it with arguments `1 ,2 and 3`:

```
@echo off

(
    shift
    shift
    echo %1
)
```

As the variable expansion will change after the the end brackets context is reached the output will be:

1

As this might be an issue when shifting inside brackets to access the argument you'll need to use `call`:

```
@echo off

(
    shift
    shift
    call echo %%1
)
```

now the output will be `3`. As `CALL` command is used (which will lead to additional variable expansion) with this technique the arguments accessing can be also parametrized:

```
@echo off

set argument=1

shift
shift
call echo %%argument%
```

with delayed expansion:

```
@echo off
setlocal enableDelayedExpansion
set argument=1

shift
shift
call echo %%!argument!
```

the output will be

3

Read Batch file command line arguments online: <https://riptutorial.com/batch-file/topic/4981/batch-file-command-line-arguments>

Chapter 6: Batch file macros

Introduction

In a command prompt, you can use DOSKEY for creating macros. In a batch file you can define a variable that can be called as a piece of code and even pass arguments to it.

Examples

Basic Macro

Using DOSKEY, we can create macros to simplify typing many commands in command prompt. Take a look at the following example.

```
DOSKEY macro=echo Hello World
```

Now if you type `macro` in the command prompt, it would return `Hello World`.

Comments

Unfortunately, DOSKEY macro doesn't support comment, but there's a workaround.

```
:= Comment
:= Comment
:= Remember to end your comment with :=
:=
```

\$ Character Usages

There are 3 usages of the \$ character in a DOSKEY macro.

Command separator

`$T` is the equivalent of `&` in a batch script. One can join commands together like so.

```
DOSKEY test=echo hello $T echo world
```

Command-line arguments

Like bash(not batch), we use `$` to indicate command-line argument.

`$1` refers to the first command-line argument

`$2` refers to second command-line argument, etc..

`$*` refers to all command-line argument

Macros In Batch Script

DOSKEY macros don't work in a batch script. However, we can use a little workaround.

```
set DOSKEYMacro=echo Hello World  
%DOSKEYMacro%
```

This script can simulate the macro function. One can also use ampersands(&) to join commands, like `$T` in DOSKEY.

If you want a relatively large "macro", you may try a [simple function](#) or take a look at other function topics [here](#).

Read Batch file macros online: <https://riptutorial.com/batch-file/topic/10791/batch-file-macros>

Chapter 7: Batch files and Powershell hybrids

Examples

Run Powershell with Temporary Files

This has been mentioned in other [hybrid topics](#) again and again. The old-school, but easy method to run Powershell is by:

- echoing the Powershell script into a temporary script
- Execute the temporary script
- Optionally remove the temporary script

This is a sample script.

```
@echo off
echo powershell-command>Temp.ps1
echo another line>>Temp.ps1
    rem echo the script into a temporary file

powershell -File Temp.ps1
    rem execute the temporary script

del Temp.ps1
    rem Optionally remove the temporary script
```

The method above requires tons of `echo` statement if a long script is required, here is a better method suggest by @Aacini

```
@echo off
setlocal

    rem Get the number of the "<resource>" line
for /F "delims=:" %%a in ('findstr /N "<resource>" "%~F0"') do set "start=%%a"

    rem Skip such number of lines and show the rest of this file
(for /F "usebackq skip=%start% delims=" %%a in ("%~F0") do echo %%a) > Temp.ps1

powershell -File Temp.ps1
del /f /s /q Temp.ps1

goto :EOF

<resource>
PS
Powershell script
```

Use POWERSHELL Command To Execute 1-line Powershell Command

Using the `POWERSHELL` command, we can execute a 1-line command directly from a batch script,

without any temporary file.

Here's the syntax.

```
powershell.exe -Command <yourPowershellCommandHere>
```

You may also want to include other flags, like `-Nologo` to improve the actual outcome.

Powershell/batch hybrid without temp files

This is the approach proposed by the stackoverflow's user [rojo](#) which also can handle the command line arguments :

```
<# : batch portion
@echo off & setlocal

(for %%I in ("%~f0";*) do @echo(%%~I) | ^
powershell -noprofile "$argv = $input | ?{$_}; iex (${%~f0} | out-string)"

goto :EOF
: end batch / begin powershell #>

"Result:"
$argv | %{ "`$argv[{0}]: $_" -f $i++ }
```

called like this:

```
psbatch.bat arg1 "This is arg2" arg3
```

will produce:

```
Result:
$argv[0]: C:\Users\rojo\Desktop\test.bat
$argv[1]: arg1
$argv[2]: This is arg2
$argv[3]: arg3
```

Read Batch files and Powershell hybrids online: <https://riptutorial.com/batch-file/topic/10711/batch-files-and-powershell-hybrids>

Chapter 8: Best Practices

Introduction

This topic will focus on the things that one should (*not mandatory*) do in a batch file. Using these "best practices" can enhance the effect and the function of a batch file.

Examples

Quotes

Most online batch scripts come with a lot of quote issues.

Examples and Solutions

Example A

```
if %var%==abc echo Test
```

This code works - when the content of `%var%` does not contain space or other special characters. Now let's assume `%var%` contains 1 whitespace. Now `cmd.exe` sees:

```
if ==abc echo Test
```

This would cause a failure because `cmd.exe` doesn't understand this syntax.

Solution A

```
if "%var%"=="abc" echo Test
```

Using quotes, `cmd.exe` sees the entire `%var%`(including space and special characters) as only one normal string. Yet this is not the safest comparison method. The safest one uses `echo`, `pipe`, and `findstr`.

Example B

```
cd C:\User\Spaced Name\Spaced FileName.txt
```

`cd` would only change directory to `C:\User\Spaced`, as `cd` only accepts one path argument.

Solution B

Simply by adding quotes around the path, the issue would be solved.

```
cd "C:\User\Spaced Name\Spaced FileName.txt"
```

There are also a few examples that work better using quotes, like the `set /a` statement, etc. But, when one works on strings that contain spaces or special characters, it is usually much safer to use quotes.

Spaghetti Code

Spaghetti code means a code snippet that uses many, and often confusing structures. Such as `GOTOS`, exceptions and inconsistent code.

Examples and Solutions

Example A

```
@echo off
set /a counter=0

:Loop
set /a counter=%counter% + 1
echo %counter%

if %counter% equ 10 goto :exit
goto :Loop

:exit
```

This program comes with plenty of jumps, making us harder to know what exactly the script is doing.

Solution A

```
@echo off
for /l %%G in (0,1,10) echo %%G
```

Using less `GOTOS`, we reduced the amount of code greatly, and we can focus on the actual code.

Example B

Consider the following statements.

```
:endGame
if %player1Score% gtr %player2Score% goto :player1wins
if %player1Score% lss %player2Score% goto :player2wins
goto :tie

:player1wins
echo player 1 wins
goto :eof

:player2wins
echo player 2 wins
goto :eof

:tie
echo tie
goto :eof
```

This snippet requires lots of `goto` statements and can be confusing to debug. To simplify these statements, we can use `call` command. Here is the above script at a better condition.

```
:endGame
if %player1Score% gtr %player2Score% call :message player 1 wins
if %player1Score% lss %player2Score% call :message player 2 wins
if %player1Score% equ %player2Score% call :message tie

goto :eof

:message
echo %*
goto :eof
```

Both scripts output the exact same result, but the new script is much shorter and clearer.

Read Best Practices online: <https://riptutorial.com/batch-file/topic/10746/best-practices>

Chapter 9: Bugs in cmd.exe processor

Introduction

This topic will focus on errors caused by the processor bugs. Here are the things we would focus on the cause and the solution of the issue.

Remarks

In the example [DEL File Extension](#), user X. Liu notices that this bug will **not** occurs when the file extension in the `DEL` command is less than 3 characters.

Examples

Parentheses Confusion

From [this](#) website, the OP has noticed a problem.

Cause

Consider the following code snippet.

```
if 1==1 (
    set /a result = 2*(3+4)
)
```

At your first glance, you may think `CMD.exe` would process it like so:

- The condition is true, execute code block
- Set variable `result`'s value to 14
- Continue

However, `CMD.exe` process like so:

- The condition is true, execute code block
- **Calculate $2 * (3 + 4)$, the) after 4 is processed at the end of if code block**
- A random) has appeared!

The second step would return `Unbalanced parentheses` error.

Solution

According to a German CMD.exe's `set /?`, we would need to quote arithmetic operations. Here's an example.

Previous	Result
<code>set /a result=2+5*4</code>	<code>set /a result="2+5*4"</code>

By the way, according to an English CMD.exe `set /?`, quotes are required if logical or modulus operators are present in the expression (although this is not a *must-do* step).

Improper Escape Character

In this [Stack Overflow question](#), user txtechhelp found an issue with the `^` character which could cause a security issue.

Cause

```
anyInvalidCommand ^
```

Note: Make sure the caret(`^`) is the last character! Any extra `CR\LF` won't work at all!

The caret looks for the next character to escape. However, there are no more characters available to escape, so `cmd` loops infinitely, looking for a character to escape. In this "loop" process, `cmd.exe` will consume your computer memory. And gradually eat all memory, bringing the computer to knees.

This issue can lead to more serious security worries as one could just enter the code into the one's unlocked computer.

Solutions

- Use codepage **UTF-16** could solve this problem. Only **UTF-8** or **ASCII** would cause the bug.
- Make sure there is an extra `CR\LF` in the file, or just simply don't use caret at the end of the file.

Extra

This bug seems to be solved in Windows 10.

DEL File Extension

This bug was reported by steve2916 from this [Microsoft Windows Forum thread](#).

Cause

Consider a folder with such files.

```
TestA.doc  
TestB.doc  
TestC.docx  
TestD.docx
```

If we want to remove all `.doc` file in this directory, we usually would do:

```
del *.doc
```

However, this command also removes the `.docx` files. The same happens on file extensions with this pattern.

File A	File B
Anynname. abc	AnotherName. abcd

As we can see, as long as the file extension string contains the string used in the `del` command, the file will be deleted. Note that this bug only occurs when the extension string used in the `del` command has at least three characters. For instance, `del *.do` doesn't delete `A.doc` or `A.docx`.

Solution

In the thread, user MicroCompsUnltd noticed that using `Powershell` would solve the issue.

Read Bugs in cmd.exe processor online: <https://riptutorial.com/batch-file/topic/10694/bugs-in-cmd-exe-processor>

Chapter 10: Bypass arithmetic limitations in batch files

Introduction

Batch files allows only 32bit integer calculations , though this can be bypassed with different approaches.

Examples

Using powershell

As the powershell is installed by default on every windows system from 7/2008 and above it can be used for more complex calculations:

```
@echo off
set "expression=(2+3)*10/1000"
for /f %%# in ("powershell %expression%") do set result=%%#
echo %result%
```

Mind the additional double quotes in the `for /f` which prevent brackets conflicts with the for command syntax.

Potential issue is that powershell is much slower than using wsh/vbscript/jscript due to the loading of the .net framework

Using jscript

WSH/JScript is installed on every windows system since NT so using it for more complex calculations makes it pretty portable. JScript is easier for combining it with batch file :

```
@if (@codesection==@batch) @then
@echo off

set "expression=2*(2+3)/1000"
for /f %%# in ('cscript //nologo //e:jscript "%~f0" "%expression%") do set
result=%%#
echo %result%
:: more batch code

exit /b %errorlevel%
@end
WScript.Echo (eval(WScript.Arguments(0)));
```

With this approach you can put your whole code in a single file. It is faster than using powershell. [Here](#) and [here](#) more advanced scripts can be found (which can be used as external files).

Emulating pen and paper calculations, math functions implementations

1. [Here](#) can be found the most comprehensive math library that emulates pen and paper calculations and allows working with bigger numbers.
2. Here are another examples of pen and paper emulations: [ADD](#) , [Comparison](#) , [Multiply](#)
3. Some math functions implementations can be found [here](#).

Read Bypass arithmetic limitations in batch files online: <https://riptutorial.com/batch-file/topic/10702/bypass-arithmetic-limitations-in-batch-files>

Chapter 11: Changing Directories and Listing their Contents

Syntax

- echo %cd% - displays the current path of the directory
- cd "C:\path\to\some\directory" -changes the path of the directory
- cd "%variableContaining_directory_path%" - also changes the path of the directory
- cd /d E: - change to E: drive from a different drive
- cd/ - changes directory back to current drive
- echo %__CD__% - displays the current path of the directory with trailing backslash (undocumented)
- echo %=C:% - The current directory of the C: drive (undocumented)
- echo %=D:% - The current directory of the D: drive if drive D: has been accessed in the current CMD session (undocumented)

Remarks

Why is it important and what are they uses and advantages:

- to open file or application in a directory using batch
- to create and write and read files in a directory using batch
- to know and list out all folders
- to know where your batch file is running

Examples

To display the current directory

Format and Usage:

```
echo %cd%
```

%cd% is a system variable that contains the current directory path

To change the current directory (without changing drives)

Format:

```
cd "<path>"
```

Example:

```
cd "C:\Program Files (x86)\Microsoft Office"
```

`cd` is an abbreviation for `chdir` and the two commands behave in the exact same way. For the sake of consistency, `cd` will be used throughout this topic.

To navigate to the directory one level above the current directory, specify the system directory ...

```
cd ..
```

To navigate to a directory that is inside of the current directory, simply `cd` to the folder name without typing the full path (wrapping the directory name in quotes if it contains spaces).

For example, to enter `C:\Program Files (x86)\Microsoft Office` while in the `C:\Program Files (x86)` directory, the following syntax may be used:

```
cd "Microsoft Office"
```

or

```
cd "C:\Program Files (x86)\Microsoft Office"
```

Navigating to a directory on a different drive

`cd` by itself will not allow a user to move between drives. To move to a different drive, the `/d` option must be specified.

e.g. Moving from `C:\Users\jdoe\Desktop` to `D:\Office Work`

```
cd /d "D:\Office Work"
```

How to show all folders and files in a directory

Usage to list all folders and files in the current directory: `dir`

A target directory can also be specified: `dir C:\TargetPath`

When specifying a path with spaces, it must be surrounded by quotes: `dir "C:\Path With Spaces"`

Changing drive without CD /D

```
Pushd "D:\Foo"  
Dir
```

```
Popd
```

`Pushd` will change the directory to the directory following (in this case D:\Foo. `Popd` returns back to the original directory.

To change the current directory to the root of the current drive

Format:

```
cd/
```

`cd/` is set to change the current directory back to the root of the current drive

Read Changing Directories and Listing their Contents online: <https://riptutorial.com/batch-file/topic/7132/changing-directories-and-listing-their-contents>

Chapter 12: Comments in Batch Files

Introduction

Comments are used to show information in a batch script.

Syntax

- REM
- &REM
- ::
- &::
- Goto :Label

Comments. You can also use |><, etc.

:Label

Examples

Using REM for Comments

```
REM This is a comment
```

- REM is the official comment command.

Using Labels as Comments

```
::This is a label that acts as a comment
```

The double-colon :: comment shown above is not documented as being a comment command, but it is a special case of a label that acts as a comment.

Caution: when labels are used as comments within a bracketed code block or for command, the command processor expects every label to be followed by at least one command, so when a jump is made to the label it will have something to execute.

The cmd shell will try to execute the second line even if it is formatted as a label (and **this causes an error**):

```
(
```

```
echo This example will fail
:: some comment
)
```

When working within bracketed code blocks it is definitely safer to use **REM** for all comment lines.

Using Variables as Comments

It is also possible to use variables as comments. This can be useful to conditionally prevent commands being executed:

```
@echo off
setlocal
if /i "%~1""=="update" (set _skip=) Else (set _skip=REM)
%_skip% copy update.dat
%_skip% echo Update applied
...
```

When using the above code snippet in a batch file the lines beginning with `%_skip%` are only executed if the batch file is called with `update` as a parameter.

Block Comments

The batch file format does not have a block comment syntax, but there is an easy workaround for this.

Normally each line of a batch file is read and then executed by the parser, but a `goto` statement can be used to jump past a block of plain text (which can be used as a block comment):

```
@echo off
goto :start

A multi-line comment block can go here.
It can also include special characters such as | >

:start
```

Since the parser never sees the lines between the `goto :start` statement and `:start` label it can contain arbitrary text (including control characters without the need to escape them) and the parser will not throw an error.

Comment on the code's line

To comment on the same line as the code you can use `&::` or `&rem`. You can also use `&&` or `||` to replace `&`.

Example :

```
@echo off
echo This is a test &::This is a comment
echo This is another test &rem This is another comment
```

```
pause
```

A curiosity: SET command allows *limited* inline comments without &rem:

```
set "varname=varvalue"    limited inline comment here
```

Limitations:

- syntax with double quotes set "varname=varvalue" **Or** set "varname=",
- an inline comment **may not** contain any double quote,
- any cmd poisonous characters | < > & **must be** properly escaped as ^| ^< ^> ^&,
- parentheses () **must be** properly escaped as ^(^) within a bracketed code block.

Batch and WSF Hybrid Comment

```
<!-- : Comment
```

This works with both batch script and WSF. The closing tag(-->), only works in WSF.

Code	Sucessful in both batch and WSF?
<!-- : Comment	True
<!-- : Comment -->	False - The closing tag only works for WSF
-->	False

Read Comments in Batch Files online: <https://riptutorial.com/batch-file/topic/3152/comments-in-batch-files>

Chapter 13: Creating Files using Batch

Introduction

One useful feature of batch files is being able to create files with them. This section shows how to create files using batch code.

Syntax

- echo (type here whatever you want in the to be) >> (filename)
- echo (variable name) >> (filename)

Remarks

If a file exists, `>` will overwrite the file and `>>` will append to the end of the file. If a file does not exist, both will create a new file.

Also, the `echo` command automatically adds a newline after your string.

So

```
echo 1 > num.txt
echo 1 > num.txt
echo 2 >> num.txt
```

will create the following file:

```
1
2
```

Not this:

```
1 1 2
```

or

```
1 2
```

Furthermore, you cannot just modify a single line in a text file. You have to read the whole file, modify it in your code and then write to the whole file again.

Examples

Redirection

Format:

```
[command] [> | >>] [filename]
```

> saves the output of [command] into [filename].

>> appends the output of [command] into [filename].

Examples:

1. echo Hello World > myfile.txt saves "Hello World" into myfile.txt

2. echo your name is %name% >> myfile.txt appends "your name is xxxx" into myfile.txt

3. dir c:\ > directory.txt saves the directory of C:\ to directory.txt

Echo to create files

Ways to create a file with the echo command:

```
ECHO. > example.bat (creates an empty file called "example.bat")
```

```
ECHO message > example.bat (creates example.bat containing "message")
```

```
ECHO message >> example.bat (adds "message" to a new line in example.bat)
```

```
(ECHO message) >> example.bat (same as above, just another way to write it)
```

If you want to create a file via the ECHO command, in a specific directory on your computer, you might run into a problem.

```
ECHO Hello how are you? > C:\Program Files\example.bat
```

(This will NOT make a file in the folder "Program Files", and might show an error message)

But then how do we do it? Well it's actually extremely simple... When typing a path or file name that has a space included in its name, then remember to use "quotes"

```
ECHO Hello how are you? > "C:\Program Files\example.bat"
```

(This will create "example.bat" in the folder "Program Files")

But what if you want to make a file that outputs a new file?

```
ECHO message > file1.bat > example.bat
```

(example.bat is NOT going to contain "message > file1.bat")
example.bat will just contain "message"... nothing else

Then how do we do this? Well for this we use the ^ symbol.

```
ECHO message ^> file1.bat > example.bat
```

```
(example.bat is going to contain "message > file1.bat")
```

Same goes for other stuff in batch

The next step requires you to have some knowledge about variables and statements:

[click here to learn about variables](#) | [click here to learn about if and else statements](#)

Variables:

```
SET example="text"  
ECHO %example% > file.bat  
(This will output "text" to file.bat)
```

if we don't want it to output "text" but just plain %example% then write:

```
ECHO ^%example^% > file.bat  
(This will output "%example%" to file.bat)
```

IF/ELSE statements:

```
ELSE statements are written with "pipes" ||  
  
IF ^%example^%=="Hello" ECHO True || ECHO False > file.bat  
  
(example.bat is going to contain "if %example%=="Hello" echo True")  
[it ignores everything after the ELSE statement]
```

to output the whole line then we do the same as before.

```
IF ^%example^%=="Hello" ECHO True ^|^ ECHO False > file.bat  
  
This will output:  
IF %example%=="Hello" ECHO True || ECHO False
```

If the variable is equal to "Hello" then it will say "True", ELSE it will say "False"

Saving the output of many commands

Using many ECHO commands to create a batch file:

```
(  
echo echo hi, this is the date today  
echo date /T  
echo echo created on %DATE%  
echo pause >nul  
)>hi.bat
```

The command interpreter treats the whole section in parenthesis as a single command, then saves all the output to hi.bat.

hi.bat

now contains:

```
echo hi, this is the date today  
date /T  
echo created on [date created]  
pause >nul
```

Read Creating Files using Batch online: <https://riptutorial.com/batch-file/topic/7129/creating-files-using-batch>

Chapter 14: Deprecated batch commands and their replacements

Examples

DEBUG

`DEBUG` command was used to create binaries/executables from batch file. The command is still available in 32bit versions of windows , but is able to create binaries only with 16bit instructions with makes it unusable for 64bit machines. Now `CERTUTIL` is used for that purpose which allows to decode/encode binary/media files from HEX or BASE64 formats.Example with a file that creates icon file:

```
@echo off

del /q /f pointer.jpg >nul 2>nul
certutil -decode "%~f0" pointer.jpg
hh.exe pointer.jpg
exit /b %errorlevel%

-----BEGIN CERTIFICATE-----
/9j/4AAQSkZJRgABAgAAZABkAAD/7AARRHVja3kAAQAEAAAAMgAA/+4ADkFkb2J1
AGTAAAAAAf/bAIQACAYGBgYGCAYGCAwIBwgMDgoICAoOEAO NDg0NEBEMDg0NDgwR
DxITFBMSDxgYGhoYGCMiIiIjJycnJycnJwEJC AgJCgkLCQkLDgsNCw4RDg4O
DhETDQ0ODQ0TGBePDw8PERgWFxQUFBcW GhoYGBoaISEgISEnJycnJycn/8AA
EQgACgAKAwEiAAIRAQMRAf/EAFsAAQEBAAAAAAAAAAAAAAGBweBAQAAAAAA
AAAAAAAAAAAAAEQAAIBAwQDAAAAAAAAAAAEDAgARBSExIwQSIhMRAQEBAAAA
AAAAAAAAAAAAARIf/aAAwDAQACEQMRAD8A13PZ5eIX3g08ktKZfFPksvQ8r4uL
ecJmx1BMSbm8D6UVKVcg/9k=
-----END CERTIFICATE-----
```

Here's the same with hex format:

```
@echo off

(echo 0000 ff d8 ff e0 00 10 4a 46 49 46 00 01 02 00 00 64) >pointer.hex
(echo 0010 00 64 00 00 ff ec 00 11 44 75 63 6b 79 00 01 00) >>pointer.hex
(echo 0020 04 00 00 00 32 00 00 ff ee 00 0e 41 64 6f 62 65) >>pointer.hex
(echo 0030 00 64 c0 00 00 00 01 ff db 00 84 00 08 06 06 06) >>pointer.hex
(echo 0040 06 06 08 06 06 08 0c 08 07 08 0c 0e 0a 08 08 0a) >>pointer.hex
(echo 0050 0e 10 0d 0d 0e 0d 0d 10 11 0c 0e 0d 0d 0e 0c 11) >>pointer.hex
(echo 0060 0f 12 13 14 13 12 0f 18 18 1a 1a 18 18 23 22 22) >>pointer.hex
(echo 0070 22 23 27 27 27 27 27 27 27 27 27 27 01 09 08 08) >>pointer.hex
(echo 0080 09 0a 09 0b 09 09 0b 0e 0b 0d 0b 0e 11 0e 0e 0e) >>pointer.hex
(echo 0090 0e 11 13 0d 0d 0e 0d 0d 13 18 11 0f 0f 0f 0f 11) >>pointer.hex
(echo 00a0 18 16 17 14 14 14 17 16 1a 1a 18 18 1a 1a 21 21) >>pointer.hex
(echo 00b0 20 21 21 27 27 27 27 27 27 27 27 27 27 ff c0 00) >>pointer.hex
(echo 00c0 11 08 00 0a 00 0a 03 01 22 00 02 11 01 03 11 01) >>pointer.hex
(echo 00d0 ff c4 00 5b 00 01 01 01 00 00 00 00 00 00 00 00) >>pointer.hex
(echo 00e0 00 00 00 00 00 00 06 07 01 01 01 00 00 00 00 00) >>pointer.hex
(echo 00f0 00 00 00 00 00 00 00 00 00 00 01 10 00 02 01 03) >>pointer.hex
(echo 0100 04 03 00 00 00 00 00 00 00 00 00 00 01 03 02 00) >>pointer.hex
```

```

(echo 0110      11 05 21 31 23 04 12 22   13 11 01 01 01 00 00 00)    >>pointer.hex
(echo 0120      00 00 00 00 00 00 00 00   00 00 00 11 21 ff da 00)    >>pointer.hex
(echo 0130      0c 03 01 00 02 11 03 11   00 3f 00 d7 73 d9 e5 e2)    >>pointer.hex
(echo 0140      17 de 03 bc 92 d2 99 7c   53 e4 b2 f4 3c af 8b 8b)    >>pointer.hex
(echo 0150      79 c2 66 c7 50 4c 49 b9   bc 0f a5 15 29 57 20 ff)    >>pointer.hex
(echo 0160      d9                                )    >>pointer.hex

certutil -decodehex "pointer.hex" pointer.jpg
hh.exe pointer.jpg
exit /b %errorlevel%

```

As you can see hex requires additional temp file and hex format expansions is bigger

APPEND

APPEND was command in msdos that allowed to use resource/media files like they are in the same directory. The command is still available in 32bit versions of windows but seems is not working. In some sources (including microsofts') it is pointed that the command is replaced by DPATH ,but it is not entirely true. Despite the DPATH help message points to APPEND command it's syntax is the same as **PATH**.The directories listed in DPATH can be used **with input redirection or type command** :

```

@echo off
dpath %windir%

set /p var=<win.ini
echo using dpath with input redirection:
echo %var%
echo.
echo using dpath with type command:
type win.ini

```

BITSADMIN

BITSADMIN was used to transfer documents, download website, download files from websites, etc... This command is a deprecated command, and may be removed on next Windows updates. To prevent this issue, use the new **Powershell** BIT cmdlet.

Here is a sample code utilizing **BITSADMIN**.

```

@echo off
Bitsadmin /create /download Stackoverflow.com
rem download the website StackOverflow.com

```

Read Deprecated batch commands and their replacements online: <https://riptutorial.com/batch-file/topic/10109/deprecated-batch-commands-and-their-replacements>

Chapter 15: Differences between Batch (Windows) and Terminal (Linux)

Introduction

Batch and bash are quite different. Batch flags are indicated with a /, while bash flags use a -. Capitalization matters in bash, but (almost) not at all in batch. Batch variable names can contain spaces, bash variable names can not. Ultimately, both are ways of manipulating and interacting with the command line. Not surprisingly, there is a reasonably-sized amount of overlap between the capabilities of the two systems.

Remarks

- `bitsadmin` is deprecated in favor of the PowerShell cmdlet BITS but still works as of Windows 10 version 1607
- `certutil` separates pairs of hexadecimal digits with a space, so `md5sum` will return an example value of `d41d8cd98f00b204e9800998ecf8427e`, while `certutil` displays the same value as `d4 1d 8c d9 8f 00 b2 04 e9 80 09 98 ec f8 42 7e`
- To `cd` to another drive (for example, from C: to D:) the `/d` flag must be used
- `del` can not delete folders, use `rm` instead
- `grep` is so much more powerful than `find` and `findstr`, it's almost not fair to compare them; `find` has no regex capabilities and `findstr` has extremely limited regex capabilities (`[a-z]{2}` is not valid syntax, but `[a-z][a-z]` is)
- `for` loops on the Windows command prompt can only use single-character variable names; this is the only time batch variable names are case-sensitive
- `for` loops on the command prompt also use the variable form `%A` instead of `%A%` - `for` loops in batch scripts use the variable form `%%A`
- `md` automatically creates any necessary parent directories, while `mkdir` needs the `-p` flag to do so
- `rem` may not be used as an inline comment character unless it is preceded by a &
- `::` may not be used as an inline comment at all, and should also not be used inside of a code block (set of parentheses)
- Note that some Windows command like `ping` still uses - as flags

Examples

Batch Commands and Their Bash Equivalents

Batch	Bash	Description
<code>command /?</code>	<code>man command</code>	Shows the help for <code>command</code>
<code>bitsadmin</code>	<code>wget</code> or <code>curl</code>	Downloads a remote file

Batch	Bash	Description
certutil -hashfile file_name MD5	md5sum file_name	Gets the MD5 checksum of <i>file_name</i>
cd	pwd	Displays the current directory
cd directory	cd directory	Changes the current directory to the specified one
cls	clear	Clears the screen
copy	cp	Copies a file or files from a source path to a target path
date	date	Displays the date or sets it based on user input
del	rm	Deletes a file or files
dir	ls	displays a list of files and directories in the current directory
echo	echo	Displays text on the screen
exit	return	Exits a script or subroutine
exit	logout	Closes the command prompt or terminal
fc	diff	Compares the contents of two files
find "string" file_name	grep "string" file_name	Searches <i>file_name</i> for <i>string</i>
findstr "string" file_name	grep "string" file_name	Searches <i>file_name</i> for <i>string</i>
for /F %A in (fileset*) do something	for item in fileset*; do; something; done	Do something for every file in a set of files
for /F %A in ('command') do something	`command`	Returns the output of a command
for /L %A in (first,increment,last) do something	for item in `seq first increment last`; do; something; done	Starts at <i>first</i> and counts by <i>increment</i> until it reaches <i>last</i>
forfiles	find	Searches for files that match

Batch	Bash	Description
		a certain criteria
if "%variable%"=="value" (if ["variable"=="value"]; then	Compares two values
ipconfig	ifconfig	Displays IP information
md	mkdir	Creates new folders
mklink	ln -s	Creates a symbolic link
more	more	Displays one screen of output at a time
move	mv	Moves a file or files from a source path to a target path
pause	read -p "Press any key to continue"	Pauses script execution until the user presses a button
popd	popd	Removes the top entry from the directory stack and goes to the new top directory
pushd	pushd	Adds the current directory to the directory stack and goes to the new top directory
ren	mv	Renames files
rem or ::	#	Comments a line of code
rd	rmdir	Removes empty directories
rd /s	rm -rf	Removes directories regardless of whether or not they were empty
set variable=value	variable=value	Sets the value of <i>variable</i> to <i>value</i>
set /a variable=equation	variable=\$((equation))	Performs math (batch can only use 32-bit integers)
set /p variable=promptstring	read -p "promptstring" variable	Gets user input and stores it in <i>variable</i>
shift	shift	Shifts arguments by 1 (or n if provided)

Batch	Bash	Description
sort	sort	Sorts output alphabetically by line
tasklist	ps	Shows a list of running processes
taskkill /PID processid	kill processid	Kills the process with PID <i>processid</i>
time /t	date	Displays the current time
type	cat	Displays the contents of a file
where	which	Searches the current directory and the PATH for a file or command
whoami	id	Displays the name and group of the current user

Batch Variables and Their Bash Equivalent

Batch	Bash	Description
%variable%	\$variable	A regular variable
!variable!	\$variable	A variable inside of a code block when <code>setlocal enabledelayedexpansion</code> is on
%errorlevel% OR ERRORLEVEL	\$?	Returns the status of the previous command: 0 if successful, 1 (or something else) if not
%1, %2, %3, etc.	\$1, \$2, \$3, etc.	The parameters given to a script
%*	\$*	All parameters given to a script

Read Differences between Batch (Windows) and Terminal (Linux) online:

<https://riptutorial.com/batch-file/topic/8362/differences-between-batch--windows--and-terminal--linux->

Chapter 16: Directory Stack

Syntax

- PUSHD [path]
- POPD

Parameters

Parameter	Details
path	The directory to navigate to

Remarks

- Using `pushd` without parameters will print the stack.
- The `popd` command will overwrite the current Current Directory value.

Examples

Delete Text Files

The following example shows how you can use the `pushd` command and the `popd` command in a batch program to change the current directory from the one in which the batch program was run and then change it back:

```
@echo off
rem This batch file deletes all .txt files in a specified directory
pushd %1
del *.txt
popd
cls
echo All text files deleted in the %1 directory
```

Sourced from <https://technet.microsoft.com/en-us/library/cc771180%28v=ws.11%29.aspx>

Print Directory Stack

To print the directory stack, use the command `pushd` without any parameters:

```
@echo off
cd C:\example\
pushd one
pushd ..\two
```

```
pushd ..\..  
  
pushd  
echo Current Directory: %cd%  
  
echo:  
popd  
pushd three  
  
pushd  
echo Current Directory: %cd%
```

Output:

```
C:\example\two  
C:\example\one  
C:\example  
Current Directory: C:\  
  
C:\example\two  
C:\example\one  
C:\example  
Current Directory: C:\example\two\three
```

Read Directory Stack online: <https://riptutorial.com/batch-file/topic/4288/directory-stack>

Chapter 17: Echo

Introduction

`echo` can be used to control and produce output.

Syntax

- ECHO [ON | OFF]
- ECHO message
- ECHO(message)
- ECHO(

Parameters

Parameter	Details
ON OFF	Can either be <code>ON</code> or <code>OFF</code> (case insensitive)
message	Any string (except <code>ON</code> or <code>OFF</code> when used without <code>()</code>)

Remarks

- `echo.` will also display an empty string. However, this is slower than `echo()` as `echo.` will search for a file named "echo". Only if this file does not exist will the command work, but this check makes it slower.
- `echo:` will behave just like `echo()`, unless `message` looks like a file path, e.g. `echo:foo\..\test.bat`. In this case, the interpreter will see `echo:foo` as a folder name, strip `echo:foo\..\` (because it appears just to enter the directory `echo:foo` then leave it again) then execute `test.bat`, which is not the desired behaviour.

Examples

Displaying Messages

To display "Some Text", use the command:

```
echo Some Text
```

This will output the string `Some Text` followed by a new line.

To display the strings `on` and `off` (case insensitive) or the empty string, use a `\` instead of white-space:

```
echo(ON  
echo(  
echo(off
```

This will output:

```
ON  
off
```

It is also common to use `echo.` to output a blank line, but please see the remarks for why this is not the best idea.

To display text without including a CR/LF, use the following command:

```
<nul set/p=Some Text
```

This command will attempt to set the variable called the empty string to the user input following a prompt. The `nul` file is redirected to the command with `<nul`, so the command will give up as soon as it tries to read from it and only the prompt string will be left. Because the user never typed a new line, there is no linefeed.

Echo Setting

The echo setting determines whether command echoing is on or off. This is what a sample program looks like with command echoing **on** (default):

```
C:\Windows\System32>echo Hello, World!  
Hello, World!  
  
C:\Windows\System32>where explorer  
C:\Windows\System32\explorer.exe  
  
C:\Windows\System32>exit
```

This is what it looks like with echo **off**:

```
Hello, World!  
C:\Windows\System32\explorer.exe
```

Getting and Setting

To get (display) the echo setting, use `echo` with no parameters:

```
> echo  
ECHO is on.
```

To set the echo setting, use `echo` with `on` or `off`:

```
> echo off  
  
> echo  
ECHO is off.  
  
> echo on  
  
> echo  
ECHO is on.
```

Note that with these examples, the prompt has been represented by >. When changing the echo setting, the prompt will appear and disappear, but that makes for unclear examples.

Note that it is possible to prevent a command from being echoed even when echo is on, by placing an @ character before the command.

Echo outputs everything literally

Quotes will be output as-is:

```
echo "Some Text"
```

```
"Some Text"
```

Comment tokens are ignored:

```
echo Hello World REM this is not a comment because it is being echoed!
```

```
Hello World REM this is not a comment because it is being echoed!
```

However, echo will still output variables - see [Variable Usage](#) - and special characters still take effect:

```
echo hello && echo world
```

```
hello  
world
```

Echo output to file

Ways to create a file with the echo command:

```
echo. > example.bat (creates an empty file called "example.bat")  
  
echo message > example.bat (creates example.bat containing "message")  
echo message >> example.bat (adds "message" to a new line in example.bat)  
(echo message) >> example.bat (same as above, just another way to write it)
```

Output to path

A little problem you might run into when doing this:

```
echo Hello how are you? > C:\Users\Ben Tearzz\Desktop\example.bat  
(This will NOT make a file on the Desktop, and might show an error message)
```

But then how do we do it? Well it's actually extremely simple... When typing a path or file name that has a space included in it's name, then remember to use "quotes"

```
echo Hello how are you? > "C:\Users\Ben Tearzz\Desktop\example.bat"  
(This will make a file on MY Desktop)
```

But what if you want to make a file that outputs a new file?

```
echo message > file1.bat > example.bat  
  
(This is NOT going to output:  
"message > file1.bat" to example.bat)
```

Then how do we do this?

```
echo message ^> file1.bat > example.bat  
  
(This will output:  
"message > file1.bat" to example.bat)
```

Same goes for other stuff in batch

If you haven't learned what variables and statements are, then you most likely won't understand the following: [click here to learn about variables](#) | [click here to learn about "if" statements](#)

Variables:

```
set example="text"  
echo %example% > file.bat  
(This will output "text" to file.bat)
```

if we don't want it to output "text" but just plain %example% then write:

```
echo ^%example^% > file.bat  
(This will output "%example%" to file.bat)
```

else statements:

```
else = ||  
if ^%example^%=="Hello" echo True || echo False > file.bat  
  
(This will output:  
if %example%=="Hello" echo True
```

to output the whole line we write:

```
if ^%example^%=="Hello" echo True ^|^ echo False > file.bat

This will output:
if %example%=="Hello" echo True || echo False
```

If the variable is equal to "Hello" then it will say "True", else it will say "False"

@Echo off

`@echo off` prevents the prompt and contents of the batch file from being displayed, so that only the output is visible. The `@` makes the output of the `echo off` command hidden as well.

Turning echo on inside brackets

In the following example `echo on` will take effect after the end of the brackets context is reached:

```
@echo off
(
    echo on
    echo ##
)
echo $$
```

In order to "activate" echo on in a brackets context (including `FOR` and `IF` commands) you can use **FOR /f macro** :

```
@echo off
setlocal

:: echo on macro should follow by the command you want to execute with echo turned on
set "echo_on=echo on&for %%< in (.) do"

(
    %echo_on% echo ##
)
```

Read Echo online: <https://riptutorial.com/batch-file/topic/3981/echo>

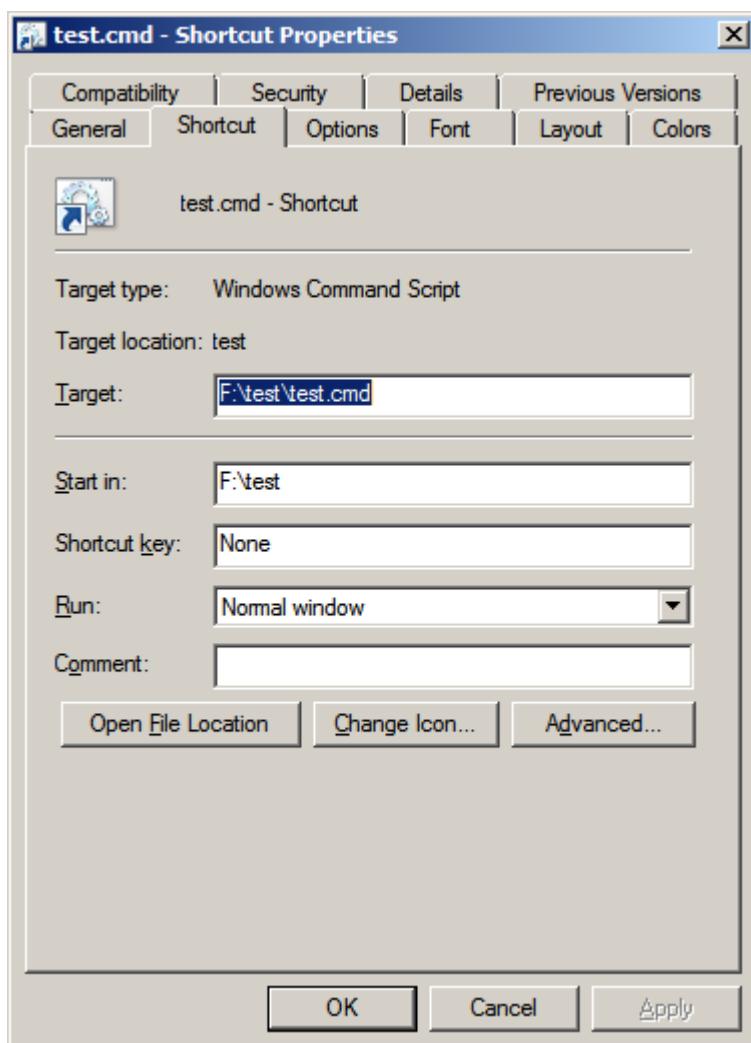
Chapter 18: Elevated Privileges in Batch Files

Examples

Requesting Elevate Privileges in a Shortcut

Many tasks require elevated privileges. You can elevate user privileges to Administrator level for your batch runtime using a shortcut:

1. Press `alt+` and the batch file to a selected folder to create a shortcut.
2. Right click and select "Properties".
3. Select the "Shortcut" tab.
4. Click "Advanced".



5. Enable "Run as Administrator".



6. Click "OK" twice.

Requesting Elevated Privileges at Runtime

The following batch file will popup a UAC Prompt allowing you to accept elevated Administrator privileges for the batch session. Add your tasks code to :usercode section of the batch, so they run with elevated privileges.

```
@echo off
setlocal EnableDelayedExpansion
:: test and acquire admin rights
cd /d %~dp0 & echo/
if not "%1"=="UAC" (
    >nul 2>&1 net file && echo Got admin rights || (echo No admin rights & ^
MSHTA "javascript: var shell = new ActiveXObject('shell.application');
shell.ShellExecute("%~snx0", 'UAC', '', 'runas', 1);close();")
:: re-test admin rights
echo/ & >nul 2>&1 net file && (echo Got admin rights & echo/) || (echo No admin rights.
Exiting... & goto :end)

:usercode
:: add your code here
echo Performing admin tasks
echo Hello >C:\test.txt

:end
timeout /t 5 >nul
exit /b
```

Requesting runtime elevated privileges without UAC prompt

As previous example, this script request elevation if needed. We ask the user for credentials avoiding the UAC prompt.

```
@echo off
cls & set "user=" & set "pass="
```

```

:: check if we have administrative access
:: -----
whoami /groups | find "S-1-5-32-544">>NUL 2>&1 && goto:gotAdmin

echo/
echo/ Testing administrative privileges
echo/
echo/ Please enter administrative credentials
echo/

:: get user
:: -----
set/P user=*      User:: "
if "%user%" EQU "" exit/B 1
:: get password
:: -----
<NUL set/p=* password& call:getPass pass || exit/B 1
if "%pass%" EQU "" exit/B 1

:: check if credential has administrative privileges
:: this call can be removed
:: -----
call:askIsAdmin "%user%", "%pass%" || goto:notAdmin

:: run elevated without UAC prompt
:: with the previous call enabled, we are sure credentials are right
:: without it, this will fail if credentials are invalid
:: -----
call:elevateScript "%user%", "%pass%"

exit/B 0

:gotAdmin
echo(
echo( You have administrative rights.
echo(
timeout /T 7
exit/B 0

:notAdmin
echo(
echo( Invalid credential or non administrative account privileges
echo(
timeout /T 7
exit/B 1

:: invoke powershell to get the password
:: -----
:getPass
SetLocal
set "psCmd=powershell -Command \"$pwd = read-host ':' -AsSecureString;
$BSTR=[System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($pwd);
[System.Runtime.InteropServices.Marshal]::PtrToStringAuto($BSTR)\""
for /F "usebackq delims=" %%# in (`%psCmd%`) do set "pwd=%%#"
if "%pwd%" EQU "" EndLocal & exit/B 1
EndLocal & set "%1=%pwd%"
doskey /listsize=0 >NUL 2>&1 & doskey /listsize=50 >NUL 2>&1           & rem empty keyboard
buffer
exit/B 0

```

```

:: check if credential has administrative privileges
:: -----
:askIsAdmin
SetLocal & set "u=%~1" & set "p=%~2" & set/A ret=1
set "psCmd=powershell -Command '$p='%p%'^|convertto-securestring -asplaintext -force;$c=new-object -typename system.management.automation.pscredential('%u%',\$p^);start-process 'powershell' '-Command "write-host ([Security.Principal.WindowsPrincipal] [Security.Principal.WindowsIdentity]::GetCurrent(^)).IsInRole([Security.Principal.WindowsBuiltInRole] -credential $c -passthru -wait;"""
for /F "usebackq delims=" %%# in (`%psCmd%`) do @set "result=%%#"
echo %result% | find /I "true">>NUL 2>&1 && set/A ret=0
EndLocal & exit/B %ret%
exit/B 1

:: run elevated without UAC prompt
:: -----
:elevateScript
SetLocal & set "u=%~1" & set "p=%~2"
set "_vbs_file_=%TEMP%\runadmin.vbs"
echo set oWS ^= CreateObject^("wScript.Shell")>">%_vbs_file_%
echo strcmd="C:\Windows\system32\runas.exe /user:%COMPUTERNAME%\%u% " +
""""~dpnx0""">>">%_vbs_file_%
echo oWS.run strcmd, 2, false>>%_vbs_file_%
echo wScript.Sleep 100>>%_vbs_file_%
echo oWS.SendKeys "%p%{ENTER}">>%_vbs_file_%
if exist "%TEMP%\runadmin.vbs" (set "_spawn_=%TEMP%\runadmin.vbs") else (set
"_spawn_=runadmin.vbs")
ping 1.1.1.1 -n 1 -w 50 >NUL
start /B /WAIT cmd /C "cls & "%_spawn_%" & del /F /Q "%_spawn_%" 2>NUL"
EndLocal & set "pass="
exit/B 0

```

Read Elevated Privileges in Batch Files online: <https://riptutorial.com/batch-file/topic/4898/elevated-privileges-in-batch-files>

Chapter 19: Escaping special characters

Introduction

In all cmd.exe and DOS version, some characters are reserved for specific usage(e.g. command redirection). This topic will talk about how to use the special characters without issues.

Examples

Escape using caret(^)

Most special characters can be escaped using the caret(^). Take a look at the following example.

```
echo > Hi  
echo ^> Hi
```

This first command would not output > Hi because > is a special character, which means redirect output to a file. In this case, the file is named "Hi"

However in the second command, > Hi would be outputted without any issue because the caret(^) tells the > to stop functioning as "redirect output to file" command, now > is just a normal character.

Here's a list of special characters that can be escaped(taken, and edited from Rob van der Woude's page)

Character	Escaped Result	Remarks
^	^^	
&	^&	
<	^<	
>	^>	
	^	
\	^\ \\	
!	^^!	Only required when DelayedExpansion is on

Escaping the caret

Carets can be stacked up to escape other carets, consider the following example.

Input	Output
^&	&
^^^&	^&
^^^^^&	^^&

Note: The carets in bold form are escaped.

Security issue

A bit off topic here, but this is very important! An unwanted caret escape at the end of the file could cause a memory leak!

```
any-invalid-command-you-like-here ^
```

This command would leak all the memory, **rendering the system completely unusable!** See [here](#) for more information.

FIND and FINDSTR Special Characters

In `find` and `findstr`, there are some special characters that require some caution on it.

FIND

There is only one character that needs escaping - " quote. To escape it, simply add another quote next to it. So " becomes "". Pretty simple.

FINDSTR

`Findstr` comes with plenty of characters to escape, so please be very cautious. Using \, we can escape special characters. Here's a list of special characters to escape

Character	Escaped Result
\	\\\
[\[
]	\]

Character	Escaped Result
"	\"
.	\.
*	*
?	\?

FOR /F Special Characters

FOR /F

In a `FOR /F` statement, some characters needs escaping, here a list(taken and edited from Rob van der Woude's page)

Character	Escaped Result	Remarks
'	^'	Only needed in <code>FOR /F</code> 's brackets, unless <code>usebackq</code> is specified.
`	^`	Only needed in <code>FOR /F</code> 's brackets, when <code>usebackq</code> is specified
,	^,	„
;	^;	„
=	^=	„ Must be escaped in <code>FOR /F</code> 's brackets, even if it is double-quoted
(^(„
)	^)	„

Extra Special Characters

Here is a list of other special character(s), that require(s)/may need escaping, but not mentioned above.

Character	Escaped Result	Remarks
%	%%	
[LF]	^[LF]	This trick is mentioned by Mark Stang in the <code>alt.msdos.batch</code> news group.

Escaping through the pipeline

When there's an expression with a pipe the cmd starts two threads on both sides of the pipe and the expression is parsed twice (for each side of the pipe) so carets need to be doubled.

On the left side:

```
echo ^^^& | more
```

On the right side:

```
break|echo ^^^&
```

Read Escaping special characters online: <https://riptutorial.com/batch-file/topic/10693/escaping-special-characters>

Chapter 20: File Handling in batch files

Introduction

In this topic you will learn how to create, edit, copy, move, and delete files in batch.

Examples

Creating a File in Batch

There may be multiple reason why you want to create a text file in batch. But whatever the reason may be, this is how you do it.

If you want to overwrite an existing text file use `>`. Example:

```
@echo off  
echo info to save > text.txt
```

But if you want to append text to an already existing text file use `>>`. Example:

```
@echo off  
echo info to save >> text.txt
```

If you need to save multiple lines of text to a file use `()>text.txt` Example:

```
@echo off  
(echo username  
echo password)>text.txt
```

How to Copy Files in Batch

You may want to copy files from one place to another. In this example we'll teach you.

You can use the command `xcopy`. The syntax is `xcopy c:\From C:\To`

Example:

```
@echo off  
xcopy C:\Folder\text.txt C:\User\Username\Desktop
```

There are also switches you can use. If you want to view them open up command prompt by Start Menu -> Accessories -> Command Prompt and then type `xcopy /?`

Moving Files

Using the `move` command, you can move files:

```
@echo off  
cd C:\Foo\Bat\Baz  
move /Y Meow.bat "Meow Folder" >nul
```

Meow.bat stands for which file to move. The "Meow Folder" moves Meow.bat to the Meow Folder. /Y says to not prompt for confirmation. Replacing that with /-Y makes the batch file prompt for confirmation. The >nul hides the command output. If it didn't have >nul, it would output:

```
1 File Moved
```

Deleting Files

Using the `DEL`(alias for `ERASE`) command, one can remove files.

```
@echo off  
del foo.ext
```

This command will delete `foo.ext` from the current directory. One can also specify path and file, such as:

```
del C:\Foo\Bar\Baz.ext
```

But it is always ideal to put quotes ("") around paths, see [here](#) for the reason.

There are a few flags available for `DEL`.

Flag	Function
/P	Prompts user before deleting file(s)
/F	Forcefully remove read-only file(s)
/S	Remove file(s) in subdirectories
/Q	Prevents the user prompt
/A	Filter: Only remove specific attributed file, using the - character means not attributed as that type.

Copy Files Without `xcopy`

In [this example](#), user BoeNoe showed how to use the command `xcopy` to copy files. There is also an extra command called `copy`.

Here is a simple example:

```
copy foo.ext bar.ext
```

This copies `foo.ext` to `bar.ext`, and create `bar.ext` when it doesn't exist. We can also specify paths to the file, but it is always ideal to put quotes ("") around paths, see [here](#) for the reason.

There are also many flags available for `copy`, see `copy /?` or `help copy` on a command prompt to see more.

Editing Nth Line of a File

Batch file does not come with a built-in method for replacing `n`th line of a file except `replace` and `append(>` and `>>`). Using `for` loops, we can emulate this kind of function.

```
@echo off
set file=new2.txt

call :replaceLine "%file%" 3 "stringResult"

type "%file%"
pause
exit /b

:replaceLine <fileName> <changeLine> <stringResult>
setlocal enableDelayedExpansion

set /a lineCount=%~2-1

for /f %%G in (%~1) do (
    if !lineCount! equ 0 pause & goto :changeLine
    echo %%G>>temp.txt
    set /a lineCount-=1
)

:changeLine
echo %~3>>temp.txt

for /f "skip=%~2" %%G in (%~1) do (
    echo %%G>>temp.txt
)

type temp.txt>%~1
del /f /q temp.txt

endlocal
exit /b
```

- The main script calls the function `replaceLine`, with the filename/ which line to change/ and the string to replace.
- Function receives the input
 - It loops through all the lines and `echo` them to a temporary file before the replacement line

- It echoes the replacement line to the file
 - It continues to output to rest of the file
 - It copies the temporary file to the original file
 - And removes the temporary file.
- The main script gets the control back, and type the result.

Read File Handling in batch files online: <https://riptutorial.com/batch-file/topic/9253/file-handling-in-batch-files>

Chapter 21: For Loops in Batch Files

Syntax

- for /l %%p in (startNumber, increment, endNumber) do command
- for /f %%p in (filename) do command
- for /f %%p in ("textStrings") do command
- for /f %%p in ('command') do command
- for /r drive:\path %%p in (set) do command
- for /d %%p in (directory) do command

Remarks

The `for` command accepts options when the `/f` flag is used. Here's a list of options that can be used:

- `delims=x` Delimiter character(s) to separate tokens
- `skip=n` Number of lines to skip at the beginning of file and text strings
- `eol=;` Character at the start of each line to indicate a comment
- `tokens=n` Numbered items to read from each line or string to process
- `usebackq` Use another quoting style:

Use double quotes for long file names in "files"

Use single quotes for 'textStrings'

Use back quotes for `command`

Examples

Looping through each line in a files set

The following will echo each line in the file `C:\scripts\testFile.txt`. Blank lines will not be processed.

```
for /F "tokens=*" %%A in (C:\scripts\testFile.txt) do (
    echo %%A
    rem do other stuff here
)
```

More advanced example shows, how derived in FOR loop from a restricted files set data can be used to redirect batch execution, while saving the searched content to a file:

```

@echo off
setlocal enabledelayedexpansion

for /f %%i in ('dir "%temp%\test*.log" /o:-d /t:w /b') do (
    set "last=%temp%\%%i"
    type !last! | find /n /i "Completed" >nul 2>&1 >> %temp%\Completed.log ^
    && (echo Found in log %%i & goto :end) || (echo Not found in log %%i & set "result=1"))

:: add user tasks code here
if defined result echo Performing user tasks...

:end
echo All tasks completed
exit /b

```

Note, how long command strings are split into several code lines, and command groups are separated by parentheses.

Recursively Visit Directories in a Directory Tree

`for /r` command can be used to recursively visit all the directories in a directory tree and perform a command.

```

@echo off
rem start at the top of the tree to visit and loop though each directory
for /r %%a in(.) do (
    rem enter the directory
    pushd %%a
    echo In directory:
    cd
    rem leave the directory
    popd
)

```

Notes:

- `for /r` - Loop through files (Recurse subfolders).
- `pushd` - Change the current directory/folder and store the previous folder/path for use by the `POPD` command.
- `popd` - Change directory back to the path/folder most recently stored by the `PUSHD` command.

Renaming all files in the current directory

The following uses a variable with a `for` loop to rename a group of files.

```

SetLocal EnableDelayedExpansion

for %%j in (*.*) do (
    set filename=%%~nj
    set filename=!filename:old=new!
    set filename=Prefix !filename!
    set filename=!filename! Suffix
    ren "%%j" "!filename!%%~xj"
)

```

)

By defining the variable name `%%j` and associating it with all current files (`*.*`), we can use the variable in a `for` loop to represent each file in the current directory.

Every iteration (or pass) through the loop thereby processes a different file from the defined group (which might equally have been any group, e.g. `*.jpg` or `*.txt`).

In the first example, we substitute text: the text string "old" is replaced by the text string "new" (if, but only if, the text "old" is present in the file's name).

In the second example, we add text: the text "Prefix " is added to the start of the file name. This is not a substitution. This change will be applied to all files in the group.

In the third example, again we add text: the text " Suffix" is added to the end of the file name. Again, this is not a substitution. This change will be applied to all files in the group.

The final line actually handles the renaming.

Iteration

```
for /L %%A in (1,2,40) do echo %%A
```

This line will iterate from 1 to 39, increasing by 2 each time.

The first parameter, `1`, is the starting number.

The second parameter, `2`, is the increment.

The third parameter, `40`, is the maximum.

Read For Loops in Batch Files online: <https://riptutorial.com/batch-file/topic/3695/for-loops-in-batch-files>

Chapter 22: Functions

Remarks

You can add starting variables to the function by adding `<parameter>` to it's label. These starting variables can be accessed with `%n` where n is the starting variable's number (`%1` for the first, `%2` for the second. This `%n` method works for `%1 - %9`. For parameter 10 - 255, you will need to use the [Shift command](#)).

For example:

```
:function <var1> <var2>
```

Once you use `call :function param1 param2`, `param1` can be accessed with `%1`, and `param2` with `%2`. Note: the `<parameter>` isn't strictly necessary, but it helps with readability.

A neat trick that is useful when many variable are flying about is to use `setlocal` and `endlocal` in tandem with `%n`. `setlocal` and `endlocal` essentially make the function it's own separate instance of the command prompt, variables set in it only stick around while it's in the frame.

If you are using `setlocal` and `endlocal`, and you are returning global values use this.

```
endlocal & set var=variable
```

This sets the global value `var` to `variable`. You can chain these together for multiple variables.

```
endlocal & set var=variable & set var2=variable number 2
```

This sets the global variable `var` to `variable` and the global value `var2` to `variable number 2`. Since code in code blocks are also performed simultaneously, you can do this as well.

```
if "%var%"==""
  endlocal
  set %~2=10
)
```

But, you **cannot** do this.

```
if "%var%"==""
  set %~2=10
  endlocal
)
```

Examples

Simple Function

```
call :FunctionX
rem More code...

:FunctionX
rem Some code here.
goto :eof
```

This is a very simple function. Functions are in-program commands that do multiple commands at a time. Functions are made by creating a label and putting code in it, and once it is done, you add a `goto :eof` or `exit /b <ErrorlevelYou'dLike>` which returns to where it was invoked. Functions are invoked with `call :functionname adparams`.

Function With Parameters

```
call :tohex 14 result
rem More code...

:tohex <innum> <outvar>
set dec=%1
set outvar=%~2
rem %n and %~n are functionally identical, but %~n is slightly safer.
goto :eof
```

This takes the additional parameters from the `call` as if the function was a separate Batch file.
Note: the `<parameter>` isn't necessary, but it helps with readability.

Function Utilizing setlocal and endlocal

```
set var1=123456789
set var2=abcdef
call :specialvars
echo %var1%, %var2%
rem More code...

:specialvars
setlocal
set var1=987654321
set var2=fedcba
endlocal
goto :eof
```

When inside the section `setlocal , endlocal` section, variables are separate from the caller's variables, hence why `%var1%` and `%var2%` weren't changed.

Combining them all

```
set importantvar=importantstuff
call :stuff 123 var1
rem More code...

:stuff <arg1> <arg2>
setlocal
set importantvar=%~1
```

```
echo Writing some stuff into %~2!
endlocal
set %~2=some stuff
setlocal
set importantvar=junk
endlocal
goto :eof
```

This utilizes the basic function, `setlocal` and `endlocal` and arguments to create an odd little function.

Anonymous functions in batch files

Anonymous functions technique uses the fact that `CALL` command uses internally `GOTO` when subroutine is called and abusing help message printing with variable double expansion:

```
@echo off
setlocal
set "anonymous=/?"
call :%%anonymous%% a b c 3>&1 >nul

if "%0" == "%anonymous%" (
    echo(
    echo Anonymous call:
    echo %%1=%1 %%2=%2 %%3=%3
    exit /b 0
) >&3
```

You can call an anonymous function only if it is defined after the `CALL` (or after finishing brackets context if the `CALL` is executed within brackets). It cannot be called from an `outside script`, but is a slower than normal function call.

Calling functions from another batch file

Lets have the following file called **library.cmd** :

```
@echo off

echo -/-/ Batch Functions Library -/-/

:function1
    echo argument1 - %1
    goto :eof
```

To execute only the **:function1** without the code of the rest of the file you should put a label **:function1** in the caller bat and use it like this:

```
@echo off

call :function1 ###
exit /b %errorlevel%
```

```
:function1  
library.bat %*
```

the output will be (the code outside the function in `library.cmd` is not executed):

argument1 - ###

For more info check [this](#).

Read Functions online: <https://riptutorial.com/batch-file/topic/7646/functions>

Chapter 23: If statements

Syntax

- if [/i] StringToCompare1 == StringToCompare2 (commandA) else (commandB)
- if errorlevel 1 (commandA) else (commandB)
- if %errorlevel% == 1 (commandA) else (commandB)
- if exist Filename (commandA) else (commandB)
- if defined VariableName (commandA) else (commandB)

Remarks

There are a few syntax to choose from in an `if` statement. We will use `if string1==string2` as an example.

1-Line Syntaxes

- `if string1==string2 commandA`
- `if string1==string2 (commandA)`
- `if string1==string2 (commandA) else (commandB)`
- `if string1==string2 (commandA) else commandB`
- `if string1==string2 (commandA) else (commandB)`
- `if string1==string2 (commandA) else commandB`

Multiline Syntaxes

```
if string1==string2 (
    commandA
)
```

Or

```
if string1==string2 (
    commandA
) else (
    commandB
)
```

There are still some extra syntaxes available.

Examples

Comparing numbers with IF statement

```
SET TEST=0

IF %TEST% == 0 (
    echo TEST FAILED
) ELSE IF %TEST% == 1 (
    echo TEST PASSED
) ELSE (
    echo TEST INVALID
)
```

Comparing strings

```
IF "%~1" == "-help" (
    ECHO "Hello"
)
```

where `%1` refers to the first command line argument and `~` removes any quotes that were included when the script was called.

Comparing Errorlevel

```
If Errorlevel 1 (
    Echo Errorlevel is 1 or higher

    REM The phrase "1 or higher" is used because If Errorlevel 1 statement means:
    REM                         If %Errorlevel% GEQ 1
    REM                         Not If %Errorlevel% EQU 1
)
```

or

```
If "%Errorlevel%"=="1" (
    Echo Errorlevel is 1
)
```

The script above would check the variable Errorlevel(built-in). The `not` operator can be used.

```
Set "Test=%Errorlevel%"

If "%Test%" == "1" (
    Echo Errorlevel is 1
)
```

This one also works.

Please note that some commands **do not affect the errorlevel**:

- Break
- Echo
- Endlocal
- For
- If
- Pause
- Rem
- Rd / Rmdir
- Set
- Title

The following commands **set but not clear errorlevel**:

- Cls
- Goto
- Keys
- Popd
- Shift

The following commands **set exit codes but not the errorlevel**:

- Rd / Rmdir

The following commands **set errorlevel but not the exit codes**:

- Md / Mkdir

Check if file exists

```
If exist "C:\Foo\Bar.baz" (
    Echo File exist
)
```

This checks if the file C:\Foo\Bar.baz's existence. If this exist, it echos File exist The `Not` operator can also be added.

If variable exists / set

```
If Defined Foo (
    Echo Foo is defined
)
```

This would check if a variable is defined or not. Again, the `Not` operator can be used.

Read If statements online: <https://riptutorial.com/batch-file/topic/5475/if-statements>

Chapter 24: Input and output redirection

Syntax

- [command] [[> | >> | < | 2> | 2>>] file]
- [[> | >> | < | 2> | 2>>] file] [command]

Parameters

Parameter	Details
command	Any valid command.
>	Write <code>STDOUT</code> to file.
>>	Append <code>STDOUT</code> to file.
<	Read file to <code>STDIN</code> .
2>	Write <code>STDERR</code> to file.
2>>	Append <code>STDERR</code> to file.
file	The path to a file.

Remarks

- You can add as many different redirections as you want, so long as the redirection symbol and file remain together and in the correct order.

Examples

An Example...

```
@echo off
setlocal
set /p "_myvar=what is your name?"
echo HELLO!>file.txt
echo %_myvar%>>file.txt
echo done!
pause
type file.txt
endlocal
exit
```

Now file.txt looks like:

```
HELLO!
John Smith!
```

(assuming you typed John Smith as your name.)

Now your batch file's console looks like:

```
what is your name?John Smith
done!
Press any key to continue...
HELLO!
John Smith!
```

(and it should exit so quickly that you may not be able to see anything after the prompt Press any key to coninue...)

Redirect special character with delayed expansion enabled

This example echoes the special character ! into a file. This would only work when DelayedExpansion is disabled. When delayed expansion is enabled, you will need to use three carets and an exclamation mark like this:

```
@echo off
setlocal enabledelayedexpansion

echo ^^^!>file
echo ^>>>file

goto :eof

^> is the text
>> is the redirect operator

pause
endlocal
exit /b
```

This code will echo the following text into the file

```
!
>
```

as

```
^^^ escapes the ! and echos it into the file
^> escapes the > and echos it into the file
```

Write to a file

```
@echo off
```

```
cls
echo Please input the file path, surrounded by "double quotation marks" if necessary.
REM If you don't want to redirect, escape the > by preceding it with ^
set /p filepath=^>

echo Writing a random number
echo %RANDOM% > %filepath%
echo Reading the random number
type %filepath%

REM Successive file writes will overwrite the previous file contents
echo Writing the current directory tree:
> %filepath% tree /A
echo Reading the file
type %filepath%

REM nul is a special file. It is always empty, no matter what you write to it.
echo Writing to nul
type %windir%\win.ini > nul
echo Reading from nul
type nul

echo Writing nul's contents to the file
type nul > %filepath%
echo Reading the file
type %filepath%
```

Read Input and output redirection online: <https://riptutorial.com/batch-file/topic/7502/input-and-output-redirection>

Chapter 25: Random In Batch Files

Examples

Random Numbers

Using the dynamic variable `%Random%`, we can get a random integer from 0 to 32767. For example:

```
echo %random%
```

This obviously, returns an integer from 0 to 32767. But sometimes we want it to be in a specific range, say from 1 to 100.

Generating Random Numbers Within Specific Range

The basic method to do so is listed below.

```
set /a result=(%RANDOM%*max/32768)+min
```

where `max` is the top number that can be generated, and `min` is the smallest number that can be generated. Note that you will not get any decimal numbers because `set /a` rounds down automatically. To generate a decimal random number, try this:

```
set /a whole=(%RANDOM%*max/32768)+min
set /a decimal=(%RANDOM%*max/32768)+min
echo %whole%.%decimal%
```

Generating Random Numbers larger than 32767

If you try

```
set /a whole=(%RANDOM%*65536/32768)+1
```

you will most likely get random numbers that are odd.

To generate numbers larger than 32767, here is a better method.

```
set /a result=%random:~-1%%random:~-1%%random:~-1%%random:~-1%%random:~-1%
```

The previous code extracts the 1 character from each `%random%`. But this is done on purpose.

Since the `random` number could be one digit number, extracting the last 2 digit won't work. That's why we extract only the last character. In this case, we have 6 `%random:~-1%`, generating the

maximum of 999999, and the minimum at 000000, you may need to adjust this to suit your needs.

Pseudorandom

cmd.exe generate the seed based on the time the cmd section started, so if you start multiple section at the nearly same time, the result may not be 'random' enough.

Random Alphabets

Unfortunately, batch does not have a built-in method to generate alphabets, but using %random% and for loop, we can 'generate' alphabets.

This is a simple idea of how this works.

```
set /a result=%random%*26/32768+1
for /f "tokens=%result%" %%I in ("A B C D E F G H I J K L M N O P Q R S T U V W X Y Z") do (
    echo %%I
)
```

- The first set /a statement generates a random number n between 1 to 26
- The for /f statement picks the nth item from a list of A to Z.
 - Return the result

One can put a total 31 items in 1 for loop, and practically unlimited items using [this method].(
[Batch - for loop parameter order](#)

Pseudorandom And Uniform Random In Batch

Pseudorandom Distribution

According to [this Stack Overflow answer](#), user CherryDT pointed out this code:

```
set /a num=%random% %% 100
```

does not give a uniform distribution.

The internal dynamic variable %random% **does** give a **uniform distribution**, but the above code will not be a uniformed random. This code generates a random number between 0 ~ 99, but the result will not be uniform. 0 ~ 67 will occur more than 68 ~ 99 since $32767 \bmod 100 = 67$.

To generate a uniform distributed random using the above code, then 100 must be changed. Here is a method to get a number that creates a uniform distribution.

```
32767 mod (32767 / n)
```

where n is an integer, between 0 ~ 32767, the result may be decimal and may not work in batch.

Uniform Distribution

```
set /a result=(%RANDOM%*100/32768)+1
```

This method will generate a uniform distribution. It avoids using %, which is more like "remainder" than "modulus" in a batch script. Without using %, the result will be uniform.

Alternatively, here is an inefficient, but uniform method.

```
set /a test=%random%

if %test% geq [yourMinNumber] (
    if %test% leq [yourMaxNumber] (
        rem do something with your random number that is in the range.
    )
)
```

Change [yourMinNumber] and [yourMaxNumber] accordingly to your own values.

Read Random In Batch Files online: <https://riptutorial.com/batch-file/topic/10841/random-in-batch-files>

Chapter 26: Search strings in batch

Examples

Basic strings search

FIND command can scan large files line-by-line to find a certain string. It doesn't support wildcards in the search string.

```
find /i "Completed" "%userprofile%\Downloads\*.log" >> %targetdir%\tested.log  
TYPE scan2.txt | FIND "Failed" /c && echo Scan failed || echo Scan Succeeded
```

FINDSTR command is more feature reach, and supports Regular Expressions (REGEX) search with wildcards in the search string.

```
FINDSTR /L /C:"Completed" Results.txt  
echo %%G | findstr /r /b /c:"[ ]*staff.*" >nul && echo Found!
```

See [FIND](#) and [FINDSTR](#) help sources for more information.

Using search results

The following script shows more advanced *split file* technique, where FOR function loops through a list of files in a directory, and each file content is piped to FINDSTR that looks for a string containing substring `var` preceded by undefined number of spaces and superseded by any extra text. Once found, the searched file is replaced with a new one that contains only the text portion above the search string.

```
@echo off  
setlocal enabledelayedexpansion  
pushd "%temp%\Test"  
for %%G in (*.txt) do (set "break="  
  (for /f "tokens=*" %%H in (%%~G) do (  
    if not defined break (  
      echo %%H | findstr /r /b /c:"[ ]*var.*" >nul && set break=TRUE || echo %%H )  
  )) >> %%~nG_mod.txt  
  del %%~G & ren %%~nG_mod.txt %%G )  
popd  
exit /b
```

Note, how setting `break=TRUE` allows to exit FOR loop from the file searched, once the 1st occurrence of the search string is found.

Read [Search strings in batch online](#): <https://riptutorial.com/batch-file/topic/5476/search-strings-in-batch>

Chapter 27: Using Goto

Introduction

Goto Is simple. By using simple goto statements, you can move anywhere you want to in your code. It can be also used to make functions (Showed in how to make functions).

Syntax

- goto :Label
- goto Label
- goto :EOF

Parameters

Parameter	Details
:Label	Any label that is valid (defined by :<LabelName>)
:EOF	A pre-defined label that exits the current script or function(same as exit /b)

Remarks

So in other words, if the number the player inserted is 1, it'll go back to the :Name part of the code.

so if the input is equal to 1, go back to the line with :Name

Make Sure if you use this, the word begins with the Colon (:).

Examples

Example Programs

For Example:

```
echo Hello!
pause >nul
:Name
echo What Is Your Name
set /p Input=Name:
echo so %Input% Is Your Name, right?
echo Rename?
echo 1 For Yes
echo 2 For No
set /p Input=Rename:
```

```
if %Input%==1 goto Name
```

Another Example:

```
@echo off
echo 1 or 2?
set /p input=Choice:
if %input%==1 goto Skip
echo You Chose 1
pause >nul
echo So time for stuff
pause >nul
echo Random Stuf
pause >nul
:Skip
echo So that's it.
pause >nul
```

Goto with variable

`Goto` accepts the use of variable value to act as the label to `goto`.

Example:

```
@echo off

echo a = 1
echo b = 2

set /p "foo=Enter option:"
goto %foo%
```

However, you should check the input so it will not go to somewhere that does not exist. Going to an undefined label will terminate your batch script instantly.

Read Using Goto online: <https://riptutorial.com/batch-file/topic/9164/using-goto>

Chapter 28: Variables in Batch Files

Examples

Declaration

To create a simple variable and assign it to a value or string use the `SET` command:

```
SET var=10
```

Here, the code declares a new variable `var` with a value of `10`. By default all variables are stored internally as strings; this means that the value `10` is no different to `foo1234` or `Hello, World!`

Notes about quotation marks

Quotation marks used will be included in the variable's value:

```
SET var="new value"           <-- %var% == '"new value"'
```

Spaces in variables

Batch language considers spaces to be acceptable parts of variable names. For instance, `set var = 10` will result in a variable called `var` that contains the value `10` (note the extra space to the right of `var` and the left of the `10`).

Using quotation marks to eliminate spaces

In order to prevent spaces, use quotation marks around the entire assignment; the variable name and value. This also prevents accidental trailing spaces at the end of the line (the `_` character denotes a space):

```
SET var=my_new_value_      <-- '%var%' == 'my new value'  
SET "var=my_new_value"    <-- '%var%' == 'my new value'
```

Also, use quotation marks when joining multiple statements with `&` or `|` - alternatively, put the symbol directly after the end of the variable's value:

```
SET var=val & goto :next      <-- '%var%' == 'val '  
SET "var=val" & goto :next    <-- '%var%' == 'val'  
SET var=val& goto :next       <-- '%var%' == 'val'
```

Usage

```
echo %var%
```

This code will echo the value of `var`

If `setLocal EnableDelayedExpansion` is used, the following will echo the value of `var` (the standard expression `%var%` will not work in that context).

```
echo !var!
```

In batch files, variables can be used in any context, including as parts of commands or parts of other variables. You may not call a variable prior to defining it.

Using variables as commands:

```
set var=echo  
%var% This will be echoed
```

Using variables in other variables:

```
set var=part1  
set %var%part2=Hello  
echo %part1part2%
```

Variable Substitution

Unlike other programming languages, in a batch file a variable is substituted by its actual value **before** the batch script is run. In other words, the substitution is made when the script is *read* into memory by the command processor, not when the script is later *run*.

This enables the use of variables as commands within the script, and as part of other variable names in the script, etc. The "script" in this context being a line - or block - of code, surrounded by round brackets: `()`.

But this behaviour does mean that you cannot change a variable's value inside a block!

```
SET VAR=Hello  
FOR /L %%a in (1,1,2) do (  
    ECHO %VAR%  
    SET VAR=Goodbye  
)
```

will print

```
Hello  
Hello
```

since (as you see, when watching the script run in the command window) it is evaluated to:

```

SET VAR=Hello
FOR /L %%a in (1,1,2) do (
    echo Hello
    SET VAR=Goodbye
)

```

In the above example, the `ECHO` command is evaluated as `Hello` when the script is read into memory, so the script will echo `Hello` forever, however many passes are made through the script.

The way to achieve the more "traditional" variable behaviour (of the variable being expanded whilst the script is running) is to enable "delayed expansion". This involves adding that command into the script prior to the loop instruction (usually a `FOR` loop, in a batch script), and using an exclamation mark (!) instead of a percent sign (%) in the variable's name:

```

setlocal enabledelayedexpansion
SET VAR=Hello
FOR /L %%a in (1,1,2) do (
    echo !VAR!
    SET VAR=Goodbye
)
endlocal

```

will print

```

Hello
Goodbye

```

The syntax `%%a in (1,1,2)` causes the loop to run 2 times: on the first occasion, the variable bears its initial value of 'Hello', but on the second pass through the loop - having executed the second `SET` instruction as the last action on the 1st pass - this has changed to the revised value 'Goodbye'.

Advanced variable substitution

Now, an advanced technique. Using the `CALL` command allows the batch command processor to expand a variable located on the same line of the script. This can deliver multilevel expansion, by repeated `CALL` and modifier use.

This is useful in, for example, a `FOR` loop. As in the following example, where we have a numbered list of variables:

```
"c:\MyFiles\test1.txt" "c:\MyFiles\test2.txt" "c:\MyFiles\test3.txt"
```

We can achieve this using the following `FOR` loop:

```

setlocal enabledelayedexpansion
for %%x in (*.*) do (
    set /a "i+=1"
    call set path!i!=%%~!i!
    call echo %%path!i!%%
)
endlocal

```

Output:

```
c:\MyFiles\test1.txt  
c:\MyFiles\test2.txt  
c:\MyFiles\test3.txt
```

Note that the variable `!i!` is first expanded to its initial value, 1, then the resulting variable, `%1`, is expanded to its actual value of `c:\MyFiles\test1.txt`. This is *double expansion* of the variable `i`. On the next line, `i` is again double expanded, by use of the `CALL ECHO` command together with the `%%` variable prefix, then printed to the screen (i.e. displayed on screen).

On each successive pass through the loop, the initial number is increased by 1 (due to the code `i+=1`). Thus it increases to `2` on the 2nd pass through the loop, and to `3` on the 3rd pass. Thus the string echoed to the screen alters with each pass.

Declare multiple variables

When multiple variables are defined at the beginning of the batch, a short definition form may be used by employing a [replacement](#) string.

```
@echo off  
set "vars=_A=good,_B=,_E=bad,_F=,_G=ugly,_C=,_H=,_I=,_J=,_K=,_L=,_D=6  
set "%vars:=" & set "%"  
  
for /f %%l in ('set _') do echo %%l  
exit /b  
  
_A=good  
_D=6  
_E=bad  
_G=ugly
```

Note in the above example, variables are natively alphabetically sorted, when printed to screen.

Using a Variable as an Array

It is possible to create a set of variables that can act similar to an array (although they are not an actual array object) by using spaces in the `SET` statement:

```
@echo off  
SET var=A "foo bar" 123  
for %%a in (%var%) do (  
    echo %%a  
)  
echo Get the variable directly: %var%
```

Result:

```
A  
"foo bar"  
123  
Get the variable directly: A "foo bar" 123
```

It is also possible to declare your variable using indexes so you may retrieve specific information. This will create multiple variables, with the illusion of an array:

```
@echo off
setlocal enabledelayedexpansion
SET var[0]=A
SET var[1]=foo bar
SET var[2]=123
for %%a in (0,1,2) do (
    echo !var[%a]!
)
echo Get one of the variables directly: %var[1]%
```

Result:

```
A
foo bar
123
Get one of the variables directly: foo bar
```

Note that in the example above, you cannot reference `var` without stating what the desired index is, because `var` does not exist in its own. This example also uses `setlocal enabledelayedexpansion` in conjunction with the exclamation points at `!var[%a]!`. You can view more information about this in the [Variable Substitution Scope Documentation](#).

Operations on Variables

```
set var=10
set /a var=%var%+10
echo %var%
```

The final value of `var` is 20.

The second line is not working within a command block used for example on an **IF** condition or on a **FOR** loop as delayed expansion would be needed instead of standard environment variable expansion.

Here is another, better way working also in a command block:

```
set var=10
set /A var+=10
echo %var%
```

The command prompt environment supports with signed 32-bit integer values:

- addition `+` and `+=`
- subtraction `-` and `-=`
- multiplication `*` and `*=`
- division `/` and `/=`
- modulus division `%` and `%=`
- bitwise AND `&`

- bitwise OR |
- bitwise NOT ~
- bitwise XOR ^
- bitwise left shift <<
- bitwise right shift >>
- logical NOT !
- unary minus -
- grouping with (and)

The Windows command interpreter does not support 64-bit integer values or floating point values in arithmetic expressions.

Note: The operator % must be written in a batch file as %% to be interpreted as operator.

In a command prompt window executing the command line `set /A Value=8 % 3` assigns the value 2 to environment variable `Value` and additionally outputs 2.

In a batch file must be written `set /A Value=8 %% 3` to assign the value 2 to environment variable `Value` and nothing is output respectively written to handle **STDOUT** (standard output). A line `set /A Value=8 % 3` in a batch file would result in error message *Missing operator* on execution of the batch file.

The environment requires the switch `/A` for arithmetic operations only, not for ordinary string variables.

Every string in the arithmetic expression after `set /A` being whether a number nor an operator is automatically interpreted as name of an environment variable.

For that reason referencing the value of a variable with `%variable%` or with `!variable!` is not necessary when the variable name consists only of word characters (0-9A-Za-z_) with first character not being a digit which is especially helpful within a command block starting with (and ending with a matching).

Numbers are converted from string to integer with C/C++ function `strtol` with `base` being zero which means automatic base determination which can easily result in unexpected results.

Example:

```
set Divided=11
set Divisor=3

set /A Quotient=Divided / Divisor
set /A Remainder=Divided %% Divisor

echo %Divided% / %Divisor% = %Quotient%
echo %Divided% %% %Divisor% = %Remainder%

set HexValue1=0x14
set HexValue2=0x0A
set /A Result=(HexValue1 + HexValue2) * -3

echo (%HexValue1% + %HexValue2%) * -3 = (20 + 10) * -3 = %Result%
```

```
set /A Result%%=7
echo -90 %%= 7 = %Result%

set OctalValue=020
set DecimalValue=12
set /A Result=OctalValue - DecimalValue

echo %OctalValue% - %DecimalValue% = 16 - 12 = %Result%
```

The output of this example is:

```
11 / 3 = 3
11 % 3 = 2
(0x14 + 0x0A) * -3 = (20 + 10) * -3 = -90
-90 %%= 7 = -6
020 - 12 = 16 - 12 = 4
```

Variables not defined on evaluation of the arithmetic expression are substituted with value 0.

Setting variables from an input

Using the `/p` switch with the `SET` command you can define variables from an Input.

This input can be a user Input (keyboard) :

```
echo Enter your name :
set /p name=
echo Your name is %name%
```

Which can be simplified like this :

```
set /p name=Enter your name :
echo Your name is %name%
```

Or you can get the input from a file :

```
set /p name=< file.txt
```

in this case you'll get the value of the first line from `file.txt`

Getting the value of various line in a file :

```
( 
    set /p line1=
    set /p line2=
    set /p line3=

) < file.txt
```

Read Variables in Batch Files online: <https://riptutorial.com/batch-file/topic/3528/variables-in-batch-files>

Credits

S. No	Chapters	Contributors
1	Getting started with batch-file	BoeNoe , Clijsters , Community , DavidPostill , Derpcode , geisterfurz007 , Jeffrey Lin , loadingnow , Mee , rudicangiotti , sambul35 , Scott Beeson , Sourav Ghosh , stark , SteveFest , ths
2	Add delay to Batch file	SteveFest , Tearzz , wizzwizz4
3	Batch and JSCript hybrids	npocmaka , SteveFest
4	Batch and VBS hybrids	npocmaka , SteveFest
5	Batch file command line arguments	DavidPostill , LotPings , npocmaka , sambul35
6	Batch file macros	SteveFest
7	Batch files and Powershell hybrids	npocmaka , SteveFest
8	Best Practices	SteveFest
9	Bugs in cmd.exe processor	SteveFest , X. Liu
10	Bypass arithmetic limitations in batch files	npocmaka
11	Changing Directories and Listing their Contents	Aravind .KEN , SomethingDark , SteveFest , wizzwizz4
12	Comments in Batch Files	0x90h , BoeNoe , DavidPostill , Gábor Bakos , JosefZ , LotPings , SachaDee , SomethingDark , Sourav Ghosh , Stephan , SteveFest , wasatchwizard
13	Creating Files using Batch	Aravind .KEN , David Starkey , fruitSalad266 , J03L , LeoDog896 , loadingnow , Tearzz
14	Deprecated batch commands and their	npocmaka , SteveFest

	replacements	
15	Differences between Batch (Windows) and Terminal (Linux)	SomethingDark , SteveFest
16	Directory Stack	DavidPostill , wizzwizz4
17	Echo	DavidPostill , fruitSalad266 , Keith Hall , npocmaka , SomethingDark , Tearzz , wizzwizz4
18	Elevated Privileges in Batch Files	DavidPostill , elzooilogico , sambul35
19	Escaping special characters	npocmaka , SteveFest
20	File Handling in batch files	BoeNoe , LeoDog896 , SteveFest
21	For Loops in Batch Files	David Starkey , DavidPostill , Ed999 , Mee , sambul35 , SomethingDark , SteveFest
22	Functions	ender_scythe , npocmaka , SteveFest
23	If statements	npocmaka , Ozair Kafray , SomethingDark , SteveFest
24	Input and output redirection	cascading-style , SomethingDark , SteveFest , wizzwizz4
25	Random In Batch Files	SteveFest
26	Search strings in batch	sambul35
27	Using Goto	LeoDog896 , SteveFest
28	Variables in Batch Files	DavidPostill , Ed999 , Jay , Jeffrey Lin , Mee , Mofi , SachaDee , sambul35 , SomethingDark , SteveFest , Tearzz , ths , Toomaja , wasatchwizard , wizzwizz4

Batch Script - Quick Guide

Batch Script - Overview

Batch Script is incorporated to automate command sequences which are repetitive in nature. Scripting is a way by which one can alleviate this necessity by automating these command sequences in order to make one's life at the shell easier and more productive. In most organizations, Batch Script is incorporated in some way or the other to automate stuff.

Some of the features of Batch Script are –

- Can read inputs from users so that it can be processed further.
- Has control structures such as for, if, while, switch for better automating and scripting.
- Supports advanced features such as Functions and Arrays.
- Supports regular expressions.
- Can include other programming codes such as Perl.

Some of the common uses of Batch Script are –

- Setting up servers for different purposes.
- Automating housekeeping activities such as deleting unwanted files or log files.
- Automating the deployment of applications from one environment to another.
- Installing programs on various machines at once.

Batch scripts are stored in simple text files containing lines with commands that get executed in sequence, one after the other. These files have the special extension BAT or CMD. Files of this type are recognized and executed through an interface (sometimes called a shell) provided by a system file called the command interpreter. On Windows systems, this interpreter is known as cmd.exe.

Running a batch file is a simple matter of just clicking on it. Batch files can also be run in a command prompt or the Start-Run line. In such case, the full path name must be used unless the file's path is in the path environment. Following is a simple example of a Batch Script. This Batch Script when run deletes all files in the current directory.

```
:: Deletes All files in the Current Directory With Prompts and Warnings
::(Hidden, System, and Read-Only Files are Not Affected)
:: @ECHO OFF
DEL . DR
```

Batch Script - Environment

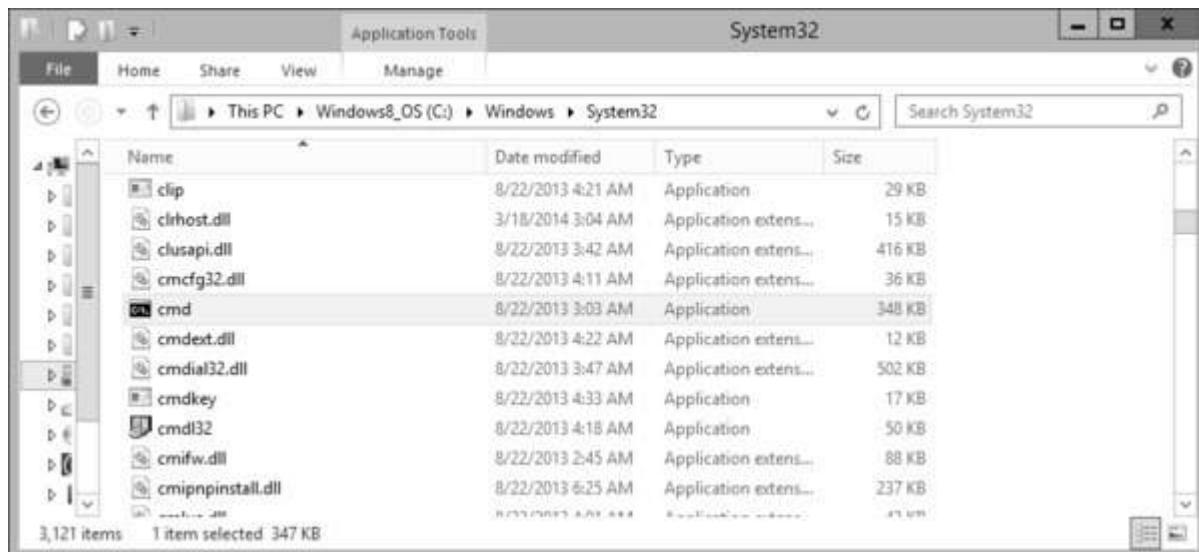
This chapter explains the environment related to Batch Script.

Writing and Executing

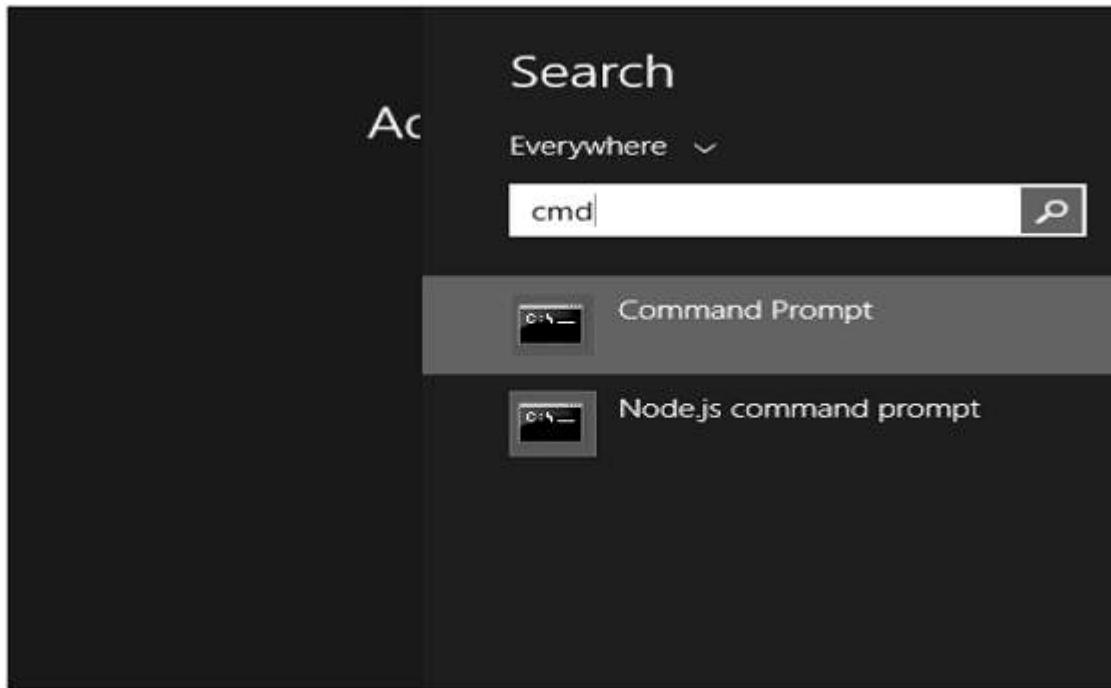
Typically, to create a batch file, notepad is used. This is the simplest tool for creation of batch files. Next is the execution environment for the batch scripts. On Windows systems, this is done via the command prompt or cmd.exe. All batch files are run in this environment.

Following are the different ways to launch cmd.exe –

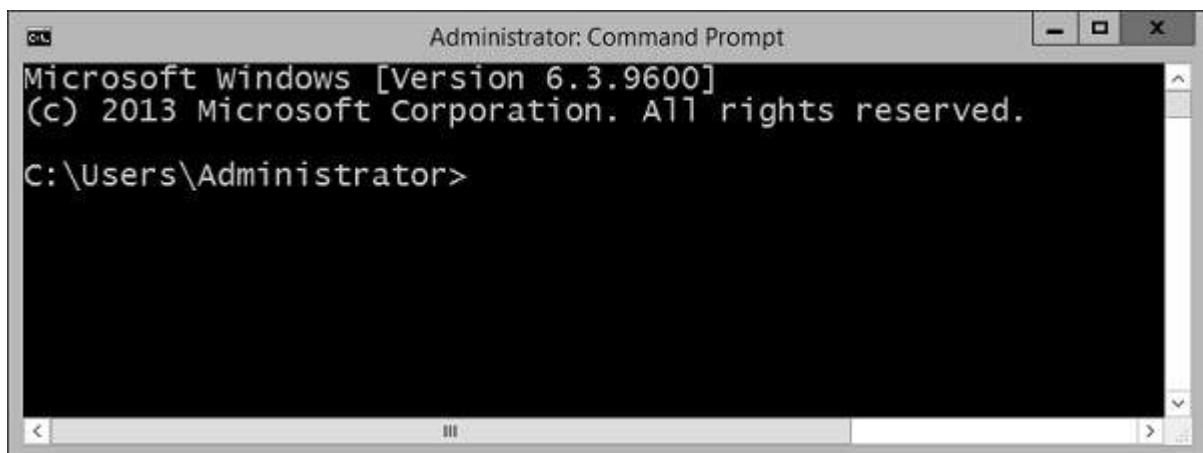
Method 1 – Go to C:\Windows\System32 and double click on the cmd file.



Method 2 – Via the run command – The following snapshot shows to find the command prompt(cmd.exe) on Windows server 2012.



Once the cmd.exe is launched, you will be presented with the following screen. This will be your environment for executing your batch scripts.



Environment Variables

In order to run batch files from the command prompt, you either need to go to the location to where the batch file is stored or alternatively you can enter the file location in the path environment variable. Thus assuming that the batch file is stored in the location C:\Application\bin, you would need to follow these instructions for the PATH variable inclusion.

OS	Output
Windows	Append the String; C:\Application\bin to the end of the system variable PATH.

Batch Script - Commands

In this chapter, we will look at some of the frequently used batch commands.

S.No	Commands & Description
1	VER This batch command shows the version of MS-DOS you are using.
2	ASSOC This is a batch command that associates an extension with a file type (FTYPE), displays existing associations, or deletes an association.
3	CD This batch command helps in making changes to a different directory, or displays the current directory.
4	CLS This batch command clears the screen.
5	COPY This batch command is used for copying files from one location to the other.
6	DEL This batch command deletes files and not directories.
7	DIR This batch command lists the contents of a directory.
8	DATE This batch command help to find the system date.
9	ECHO This batch command displays messages, or turns command echoing on or off.
10	EXIT This batch command exits the DOS console.
11	MD This batch command creates a new directory in the current location.
12	MOVE This batch command moves files or directories between directories.

13	PATH This batch command displays or sets the path variable.
14	PAUSE This batch command prompts the user and waits for a line of input to be entered.
15	PROMPT This batch command can be used to change or reset the cmd.exe prompt.
16	RD This batch command removes directories, but the directories need to be empty before they can be removed.
17	REN Renames files and directories
18	REM This batch command is used for remarks in batch files, preventing the content of the remark from being executed.
19	START This batch command starts a program in new window, or opens a document.
20	TIME This batch command sets or displays the time.
21	TYPE This batch command prints the content of a file or files to the output.
22	VOL This batch command displays the volume labels.
23	ATTRIB Displays or sets the attributes of the files in the current directory
24	CHKDSK This batch command checks the disk for any problems.
25	CHOICE This batch command provides a list of options to the user.
26	CMD This batch command invokes another instance of command prompt.

	COMP
27	This batch command compares 2 files based on the file size.
	CONVERT
28	This batch command converts a volume from FAT16 or FAT32 file system to NTFS file system.
	DRIVERQUERY
29	This batch command shows all installed device drivers and their properties.
	EXPAND
30	This batch command extracts files from compressed .cab cabinet files.
	FIND
31	This batch command searches for a string in files or input, outputting matching lines.
	FORMAT
32	This batch command formats a disk to use Windows-supported file system such as FAT, FAT32 or NTFS, thereby overwriting the previous content of the disk.
	HELP
33	This batch command shows the list of Windows-supplied commands.
	IPCONFIG
34	This batch command displays Windows IP Configuration. Shows configuration by connection and the name of that connection.
	LABEL
35	This batch command adds, sets or removes a disk label.
	MORE
36	This batch command displays the contents of a file or files, one screen at a time.
	NET
37	Provides various network services, depending on the command used.
	PING
38	This batch command sends ICMP/IP "echo" packets over the network to the designated address.
	SHUTDOWN
39	This batch command shuts down a computer, or logs off the current user.

	SORT
40	This batch command takes the input from a source file and sorts its contents alphabetically, from A to Z or Z to A. It prints the output on the console.
	SUBST
41	This batch command assigns a drive letter to a local folder, displays current assignments, or removes an assignment.
	SYSTEMINFO
42	This batch command shows configuration of a computer and its operating system.
	TASKKILL
43	This batch command ends one or more tasks.
	TASKLIST
44	This batch command lists tasks, including task name and process id (PID).
	XCOPY
45	This batch command copies files and directories in a more advanced way.
	TREE
46	This batch command displays a tree of all subdirectories of the current directory to any level of recursion or depth.
	FC
47	This batch command lists the actual differences between two files.
	DISKPART
48	This batch command shows and configures the properties of disk partitions.
	TITLE
49	This batch command sets the title displayed in the console window.
	SET
50	Displays the list of environment variables on the current system.

Batch Script - Files

In this chapter, we will learn how to create, save, execute, and modify batch files.

Creating Batch Files

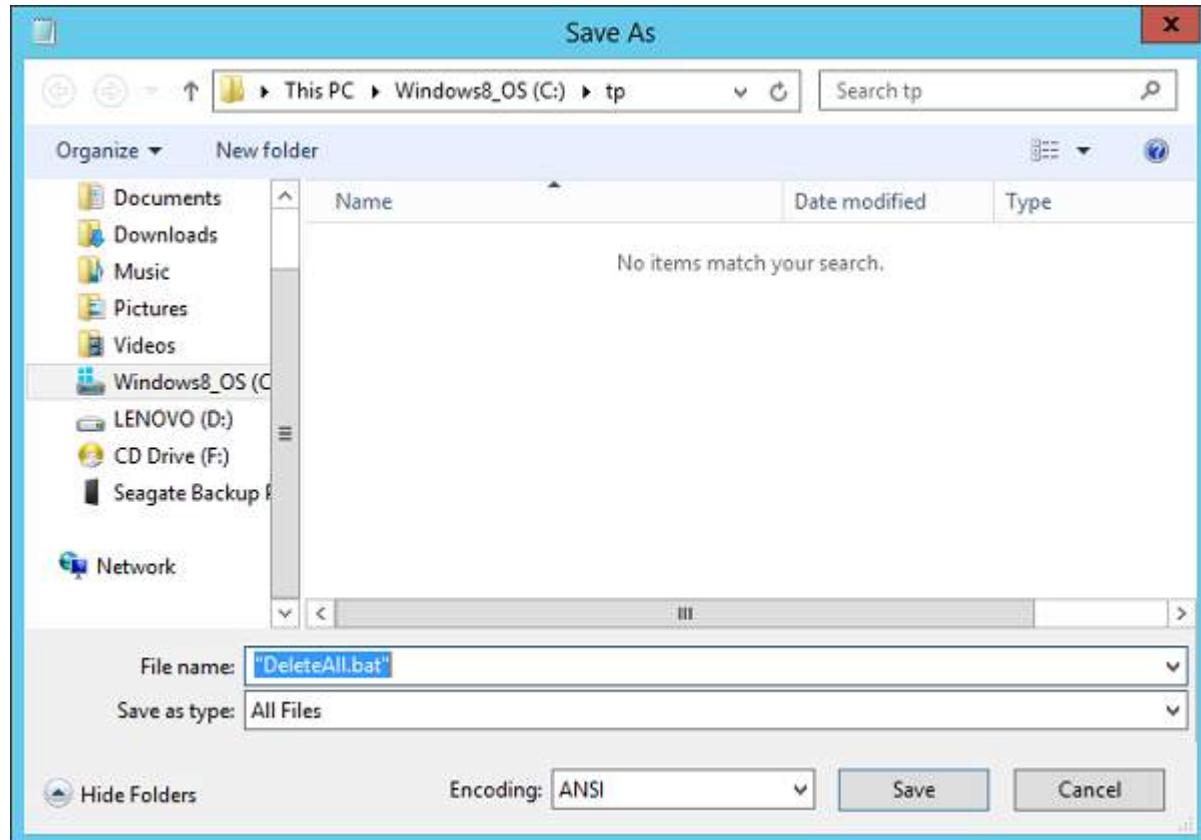
Batch files are normally created in notepad. Hence the simplest way is to open notepad and enter the commands required for the script. For this exercise, open notepad and enter the following statements.

```
:: Deletes All files in the Current Directory With Prompts and Warnings  
::(Hidden, System, and Read-Only Files are Not Affected)  
::  
@ECHO OFF  
DEL .  
DR
```

Saving Batch Files

After your batch file is created, the next step is to save your batch file. Batch files have the extension of either .bat or .cmd. Some general rules to keep in mind when naming batch files –

- Try to avoid spaces when naming batch files, it sometime creates issues when they are called from other scripts.
- Don't name them after common batch files which are available in the system such as ping.cmd.



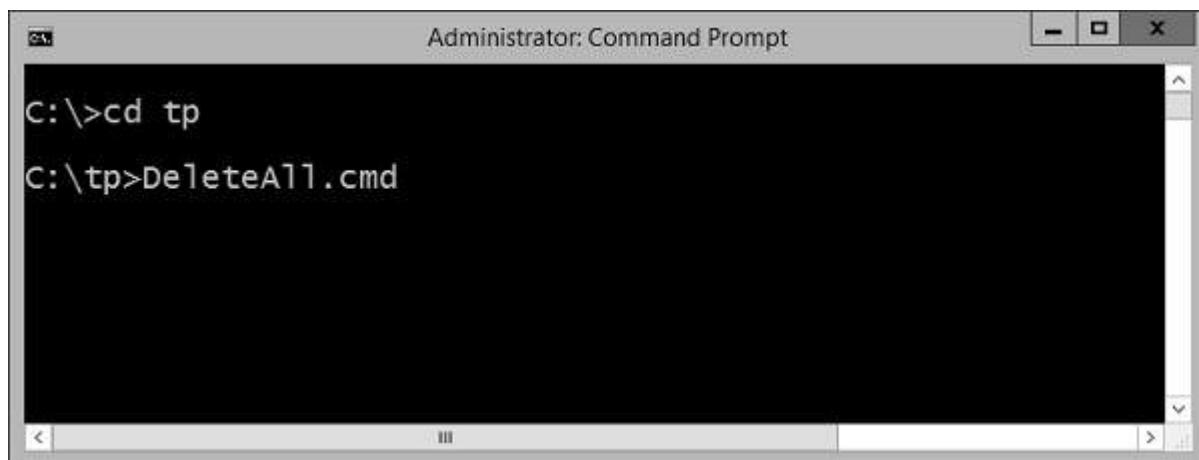
The above screenshot shows how to save the batch file. When saving your batch file a few points to keep in mind.

- Remember to put the .bat or .cmd at the end of the file name.
- Choose the “Save as type” option as “All Files”.
- Put the entire file name in quotes “”.

Executing Batch Files

Following are the steps to execute a batch file –

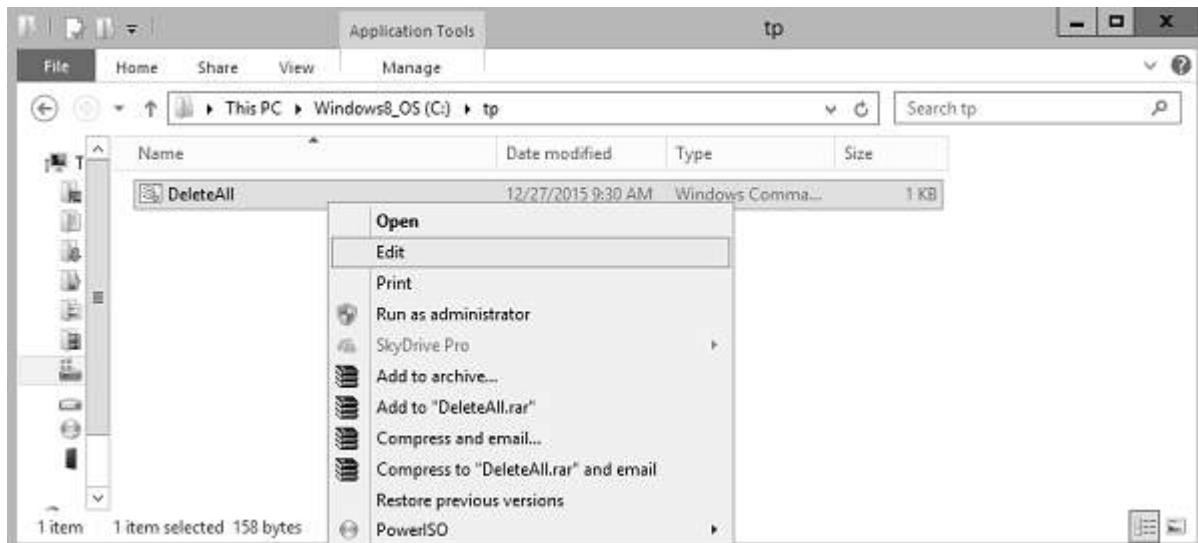
- **Step 1** – Open the command prompt (cmd.exe).
- **Step 2** – Go to the location where the .bat or .cmd file is stored.
- **Step 3** – Write the name of the file as shown in the following image and press the Enter button to execute the batch file.



Modifying Batch Files

Following are the steps for modifying an existing batch file.

- **Step 1** – Open windows explorer.
- **Step 2** – Go to the location where the .bat or .cmd file is stored.
- **Step 3** – Right-click the file and choose the “Edit” option from the context menu. The file will open in Notepad for further editing.



Batch Script - Syntax

Normally, the first line in a batch file often consists of the following command.

ECHO Command

```
@echo off
```

By default, a batch file will display its command as it runs. The purpose of this first command is to turn off this display. The command "echo off" turns off the display for the whole script, except for the "echo off" command itself. The "at" sign "@" in front makes the command apply to itself as well.

Documentation

Very often batch files also contain lines that start with the "Rem" command. This is a way to enter comments and documentation. The computer ignores anything on a line following Rem. For batch files with increasing amount of complexity, this is often a good idea to have comments.

First Batch Script Program

Let's construct our simple first batch script program. Open notepad and enter the following lines of code. Save the file as "List.cmd".

The code does the following –

- Uses the echo off command to ensure that the commands are not shown when the code is executed.
- The Rem command is used to add a comment to say what exactly this batch file does.
- The dir command is used to take the contents of the location C:\Program Files.
- The '>' command is used to redirect the output to the file C:\lists.txt.
- Finally, the echo command is used to tell the user that the operation is completed.

```
@echo off
Rem This is for listing down all the files in the directory Program files
dir "C:\Program Files" > C:\lists.txt
echo "The program has completed"
```

When the above command is executed, the names of the files in C:\Program Files will be sent to the file C:\Lists.txt and in the command prompt the message "The program has completed" will be displayed.

Batch Script - Variables

There are two types of variables in batch files. One is for parameters which can be passed when the batch file is called and the other is done via the set command.

Command Line Arguments

Batch scripts support the concept of command line arguments wherein arguments can be passed to the batch file when invoked. The arguments can be called from the batch files through the variables %1, %2, %3, and so on.

The following example shows a batch file which accepts 3 command line arguments and echo's them to the command line screen.

```
@echo off
echo %1
echo %2
echo %3
```

If the above batch script is stored in a file called test.bat and we were to run the batch as

Test.bat 1 2 3

Following is a screenshot of how this would look in the command prompt when the batch file is executed.



The above command produces the following output.

1
2
3

If we were to run the batch as

Example 1 2 3 4

The output would still remain the same as above. However, the fourth parameter would be ignored.

Set Command

The other way in which variables can be initialized is via the 'set' command. Following is the syntax of the set command.

Syntax

```
set /A variable-name=value
```

where,

- **variable-name** is the name of the variable you want to set.
- **value** is the value which needs to be set against the variable.
- **/A** – This switch is used if the value needs to be numeric in nature.

The following example shows a simple way the set command can be used.

Example

```
@echo off  
set message=Hello World  
echo %message%
```

- In the above code snippet, a variable called message is defined and set with the value of "Hello World".
- To display the value of the variable, note that the variable needs to be enclosed in the % sign.

Output

The above command produces the following output.

```
Hello World
```

Working with Numeric Values

In batch script, it is also possible to define a variable to hold a numeric value. This can be done by using the /A switch.

The following code shows a simple way in which numeric values can be set with the /A switch.

```
@echo off  
SET /A a = 5  
SET /A b = 10  
SET /A c = %a% + %b%  
echo %c%
```

- We are first setting the value of 2 variables, a and b to 5 and 10 respectively.
- We are adding those values and storing in the variable c.
- Finally, we are displaying the value of the variable c.

The output of the above program would be 15.

All of the arithmetic operators work in batch files. The following example shows arithmetic operators can be used in batch files.

```
@echo off  
SET /A a = 5  
SET /A b = 10  
SET /A c = %a% + %b%  
echo %c%  
SET /A c = %a% - %b%  
echo %c%  
SET /A c = %b% / %a%  
echo %c%  
SET /A c = %b% * %a%  
echo %c%
```

The above command produces the following output.

```
15  
-5  
2  
50
```

Local vs Global Variables

In any programming language, there is an option to mark variables as having some sort of scope, i.e. the section of code on which they can be accessed. Normally,

variable having a global scope can be accessed anywhere from a program whereas local scoped variables have a defined boundary in which they can be accessed.

DOS scripting also has a definition for locally and globally scoped variables. By default, variables are global to your entire command prompt session. Call the SETLOCAL command to make variables local to the scope of your script. After calling SETLOCAL, any variable assignments revert upon calling ENDLOCAL, calling EXIT, or when execution reaches the end of file (EOF) in your script. The following example shows the difference when local and global variables are set in the script.

Example

```
@echo off  
set globalvar = 5  
SETLOCAL  
set var = 13145  
set /A var = %var% + 5  
echo %var%  
echo %globalvar%  
ENDLOCAL
```

Few key things to note about the above program.

- The 'globalvar' is defined with a global scope and is available throughout the entire script.
- The 'var' variable is defined in a local scope because it is enclosed between a 'SETLOCAL' and 'ENDLOCAL' block. Hence, this variable will be destroyed as soon the 'ENDLOCAL' statement is executed.

Output

The above command produces the following output.

```
13150  
5
```

You will notice that the command echo %var% will not yield anything because after the ENDLOCAL statement, the 'var' variable will no longer exist.

Working with Environment Variables

If you have variables that would be used across batch files, then it is always preferable to use environment variables. Once the environment variable is defined, it can be accessed via the % sign. The following example shows how to see the JAVA_HOME defined on a system. The JAVA_HOME variable is a key component that is normally used by a wide variety of applications.

```
@echo off  
echo %JAVA_HOME%
```

The output would show the JAVA_HOME directory which would depend from system to system. Following is an example of an output.

```
C:\Atlassian\Bitbucket\4.0.1\jre
```

Batch Script - Comments

It's always a good practice to add comments or documentation for the scripts which are created. This is required for maintenance of the scripts to understand what the script actually does.

For example, consider the following piece of code which has no form of comments. If any average person who has not developed the following script tries to understand the script, it would take a lot of time for that person to understand what the script actually does.

```
ECHO OFF  
IF NOT "%OS%"=="Windows_NT" GOTO Syntax  
ECHO.%* | FIND "?" >NUL  
IF NOT ERRORLEVEL 1 GOTO Syntax  
IF NOT [%2]==[] GOTO Syntax  
SETLOCAL  
SET WSS=  
IF NOT [%1]==[] FOR /F "tokens = 1 delims = \ " %%A IN ('ECHO.%~1') DO SET WS=%%A  
FOR /F "tokens = 1 delims = \ " %%a IN ('NET VIEW ^| FIND /I "\%WSS%\") DO F  
"tokens = 1 delims = " %%A IN ('NBTSTAT -a %%a ^| FIND /I /V "%a" ^| FIND "<  
DO ECHO.%a %%A  
ENDLOCAL  
GOTO:EOF
```

```
ECHO Display logged on users and their workstations.  
ECHO Usage: ACTUSR [ filter ]  
IF "%OS%"=="Windows_NT" ECHO Where: filter is the first part  
of the computer name^(s^) to be displayed
```

Comments Using the Rem Statement

There are two ways to create comments in Batch Script; one is via the Rem command. Any text which follows the Rem statement will be treated as comments and will not be executed. Following is the general syntax of this statement.

Syntax

Rem Remarks

where 'Remarks' is the comments which needs to be added.

The following example shows a simple way the **Rem** command can be used.

Example

```
@echo off  
Rem This program just displays Hello World  
set message=Hello World  
echo %message%
```

Output

The above command produces the following output. You will notice that the line with the Rem statement will not be executed.

Hello World

Comments Using the :: Statement

The other way to create comments in Batch Script is via the :: command. Any text which follows the :: statement will be treated as comments and will not be executed. Following is the general syntax of this statement.

Syntax

```
:: Remarks
```

where 'Remarks' is the comment which needs to be added.

The following example shows a simple way the Rem command can be used.

Example

```
@echo off
:: This program just displays Hello World
set message = Hello World
echo %message%
```

Output

The above command produces the following output. You will notice that the line with the :: statement will not be executed.

```
Hello World
```

Note – If you have too many lines of Rem, it could slow down the code, because in the end each line of code in the batch file still needs to be executed.

Let's look at the example of the large script we saw at the beginning of this topic and see how it looks when documentation is added to it.

```
=====
:: The below example is used to find computer and logged on users
::
=====
ECHO OFF
:: Windows version check
IF NOT "%OS%"=="Windows_NT" GOTO Syntax
ECHO.%* | FIND "?" >NUL
:: Command line parameter check
IF NOT ERRORLEVEL 1 GOTO Syntax
IF NOT [%2]==[] GOTO Syntax
:: Keep variable local
```

```

SETLOCAL
:: Initialize variable
SET WSS=
:: Parse command line parameter
IF NOT [%1]==[] FOR /F "tokens = 1 delims = \ " %%A IN ('ECHO.%~1') DO SET WS=%A
:: Use NET VIEW and NBTSTAT to find computers and logged on users
FOR /F "tokens = 1 delims = \ " %%a IN ('NET VIEW ^| FIND /I "\%WSS%") DO FOR /F "tokens = 1 delims = " %%A IN ('NBTSTAT -a %%a ^| FIND /I /V "%a" ^| FIND "<03>"') DO ECHO.%a %%A
:: Done
ENDLOCAL
GOTO:EOF
:Syntax
ECHO Display logged on users and their workstations.
ECHO Usage: ACTUSR [ filter ]
IF "%OS%"=="Windows_NT" ECHO Where: filter is the first part of the
computer name^(s^) to be displayed

```

You can now see that the code has become more understandable to users who have not developed the code and hence is more maintainable.

Batch Script - Strings

In DOS, a string is an ordered collection of characters, such as "Hello, World!".

S.No	Strings & Description
1	Create String A string can be created in DOS in the following way.
2	Empty String Empty String
3	String Interpolation String interpolation is a way to construct a new String value from a mix of constants, variables, literals, and expressions by including their values inside a string literal.
4	String Concatenation You can use the set operator to concatenate two strings or a string and a character, or two characters. Following is a simple example which shows how to use string concatenation.

	String length
5	In DOS scripting, there is no length function defined for finding the length of a string. There are custom-defined functions which can be used for the same. Following is an example of a custom-defined function for seeing the length of a string.
6	toInt A variable which has been set as string using the set variable can be converted to an integer using the /A switch which is using the set variable. The following example shows how this can be accomplished.
7	Align Right This used to align text to the right, which is normally used to improve readability of number columns.
8	Left String This is used to extract characters from the beginning of a string.
9	Mid String This is used to extract a substring via the position of the characters in the string.
10	Remove The string substitution feature can also be used to remove a substring from another string.
11	Remove Both Ends This is used to remove the first and the last character of a string.
12	Remove All Spaces This is used to remove all spaces in a string via substitution.
13	Replace a String To replace a substring with another string use the string substitution feature.
14	Right String This is used to extract characters from the end of a string.

Batch Script - Arrays

Arrays are not specifically defined as a type in Batch Script but can be implemented. The following things need to be noted when arrays are implemented in Batch Script.

- Each element of the array needs to be defined with the set command.
- The 'for' loop would be required to iterate through the values of the array.

Creating an Array

An array is created by using the following set command.

```
set a[0]=1
```

Where 0 is the index of the array and 1 is the value assigned to the first element of the array.

Another way to implement arrays is to define a list of values and iterate through the list of values. The following example show how this can be implemented.

Example

```
@echo off
set list = 1 2 3 4
(for %%a in (%list%) do (
    echo %%a
))
```

Output

The above command produces the following output.

```
1
2
3
4
```

Accessing Arrays

You can retrieve a value from the array by using subscript syntax, passing the index of the value you want to retrieve within square brackets immediately after the name of the array.

Example

```
@echo off  
set a[0]=1  
echo %a[0]%
```

In this example, the index starts from 0 which means the first element can be accessed using index as 0, the second element can be accessed using index as 1 and so on. Let's check the following example to create, initialize and access arrays –

```
@echo off  
set a[0] = 1  
set a[1] = 2  
set a[2] = 3  
echo The first element of the array is %a[0]%
echo The second element of the array is %a[1]%
echo The third element of the array is %a[2]%
```

The above command produces the following output.

```
The first element of the array is 1  
The second element of the array is 2  
The third element of the array is 3
```

Modifying an Array

To add an element to the end of the array, you can use the set element along with the last index of the array element.

Example

```
@echo off  
set a[0] = 1  
set a[1] = 2  
set a[2] = 3  
Rem Adding an element at the end of an array  
Set a[3] = 4  
echo The last element of the array is %a[3]%
```

The above command produces the following output.

The last element of the array is 4

You can modify an existing element of an Array by assigning a new value at a given index as shown in the following example –

```
@echo off
set a[0] = 1
set a[1] = 2
set a[2] = 3
Rem Setting the new value for the second element of the array
Set a[1] = 5
echo The new value of the second element of the array is %a[1]%
```

The above command produces the following output.

The new value of the second element of the array is 5

Iterating Over an Array

Iterating over an array is achieved by using the ‘for’ loop and going through each element of the array. The following example shows a simple way that an array can be implemented.

```
@echo off
setlocal enabledelayedexpansion
set topic[0] = comments
set topic[1] = variables
set topic[2] = Arrays
set topic[3] = Decision making
set topic[4] = Time and date
set topic[5] = Operators

for /l %%n in (0,1,5) do (
    echo !topic[%n]!
)
```

Following things need to be noted about the above program –

- Each element of the array needs to be specifically defined using the set command.
- The 'for' loop with the /L parameter for moving through ranges is used to iterate through the array.

Output

The above command produces the following output.

Comments

variables

Arrays

Decision making

Time and date

Operators

Length of an Array

The length of an array is done by iterating over the list of values in the array since there is no direct function to determine the number of elements in an array.

```
@echo off
set Arr[0] = 1
set Arr[1] = 2
set Arr[2] = 3
set Arr[3] = 4
set "x = 0"
:SymLoop

if defined Arr[%x%] (
    call echo %%Arr[%x%]%%
    set /a "x+=1"
    GOTO :SymLoop
)
echo "The length of the array is" %x%
```

Output

The above command produces the following output.

The length of the array is 4

Creating Structures in Arrays

Structures can also be implemented in batch files using a little bit of an extra coding for implementation. The following example shows how this can be achieved.

Example

```
@echo off
set len = 3
set obj[0].Name = Joe
set obj[0].ID = 1
set obj[1].Name = Mark
set obj[1].ID = 2
set obj[2].Name = Mohan
set obj[2].ID = 3
set i = 0
:loop

if %i% equ %len% goto :eof
set cur.Name=
set cur.ID=

for /f "usebackq delims==.tokens=1-3" %%j in (`set obj[%i%]`) do (
    set cur.%%k=%&1
)
echo Name = %cur.Name%
echo Value = %cur.ID%
set /a i = %i%+1
goto loop
```

The following key things need to be noted about the above code.

- Each variable defined using the set command has 2 values associated with each index of the array.
- The variable **i** is set to 0 so that we can loop through the structure will the length of the array which is 3.

- We always check for the condition on whether the value of **i** is equal to the value of **len** and if not, we loop through the code.
- We are able to access each element of the structure using the **obj[%i%]** notation.

Output

The above command produces the following output.

```
Name = Joe
Value = 1
Name = Mark
Value = 2
Name = Mohan
Value = 3
```

Batch Script - Decision Making

Decision-making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be **true**, and optionally, other statements to be executed if the condition is determined to be **false**.

S.No	Strings & Description
1	If Statement The first decision-making statement is the 'if' statement.
2	If/else Statement The next decision making statement is the If/else statement. Following is the general form of this statement.
3	Nested If Statements Sometimes, there is a requirement to have multiple 'if' statement embedded inside each other. Following is the general form of this statement.

Batch Script - Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

In batch script, the following types of operators are possible.

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- Bitwise operators

Arithmetic Operators

Batch script language supports the normal Arithmetic operators as any language. Following are the Arithmetic operators available.

[Show Example](#)

Operator	Description	Example
+	Addition of two operands	1 + 2 will give 3
-	Subtracts second operand from the first	2 - 1 will give 1
*	Multiplication of both operands	2 * 2 will give 4
/	Division of the numerator by the denominator	3 / 2 will give 1.5
%	Modulus operator and remainder of after an integer/float division	3 % 2 will give 1

Relational Operators

Relational operators allow of the comparison of objects. Below are the relational operators available.

[Show Example](#)

Operator	Description	Example
EQU	Tests the equality between two objects	2 EQU 2 will give true
NEQ	Tests the difference between two objects	3 NEQ 2 will give true

LSS	Checks to see if the left object is less than the right operand	2 LSS 3 will give true
LEQ	Checks to see if the left object is less than or equal to the right operand	2 LEQ 3 will give true
GTR	Checks to see if the left object is greater than the right operand	3 GTR 2 will give true
GEQ	Checks to see if the left object is greater than or equal to the right operand	3 GEQ 2 will give true

Logical Operators

Logical operators are used to evaluate Boolean expressions. Following are the logical operators available.

The batch language is equipped with a full set of Boolean logic operators like AND, OR, XOR, but only for binary numbers. Neither are there any values for TRUE or FALSE. The only logical operator available for conditions is the NOT operator.

Show Example

Operator	Description
AND	This is the logical “and” operator
OR	This is the logical “or” operator
NOT	This is the logical “not” operator

Assignment Operators

Batch Script language also provides assignment operators. Following are the assignment operators available.

Show Example

Operator	Description	Example
<code>+=</code>	This adds right operand to the left operand and assigns the result to left operand	Set /A a = 5 a += 3 Output will be 8

-=	This subtracts the right operand from the left operand and assigns the result to the left operand	Set /A a = 5 a -= 3 Output will be 2
*=	This multiplies the right operand with the left operand and assigns the result to the left operand	Set /A a = 5 a *= 3 Output will be 15
/=	This divides the left operand with the right operand and assigns the result to the left operand	Set /A a = 6 a/ = 3 Output will be 2
%=	This takes modulus using two operands and assigns the result to the left operand	Set /A a = 5 a% = 3 Output will be 2

Bitwise Operators

Bitwise operators are also possible in batch script. Following are the operators available.

Show Example

Operator	Description
&	This is the bitwise "and" operator
	This is the bitwise "or" operator
^	This is the bitwise "xor" or Exclusive or operator

Following is the truth table showcasing these operators.

p	q	p & q	p q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Batch Script - DATE and TIME

The date and time in DOS Scripting have the following two basic commands for retrieving the date and time of the system.

DATE

This command gets the system date.

Syntax

```
DATE
```

Example

```
@echo off  
echo %DATE%
```

Output

The current date will be displayed in the command prompt. For example,

```
Mon 12/28/2015
```

TIME

This command sets or displays the time.

Syntax

```
TIME
```

Example

```
@echo off  
echo %TIME%
```

Output

The current system time will be displayed. For example,

```
22:06:52.87
```

Following are some implementations which can be used to get the date and time in different formats.

Date in Format Year-Month-Day

Example

```
@echo off
echo/Today is: %year%-%month%-%day%
goto :EOF
setlocal ENABLEEXTENSIONS
set t = 2&if "%date%z" LSS "A" set t = 1

for /f "skip=1 tokens = 2-4 delims = (-)" %%a in ('echo/^|date') do (
    for /f "tokens = %t%-4 delims=-./ " %%d in ('date/t') do (
        set %%a=%%d&set %%b=%%e&set %%c=%%f)
endlocal&set %1=%yy%&set %2=%mm%&set %3=%dd%&goto :EOF
```

Output

The above command produces the following output.

```
Today is: 2015-12-30
```

Batch Script - Input / Output

There are three universal “files” for keyboard input, printing text on the screen and printing errors on the screen. The “Standard In” file, known as **stdin**, contains the input to the program/script. The “Standard Out” file, known as **stdout**, is used to write output for display on the screen. Finally, the “Standard Err” file, known as **stderr**, contains any error messages for display on the screen.

Each of these three standard files, otherwise known as the standard streams, are referenced using the numbers 0, 1, and 2. Stdin is file 0, stdout is file 1, and stderr is file 2.

Redirecting Output (Stdout and Stderr)

One common practice in batch files is sending the output of a program to a log file. The `>` operator sends, or redirects, stdout or stderr to another file. The following example shows how this can be done.

```
Dir C:\ > list.txt
```

In the above example, the **stdout** of the command `Dir C:\` is redirected to the file `list.txt`.

If you append the number 2 to the redirection filter, then it would redirect the **stderr** to the file `lists.txt`.

```
Dir C:\ 2> list.txt
```

One can even combine the **stdout** and **stderr** streams using the file number and the '`&`' prefix. Following is an example.

```
DIR C:\ > lists.txt 2>&1
```

Suppressing Program Output

The pseudo file `NUL` is used to discard any output from a program. The following example shows that the output of the command `DIR` is discarded by sending the output to `NUL`.

```
Dir C:\ > NUL
```

Stdin

To work with the `Stdin`, you have to use a workaround to achieve this. This can be done by redirecting the command prompt's own `stdin`, called `CON`.

The following example shows how you can redirect the output to a file called `lists.txt`. After you execute the below command, the command prompt will take all the input entered by user till it gets an EOF character. Later, it sends all the input to the file `lists.txt`.

TYPE CON > lists.txt

Batch Script - Return Code

By default when a command line execution is completed it should either return zero when execution succeeds or non-zero when execution fails. When a batch script returns a non-zero value after the execution fails, the non-zero value will indicate what is the error number. We will then use the error number to determine what the error is about and resolve it accordingly.

Following are the common exit code and their description.

Error Code	Description
0	Program successfully completed.
1	Incorrect function. Indicates that Action has attempted to execute non-recognized command in Windows command prompt cmd.exe.
2	The system cannot find the file specified. Indicates that the file cannot be found in specified location.
3	The system cannot find the path specified. Indicates that the specified path cannot be found.
5	Access is denied. Indicates that user has no access right to specified resource.
9009 0x2331	Program is not recognized as an internal or external command, operable program or batch file. Indicates that command, application name or path has been misspelled when configuring the Action.
221225495 0xC0000017 -1073741801	Not enough virtual memory is available. It indicates that Windows has run out of memory.
3221225786 0xC000013A -1073741510	The application terminated as a result of a CTRL+C. Indicates that the application has been terminated either by the user's keyboard input CTRL+C or CTRL+Break or closing command prompt window.

3221225794
0xC0000142
-1073741502

The application failed to initialize properly. Indicates that the application has been launched on a Desktop to which the current user has no access rights. Another possible cause is that either gdi32.dll or user32.dll has failed to initialize.

Error Level

The environmental variable %ERRORLEVEL% contains the return code of the last executed program or script.

By default, the way to check for the ERRORLEVEL is via the following code.

Syntax

```
IF %ERRORLEVEL% NEQ 0 (
    DO_Something
)
```

It is common to use the command EXIT /B %ERRORLEVEL% at the end of the batch file to return the error codes from the batch file.

EXIT /B at the end of the batch file will stop execution of a batch file.

Use EXIT /B < exitcodes > at the end of the batch file to return custom return codes.

Environment variable %ERRORLEVEL% contains the latest errorlevel in the batch file, which is the latest error codes from the last command executed. In the batch file, it is always a good practice to use environment variables instead of constant values, since the same variable get expanded to different values on different computers.

Let's look at a quick example on how to check for error codes from a batch file.

Example

Let's assume we have a batch file called Find.cmd which has the following code. In the code, we have clearly mentioned that we if don't find the file called lists.txt then we should set the errorlevel to 7. Similarly, if we see that the variable userprofile is not defined then we should set the errorlevel code to 9.

```
if not exist c:\lists.txt exit 7
if not defined userprofile exit 9
```

```
exit 0
```

Let's assume we have another file called App.cmd that calls Find.cmd first. Now, if the Find.cmd returns an error wherein it sets the errorlevel to greater than 0 then it would exit the program. In the following batch file, after calling the Find.cmd find, it actually checks to see if the errorlevel is greater than 0.

Call Find.cmd

```
if errorlevel gtr 0 exit  
echo "Successful completion"
```

Output

In the above program, we can have the following scenarios as the output –

- If the file c:\lists.txt does not exist, then nothing will be displayed in the console output.
- If the variable userprofile does not exist, then nothing will be displayed in the console output.
- If both of the above condition passes then the string “Successful completion” will be displayed in the command prompt.

Loops

In the decision making chapter, we have seen statements which have been executed one after the other in a sequential manner. Additionally, implementations can also be done in Batch Script to alter the flow of control in a program's logic. They are then classified into flow of control statements.

S.No	Loops & Description
1	While Statement Implementation There is no direct while statement available in Batch Script but we can do an implementation of this loop very easily by using the if statement and labels.
2	For Statement - List Implementations The "FOR" construct offers looping capabilities for batch files. Following is the common construct of the 'for' statement for working with a list of

values.

Looping through Ranges

- 3 The 'for' statement also has the ability to move through a range of values. Following is the general form of the statement.

Classic for Loop Implementation

- 4 Following is the classic 'for' statement which is available in most programming languages.

Looping through Command Line Arguments

The 'for' statement can also be used for checking command line arguments. The following example shows how the 'for' statement can be used to loop through the command line arguments.

Example

```
@ECHO OFF
:Loop

IF "%1"=="" GOTO completed
FOR %%F IN (%1) DO echo %%F
SHIFT
GOTO Loop
:completed
```

Output

Let's assume that our above code is stored in a file called Test.bat. The above command will produce the following output if the batch file passes the command line arguments of 1,2 and 3 as Test.bat 1 2 3.

```
1
2
3
```

S.No	Loops & Description
1	Break Statement Implementation

The break statement is used to alter the flow of control inside loops within any programming language. The break statement is normally used in looping constructs and is used to cause immediate termination of the innermost enclosing loop.

Batch Script - Functions

A function is a set of statements organized together to perform a specific task. In batch scripts, a similar approach is adopted to group logical statements together to form a function.

As like any other languages, functions in Batch Script follows the same procedure –

- **Function Declaration** – It tells the compiler about a function's name, return type, and parameters.
- **Function Definition** – It provides the actual body of the function.

Function Definition

In Batch Script, a function is defined by using the label statement. When a function is newly defined, it may take one or several values as input 'parameters' to the function, process the functions in the main body, and pass back the values to the functions as output 'return types'.

Every function has a function name, which describes the task that the function performs. To use a function, you "call" that function with its name and pass its input values (known as arguments) that matches the types of the function's parameters.

Following is the syntax of a simple function.

```
:function_name  
Do_something  
EXIT /B 0
```

- The function_name is the name given to the function which should have some meaning to match what the function actually does.
- The EXIT statement is used to ensure that the function exits properly.

Following is an example of a simple function.

Example

```
:Display
SET /A index=2
echo The value of index is %index%
EXIT /B 0
```

S.No	Functions & Description
1	Calling a Function A function is called in Batch Script by using the call command.
2	Functions with Parameters Functions can work with parameters by simply passing them when a call is made to the function.
3	Functions with Return Values Functions can work with return values by simply passing variables names
4	Local Variables in Functions Local variables in functions can be used to avoid name conflicts and keep variable changes local to the function.
5	Recursive Functions The ability to completely encapsulate the body of a function by keeping variable changes local to the function and invisible to the caller.
6	File I/O In Batch Script, it is possible to perform the normal file I/O operations that would be expected in any programming language.
7	Creating Files The creation of a new file is done with the help of the redirection filter >. This filter can be used to redirect any output to a file.
8	Writing to Files Content writing to files is also done with the help of the redirection filter >. This filter can be used to redirect any output to a file.
9	Appending to Files Content writing to files is also done with the help of the double redirection filter >>. This filter can be used to append any output to a file.

	Reading from Files
10	Reading of files in a batch script is done via using the FOR loop command to go through each line which is defined in the file that needs to be read.
11	Deleting Files For deleting files, Batch Script provides the DEL command.
12	Renaming Files For renaming files, Batch Script provides the REN or RENAME command.
13	Moving Files For moving files, Batch Script provides the MOVE command.
14	Batch Files – Pipes The pipe operator () takes the output (by default, STDOUT) of one command and directs it into the input (by default, STDIN) of another command.
15	Batch Files – Inputs When a batch file is run, it gives you the option to pass in command line parameters which can then be read within the program for further processing.
16	Using the SHIFT Operator One of the limitations of command line arguments is that it can accept only arguments till %9. Let's take an example of this limitation.
17	Folders In Batch Script, it is possible to perform the normal folder based operations that would be expected in any programming language.
18	Creating Folders The creation of a folder is done with the assistance of the MD (Make directory) command.
19	Listing Folder Contents The listing of folder contents can be done with the dir command. This command allows you to see the available files and directories in the current directory.
20	Deleting Folders For deleting folders, Batch Scripting provides the DEL command.
21	Renaming Folders For renaming folders, Batch Script provides the REN or RENAME command.

22

Moving Folders

For moving folders, Batch Script provides the MOVE command.

Batch Script - Process

In this chapter, we will discuss the various processes involved in Batch Script.

Viewing the List of Running Processes

In Batch Script, the TASKLIST command can be used to get the list of currently running processes within a system.

Syntax

```
TASKLIST [/S system [/U username [/P [password]]]] [/M [module] | /SVC | /V] [/FI filter]
[/FO format] [/NH]
```

Following are the description of the options which can be presented to the TASKLIST command.

S.No.	Options & Description
1.	/S system Specifies the remote system to connect to
2.	/U [domain\]user Specifies the user context under which the command should execute.
3.	/P [password] Specifies the password for the given user context. Prompts for input if omitted.
4.	/M [module] Lists all tasks currently using the given exe/dll name. If the module name is not specified all loaded modules are displayed.
5.	/SVC Displays services hosted in each process.

6.	/V Displays verbose task information.
7.	/FI filter Displays a set of tasks that match a given criteria specified by the filter.
8.	/FO format Specifies the output format. Valid values: "TABLE", "LIST", "CSV".
9.	/NH Specifies that the "Column Header" should not show in the output. Valid only for "TABLE" and "CSV" formats.

Examples

TASKLIST

The above command will get the list of all the processes running on your local system. Following is a snapshot of the output which is rendered when the above command is run as it is. As you can see from the following output, not only do you get the various processes running on your system, you also get the memory usage of each process.

Image Name	PID	Session Name	Session#	Mem Usage
System Idle Process	0	Services	0	4 K
System	4	Services	0	272 K
smss.exe	344	Services	0	1,040 K
csrss.exe	528	Services	0	3,892 K
csrss.exe	612	Console	1	41,788 K
wininit.exe	620	Services	0	3,528 K
winlogon.exe	648	Console	1	5,884 K
services.exe	712	Services	0	6,224 K
lsass.exe	720	Services	0	9,712 K
svchost.exe	788	Services	0	10,048 K
svchost.exe	832	Services	0	7,696 K
dwm.exe	916	Console	1	117,440 K
nvvsvc.exe	932	Services	0	6,692 K
nvxdsync.exe	968	Console	1	16,328 K
nvvsvc.exe	976	Console	1	12,756 K
svchost.exe	1012	Services	0	21,648 K

svchost.exe	236	Services	0	33,864 K
svchost.exe	480	Services	0	11,152 K
svchost.exe	1028	Services	0	11,104 K
svchost.exe	1048	Services	0	16,108 K
wlanext.exe	1220	Services	0	12,560 K
conhost.exe	1228	Services	0	2,588 K
svchost.exe	1276	Services	0	13,888 K
svchost.exe	1420	Services	0	13,488 K
spoolsv.exe	1556	Services	0	9,340 K

```
tasklist > process.txt
```

The above command takes the output displayed by tasklist and saves it to the process.txt file.

```
tasklist /fi "memusage gt 40000"
```

The above command will only fetch those processes whose memory is greater than 40MB. Following is a sample output that can be rendered.

Image Name	PID	Session Name	Session#	Mem Usage
dwm.exe	916	Console	1	127,912 K
explorer.exe	2904	Console	1	125,868 K
ServerManager.exe	1836	Console	1	59,796 K
WINWORD.EXE	2456	Console	1	144,504 K
chrome.exe	4892	Console	1	123,232 K
chrome.exe	4976	Console	1	69,412 K
chrome.exe	1724	Console	1	76,416 K
chrome.exe	3992	Console	1	56,156 K
chrome.exe	1168	Console	1	233,628 K
chrome.exe	816	Console	1	66,808 K

Killing a Particular Process

Allows a user running Microsoft Windows XP professional, Windows 2003, or later to kill a task from a Windows command line by process id (PID) or image name. The command used for this purpose is the TASKKILL command.

Syntax

```
TASKKILL [/S system [/U username [/P [password]]]] { [/FI filter]
[/PID processid | /IM imagename] } [/T] [/F]
```

Following are the description of the options which can be presented to the TASKKILL command.

S.No.	Options & Description
1.	/S system Specifies the remote system to connect to
2.	/U [domain\]user Specifies the user context under which the command should execute.
3.	/P [password] Specifies the password for the given user context. Prompts for input if omitted.
4.	/FI FilterName Applies a filter to select a set of tasks. Allows "*" to be used. ex. imagename eq acme* See below filters for additional information and examples.
5.	/PID processID Specifies the PID of the process to be terminated. Use TaskList to get the PID.
6.	/IM ImageName Specifies the image name of the process to be terminated. Wildcard '*' can be used to specify all tasks or image names.
7.	/T Terminates the specified process and any child processes which were started by it.
8.	/F

Specifies to forcefully terminate the process(es).

Examples

```
taskkill /f /im notepad.exe
```

The above command kills the open notepad task, if open.

```
taskkill /pid 9214
```

The above command kills a process which has a process of 9214.

Starting a New Process

DOS scripting also has the availability to start a new process altogether. This is achieved by using the START command.

Syntax

```
START "title" [/D path] [options] "command" [parameters]
```

Wherein

- **title** – Text for the CMD window title bar (required.)
- **path** – Starting directory.
- **command** – The command, batch file or executable program to run.
- **parameters** – The parameters passed to the command.

Following are the description of the options which can be presented to the START command.

S.No.	Options & Description
1.	/MIN Start window Minimized
2.	/MAX

	Start window maximized.
3.	/LOW Use IDLE priority class.
4.	/NORMAL Use NORMAL priority class.
5.	/ABOVENORMAL Use ABOVENORMAL priority class.
6.	/BELOWNORMAL Use BELOWNORMAL priority class.
7.	/HIGH Use HIGH priority class.
8.	/REALTIME Use REALTIME priority class.

Examples

```
START "Test Batch Script" /Min test.bat
```

The above command will run the batch script test.bat in a new window. The windows will start in the minimized mode and also have the title of "Test Batch Script".

```
START "" "C:\Program Files\Microsoft Office\Winword.exe" "D:\test\TESTA.txt"
```

The above command will actually run Microsoft word in another process and then open the file TESTA.txt in MS Word.

Batch Script - Aliases

Aliases means creating shortcuts or keywords for existing commands. Suppose if we wanted to execute the below command which is nothing but the directory listing command with the /w option to not show all of the necessary details in a directory listing.

```
Dir /w
```

Suppose if we were to create a shortcut to this command as follows.

```
dw = dir /w
```

When we want to execute the **dir /w** command, we can simply type in the word **dw**. The word 'dw' has now become an alias to the command Dir /w.

Creating an Alias

Alias are managed by using the **doskey** command.

Syntax

```
DOSKEY [options] [macroname=[text]]
```

Wherein

- **macroname** – A short name for the macro.
- **text** – The commands you want to recall.

Following are the description of the options which can be presented to the DOSKEY command.

S.No.	Options & Description
1.	/REINSTALL Installs a new copy of Doskey
2.	/LISTSIZE = size Sets size of command history buffer.
3.	/MACROS Displays all Doskey macros.
4.	/MACROS:ALL Displays all Doskey macros for all executables which have Doskey macros.
5.	/MACROS:exename Displays all Doskey macros for the given executable.
6.	/HISTORY Displays all commands stored in memory.

7.	/INSERT Specifies that new text you type is inserted in old text.
8.	/OVERSTRIKE Specifies that new text overwrites old text.
9.	/EXENAME = exename Specifies the executable.
10.	/MACROFILE = filename Specifies a file of macros to install.
11.	macroname Specifies a name for a macro you create.
12.	text Specifies commands you want to record.

Example

Create a new file called keys.bat and enter the following commands in the file. The below commands creates two aliases, one if for the cd command, which automatically goes to the directory called test. And the other is for the dir command.

```
@echo off
doskey cd = cd/test
doskey d = dir
```

Once you execute the command, you will able to run these aliases in the command prompt.

Output

The following screenshot shows that after the above created batch file is executed, you can freely enter the 'd' command and it will give you the directory listing which means that your alias has been created.

```
C:\tp>d
Volume in drive C is Windows8_0S
Volume Serial Number is E41C-6F43

Directory of C:\tp

21/04/2016  02:57 AM    <DIR>
21/04/2016  02:57 AM    <DIR>
21/04/2016  02:57 AM                .
21/04/2016  02:58 AM                34 Keys.bat
21/04/2016  02:58 AM    <DIR>            28 Lists.cmd
12/28/2015  10:13 PM    <DIR>          newdir
12/28/2015  10:13 PM    <DIR>          newdir1
12/28/2015  10:13 PM    <DIR>          newdir2
                           2 File(s)           62 bytes
                           5 Dir(s)   161,492,418,560 bytes free

C:\tp>_
```

Deleting an Alias

An alias or macro can be deleted by setting the value of the macro to NULL.

Example

```
@echo off
doskey cd = cd/test
doskey d = dir
d=
```

In the above example, we are first setting the macro d to d = dir. After which we are setting it to NULL. Because we have set the value of d to NULL, the macro d will be deleted.

Replacing an Alias

An alias or macro can be replaced by setting the value of the macro to the new desired value.

Example

```
@echo off
doskey cd = cd/test
doskey d = dir
```

```
d = dir /w
```

In the above example, we are first setting the macro d to d = dir. After which we are setting it to dir /w. Since we have set the value of d to a new value, the alias 'd' will now take on the new value.

Batch Script - Devices

Windows now has an improved library which can be used in Batch Script for working with devices attached to the system. This is known as the device console – DevCon.exe.

Windows driver developers and testers can use DevCon to verify that a driver is installed and configured correctly, including the proper INF files, driver stack, driver files, and driver package. You can also use the DevCon commands (enable, disable, install, start, stop, and continue) in scripts to test the driver. **DevCon** is a command-line tool that performs device management functions on local computers and remote computers.

Display driver and device info DevCon can display the following properties of drivers and devices on local computers, and remote computers (running Windows XP and earlier) –

- Hardware IDs, compatible IDs, and device instance IDs. These identifiers are described in detail in device identification strings.
- Device setup classes.
- The devices in a device setup class.
- INF files and device driver files.
- Details of driver packages.
- Hardware resources.
- Device status.
- Expected driver stack.
- Third-party driver packages in the driver store.
- Search for devices DevCon can search for installed and uninstalled devices on a local or remote computer by hardware ID, device instance ID, or device setup class.
- Change device settings DevCon can change the status or configuration of Plug and Play (PnP) devices on the local computer in the following ways –

- Enable a device.
- Disable a device.
- Update drivers (interactive and non-interactive).
- Install a device (create a devnode and install software).
- Remove a device from the device tree and delete its device stack.
- Rescan for Plug and Play devices.
- Add, delete, and reorder the hardware IDs of root-enumerated devices.
- Change the upper and lower filter drivers for a device setup class.
- Add and delete third-party driver packages from the driver store.

DevCon (DevCon.exe) is included when you install the WDK, Visual Studio, and the Windows SDK for desktop apps. DevCon.exe kit is available in the following locations when installed.

```
%WindowsSdkDir%\tools\x64\devcon.exe  
%WindowsSdkDir%\tools\x86\devcon.exe  
%WindowsSdkDir%\tools\arm\devcon.exe
```

Syntax

```
devcon [/m:\computer] [/r] command [arguments]
```

wherein

- **/m:\computer** – Runs the command on the specified remote computer. The backslashes are required.
- **/r** – Conditional reboot. Reboots the system after completing an operation only if a reboot is required to make a change effective.
- **command** – Specifies a DevCon command.
- To list and display information about devices on the computer, use the following commands –
 - DevCon HwIDs
 - DevCon Classes

- DevCon ListClass
 - DevCon DriverFiles
 - DevCon DriverNodes
 - DevCon Resources
 - DevCon Stack
 - DevCon Status
 - DevCon Dp_enum
-
- To search for information about devices on the computer, use the following commands –
 - DevCon Find
 - DevCon FindAll
-
- To manipulate the device or change its configuration, use the following commands –
 - DevCon Enable
 - DevCon Disable
 - DevCon Update
 - DevCon UpdateNI
 - DevCon Install
 - DevCon Remove
 - DevCon Rescan
 - DevCon Restart
 - DevCon Reboot
 - DevCon SetHwID
 - DevCon ClassFilter
 - DevCon Dp_add
 - DevCon Dp_delete

Examples

Following are some examples on how the DevCon command is used.

List all driver files

The following command uses the DevCon DriverFiles operation to list the file names of drivers that devices on the system use. The command uses the wildcard character (*) to indicate all devices on the system. Because the output is extensive, the command uses the redirection character (>) to redirect the output to a reference file, driverfiles.txt.

```
devcon driverfiles * > driverfiles.txt
```

The following command uses the DevCon status operation to find the status of all devices on the local computer. It then saves the status in the status.txt file for logging or later review. The command uses the wildcard character (*) to represent all devices and the redirection character (>) to redirect the output to the status.txt file.

```
devcon status * > status.txt
```

The following command enables all printer devices on the computer by specifying the Printer setup class in a DevCon Enable command. The command includes the /r parameter, which reboots the system if it is necessary to make the enabling effective.

```
devcon /r enable = Printer
```

The following command uses the DevCon Install operation to install a keyboard device on the local computer. The command includes the full path to the INF file for the device (keyboard.inf) and a hardware ID (*PNP030b).

```
devcon /r install c:\windows\inf\keyboard.inf *PNP030b
```

The following command will scan the computer for new devices.

```
devcon scan
```

The following command will rescan the computer for new devices.

```
devcon rescan
```

Batch Script - Registry

The Registry is one of the key elements on a windows system. It contains a lot of information on various aspects of the operating system. Almost all applications installed on a windows system interact with the registry in some form or the other.

The Registry contains two basic elements: keys and values. **Registry keys** are container objects similar to folders. **Registry values** are non-container objects similar to files. Keys may contain values or further keys. Keys are referenced with a syntax similar to Windows' path names, using backslashes to indicate levels of hierarchy.

This chapter looks at various functions such as querying values, adding, deleting and editing values from the registry.

S.No	Types of Registry & Description
1	Reading from the Registry Reading from the registry is done via the REG QUERY command.
2	Adding to the Registry Adding to the registry is done via the REG ADD command.
3	Deleting from the Registry Deleting from the registry is done via the REG DEL command.
4	Copying Registry Keys Copying from the registry is done via the REG COPY command.
5	Comparing Registry Keys Comparing registry keys is done via the REG COMPARE command.

Batch Script - Network

Batch script has the facility to work with network settings. The NET command is used to update, fix, or view the network or network settings. This chapter looks at the different options available for the net command.

S.No	NET Commands & Description
1	NET ACCOUNTS View the current password & logon restrictions for the computer.

2	NET CONFIG Displays your current server or workgroup settings.
3	NET COMPUTER Adds or removes a computer attached to the windows domain controller.
4	NET USER This command can be used for the following View the details of a particular user account.
5	NET STOP/START This command is used to stop and start a particular service.
6	NET STATISTICS Display network statistics of the workstation or server.
7	NET USE Connects or disconnects your computer from a shared resource or displays information about your connections.

Batch Script - Printing

Printing can also be controlled from within Batch Script via the NET PRINT command.

Syntax

```
PRINT [/D:device] [[drive:][path]filename[...]]
```

Where /D:device - Specifies a print device.

Example

```
print c:\example.txt /c /d:lpt1
```

The above command will print the example.txt file to the parallel port lpt1.

Command Line Printer Control

As of Windows 2000, many, but not all, printer settings can be configured from Windows's command line using PRINTUI.DLL and RUNDLL32.EXE

Syntax

```
RUNDLL32.EXE PRINTUI.DLL,PrintUIEntry [ options ] [ @commandfile ]
```

Where some of the options available are the following –

- **/dl** – Delete local printer.
- **/dn** – Delete network printer connection.
- **/dd** – Delete printer driver.
- **/e** – Display printing preferences.
- **/f[file]** – Either inf file or output file.
- **/F[file]** – Location of an INF file that the INF file specified with /f may depend on.
- **/ia** – Install printer driver using inf file.
- **/id** – Install printer driver using add printer driver wizard.
- **/if** – Install printer using inf file.
- **/ii** – Install printer using add printer wizard with an inf file.
- **/il** – Install printer using add printer wizard.
- **/in** – Add network printer connection.
- **/ip** – Install printer using network printer installation wizard.
- **/k** – Print test page to specified printer, cannot be combined with command when installing a printer.
- **/I[path]** – Printer driver source path.
- **/m[model]** – Printer driver model name.
- **/n[name]** – Printer name.
- **/o** – Display printer queue view.
- **/p** – Display printer properties.
- **/Ss** – Store printer settings into a file.
- **/Sr** – Restore printer settings from a file.
- **/y** – Set printer as the default.
- **/Xg** – Get printer settings.
- **/Xs** – Set printer settings.

Testing if a Printer Exists

There can be cases wherein you might be connected to a network printer instead of a local printer. In such cases, it is always beneficial to check if a printer exists in the first place before printing.

The existence of a printer can be evaluated with the help of the RUNDLL32.EXE PRINTUI.DLL which is used to control most of the printer settings.

Example

```
SET PrinterName = Test Printer
SET file=%TEMP%\Prt.txt
RUNDLL32.EXE PRINTUI.DLL,PrintUIEntry /Xg /n "%PrinterName%" /f "%file%" /q

IF EXIST "%file%" (
    ECHO %PrinterName% printer exists
) ELSE (
    ECHO %PrinterName% printer does NOT exists
)
```

The above command will do the following –

- It will first set the printer name and set a file name which will hold the settings of the printer.
- The RUNDLL32.EXE PRINTUI.DLL commands will be used to check if the printer actually exists by sending the configuration settings of the file to the file Prt.txt

Batch Script - Debugging

Debugging a batch script becomes important when you are working on a big complex batch script.

Following are the ways in which you can debug the batch file.

Using echo command

A very simple debug option is to make use of echo command in your batch script wherever possible. It will display the message in the command prompt and help you debug where things have gone wrong.

Here is a simple example that displays even numbers based on the input given. The echo command is used to display the result and also if the input is not given. Similarly, the echo command can be used in place when you think that the error can happen. For example, if the input given is a negative number, less than 2, etc.

Example

```
@echo off
if [%1] == [] (
    echo input value not provided
    goto stop
)
rem Display numbers
for /l %%n in (2,2,%1) do (
    echo %%n
)
:stop
pause
```

Output

```
C:\>test.bat
10
2
4
6
8
10
22
Press any key to continue ...
```

Using pause command

Another way is to pause the batch execution when there is an error. When the script is paused, the developer can fix the issue and restart the processing.

In the example below, the batch script is paused as the input value is mandatory and not provided.

Example

```
@echo off
if [%1] == [] (
    echo input value not provided
    goto stop
) else (
    echo "Valid value"
)
:stop
pause
```

Output

```
C:\>test.bat
input value not provided
Press any key to continue..
```

Logging the error messages to another file

It might get hard to debug the error just looking at a bunch of echo displayed on the command prompt. Another easy way out is to log those messages in another file and view it step by step to understand what went wrong.

Here is an example, consider the following test.bat file:

```
net statistics /Server
```

The command given in the .bat file is wrong. Let us log the message and see what we get.

Execute the following command in your command line:

```
C:\>test.bat > testlog.txt 2> testerrors.txt
```

The file testerrors.txt will display the error messages as shown below:

```
The option /SERVER is unknown.  
The syntax of this command is:  
NET STATISTICS  
[WORKSTATION | SERVER]  
More help is available by typing NET HELPMSG 3506.
```

Looking at the above file the developer can fix the program and execute again.

Using ErrorLevel to detect errors and log them

Errorlevel returns 0 if the command executes successfully and 1 if it fails.

Consider the following example:

```
@echo off  
PING google.com  
if errorlevel 1 GOTO stop  
:stop  
    echo Unable to connect to google.com  
pause
```

During execution, you can see errors as well as logs:

```
C:\>test.bat > testlog.txt
```

testlog.txt

```
Pinging google.com [172.217.26.238] with 32 bytes of data:  
Reply from 172.217.26.238: bytes=32 time=160ms TTL=111  
Reply from 172.217.26.238: bytes=32 time=82ms TTL=111  
Reply from 172.217.26.238: bytes=32 time=121ms TTL=111  
Reply from 172.217.26.238: bytes=32 time=108ms TTL=111  
Ping statistics for 172.217.26.238:  
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),  
Approximate round trip times in milli-seconds:  
    Minimum = 82ms, Maximum = 160ms, Average = 117ms  
Connected successfully  
Press any key to continue . . .
```

In case of failure, you will see the following logs inside testlog.txt.

```
Ping request could not find host google.com. Please check the name and try again.  
Unable to connect to google.com  
Press any key to continue . . .
```

Batch Script - Logging

Logging in is possible in Batch Script by using the redirection command.

Syntax

```
test.bat > testlog.txt 2> testerrors.txt
```

Example

Create a file called test.bat and enter the following command in the file.

```
net statistics /Server
```

The above command has an error because the option to the net statistics command is given in the wrong way.

Output

If the command with the above test.bat file is run as

```
test.bat > testlog.txt 2> testerrors.txt
```

And you open the file testerrors.txt, you will see the following error.

```
The option /SERVER is unknown.
```

The syntax of this command is –

```
NET STATISTICS  
[WORKSTATION | SERVER]
```

More help is available by typing NET HELPMSG 3506.

If you open the file called testlog.txt, it will show you a log of what commands were executed.

```
C:\tp>net statistics /Server
```



AUDISANKARA COLLEGE OF ENGINEERING & TECHNOLOGY
(AUTONOMOUS)
Gudur, Nellore Dist, A.P (India)

DATA STRUCTURES

LECTURE NOTES

Dr.K VENKATA NAGENDRA

Mr.G.RAJESH

DATA STRUCTURES(18CS202)

OBJECTIVES:

The course should enable the students to :

1. Demonstrate familiarity with major algorithms and data structures.
2. Choose the appropriate data structure and algorithm design method for a specified application.
3. Determine which algorithm or data structure to use in different scenarios.
4. To improve the logical ability.

UNIT-I	INTRODUCTION TO ALGORITHMS AND DATA STRUCTURES	Classes:12
---------------	---	-------------------

Algorithms: Definition, Properties, Performance Analysis-Space Complexity, Time Complexity, Asymptotic Notations.**Data structures:** Introduction, Data Structures types, DS Operations.

UNIT-II	STACKS AND QUEUES	Classes:12
----------------	--------------------------	-------------------

Stacks: Introduction, Stack Operations, Applications: Infix to Postfix Conversion, Evaluation of Postfix Expression.**queues:** Introduction, Operations on queues, Circular queues, Priority queues.

UNIT-III	LINKED LISTS AND APPLICATIONS	Classes:12
-----------------	--------------------------------------	-------------------

Linked lists: Introduction, Singly linked lists, Circular linked lists, Doubly linked lists, Multiply linked lists, Applications: Polynomial Representation.Implementation of Stack and Queue using linked list.

UNIT-IV	SORTING AND SEARCHING	Classes:12
----------------	------------------------------	-------------------

Sorting: Introduction, Selection sort, Bubble sort, Insertion sort, Merge sort, Quick sort, Heap Sort.**Searching:** Introduction, Linear search, Binary search, Fibonacci search.

UNIT-V	TREES AND BINARY TREES	Classes:12
---------------	-------------------------------	-------------------

Trees: Introduction, Definition and basic terminologies, Representation of trees.

Binary Trees: Basic Terminologies and Types, Binary Tree Traversals, Binary Search Trees.

Text Books:

1. G.A.V PAI, Data Structures and Algorithms, Concepts, Techniques and Applications, Volume1, 1stEdition, Tata McGraw-Hill, 2008.
2. Richard F. Gilberg& Behrouz A. Forouzan, Data Structures, Pseudo code Approach with C, 2ndEdition, Cengage Learning India Edition, 2007.

Reference Books:

1. Langsam,M. J. Augenstein, A. M. Tanenbaum, Datastructures using C and C++, 2nd Edition, PHI Education, 2008.
2. Sartaj Sahni, Ellis Horowitz, Fundamentals of at Structures in C, 2nd Edition, Orientblackswan, 2010.

Web References:

1. <https://www.geeksforgeeks.org/data-structures/>
2. <https://www.programiz.com/dsa>
3. <https://www.w3schools.in/data-structures-tutorial/intro/>

Outcomes:

At the end of the course students able to

1. Apply Concepts of Stacks, Queues, Linked Lists.
2. Develop Programs for Searching and Sorting, Trees.
3. Interpret concepts of trees.
4. Develop programs for Sorting and Searching.

UNIT-I

INTRODUCTION TO ALGORITHMS AND DATA STRUCTURES

Definition: - An algorithm is a **Step By Step** process to solve a problem, where each step indicates an intermediate task. Algorithm contains finite number of steps that leads to the solution of the problem.

Properties /Characteristics of an Algorithm:-

Algorithm has the following basic properties

- **Input-Output:-** Algorithm takes '0' or more input and produces the required output. This is the basic characteristic of an algorithm.
- **Finiteness:-** An algorithm must terminate in countable number of steps.
- **Definiteness:** Each step of an algorithm must be stated clearly and unambiguously.
- **Effectiveness:** Each and every step in an algorithm can be converted in to programming language statement.
- **Generality:** Algorithm is generalized one. It works on all set of inputs and provides the required output. In other words it is not restricted to a single input value.

Categories of Algorithm:

Based on the different types of steps in an Algorithm, it can be divided into three categories, namely

- Sequence
- Selection and
- Iteration

Sequence: The steps described in an algorithm are performed successively one by one without skipping any step. The sequence of steps defined in an algorithm should be simple and easy to understand. Each instruction of such an algorithm is executed, because no selection procedure or conditional branching exists in a sequence algorithm.

Example:

```
// adding two numbers
```

Step 1: start

Step 2: read a,b

Step 3: Sum=a+b

Step 4: write Sum

Step 5: stop

Selection: The sequence type of algorithms are not sufficient to solve the problems, which involves decision and conditions. In order to solve the problem which involve decision making or option selection, we go for Selection type of algorithm. The general format of Selection type of statement is as shown below:

```
if(condition)
    Statement-1;
else
    Statement-2;
```

The above syntax specifies that if the condition is true, statement-1 will be executed otherwise statement-2 will be executed. In case the operation is unsuccessful. Then sequence of algorithm should be changed/ corrected in such a way that the system will re-execute until the operation is successful.

Example1:

```
// Person eligibility for vote
Step 1 : start
Step 2 : read age
Step 3 : if age > = 18 then step_4 else step_5
Step 4 : write "person is eligible for vote"
Step 5 : write " person is not eligible for vote"
Step 6 : stop
```

Example2:

```
// biggest among two numbers
Step 1 : start
Step 2 : read a,b
Step 3 : if a > b then
Step 4 : write "a is greater than b"
Step 5 : else
Step 6 : write "b is greater than a"
Step 7 : stop
```

Iteration: Iteration type algorithms are used in solving the problems which involves repetition of statement. In this type of algorithms, a particular number of statements are repeated 'n' no. of times.

Example1:

```
Step 1 : start
Step 2 : read n
Step 3 : repeat step 4 until n>0
Step 4 : (a) r=n mod 10
         (b) s=s+r
         (c) n=n/10
Step 5 : write s
Step 6 : stop
```

Performance Analysis an Algorithm:

The Efficiency of an Algorithm can be measured by the following metrics.

- i. Time Complexity and
- ii. Space Complexity.

i.Time Complexity:

The amount of time required for an algorithm to complete its execution is its time complexity. An algorithm is said to be efficient if it takes the minimum (reasonable) amount of time to complete its execution.

ii. Space Complexity:

The amount of space occupied by an algorithm is known as Space Complexity. An algorithm is said to be efficient if it occupies less space and required the minimum amount of time to complete its execution.

1. Write an algorithm for roots of a Quadratic Equation?

```
// Roots of a quadratic Equation
```

```
Step 1 : start
Step 2 : read a,b,c
Step 3 : if (a= 0) then step 4 else step 5
Step 4 : Write " Given equation is a linear equation "
Step 5 : d=(b * b) _ (4 *a *c)
Step 6 : if ( d>0) then step 7 else step8
Step 7 : Write " Roots are real and Distinct"
Step 8: if(d=0) then step 9 else step 10
Step 9: Write "Roots are real and equal"
Step 10: Write " Roots are Imaginary"
Step 11: stop
```

2. Write an algorithm to find the largest among three different numbers entered by user

Step 1: Start

Step 2: Declare variables a,b and c.

Step 3: Read variables a,b and c.

Step 4: If $a > b$

If $a > c$

 Display a is the largest number.

Else

 Display c is the largest number.

Else

If $b > c$

 Display b is the largest number.

Else

 Display c is the greatest number.

Step 5: Stop

3. Write an algorithm to find the factorial of a number entered by user.

Step 1: Start

Step 2: Declare variables n,factorial and i.

Step 3: Initialize variables

 factorial $\leftarrow 1$

 i $\leftarrow 1$

Step 4: Read value of n

Step 5: Repeat the steps until $i=n$

 5.1: factorial \leftarrow factorial*i

 5.2: i \leftarrow i+1

Step 6: Display factorial

Step 7: Stop

4. Write an algorithm to find the Simple Interest for given Time and Rate of Interest .

Step 1: Start

Step 2: Read P,R,S,T.

Step 3: Calculate $S=(PTR)/100$

Step 4: Print S

Step 5: Stop

ASYMPTOTIC NOTATIONS

Asymptotic analysis of an algorithm refers to defining the mathematical boundation/framing of its run-time performance. Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.

Asymptotic analysis is input bound i.e., if there's no input to the algorithm, it is concluded to work in a constant time. Other than the "input" all other factors are considered constant.

Asymptotic analysis refers to computing the running time of any operation in mathematical units of computation. For example, the running time of one operation is computed as $f(n)$ and may be for another operation it is computed as $g(n^2)$. This means the first operation running time will increase linearly with the increase in n and the running time of the second operation will increase exponentially when n increases. Similarly, the running time of both operations will be nearly the same if n is significantly small.

The time required by an algorithm falls under three types –

- **Best Case** – Minimum time required for program execution.
- **Average Case** – Average time required for program execution.
- **Worst Case** – Maximum time required for program execution.

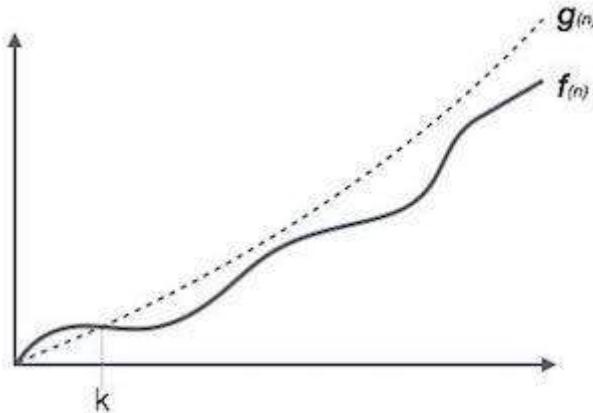
Asymptotic Notations

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

- O Notation
- Ω Notation
- Θ Notation

Big Oh Notation, O

The notation $O(n)$ is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.

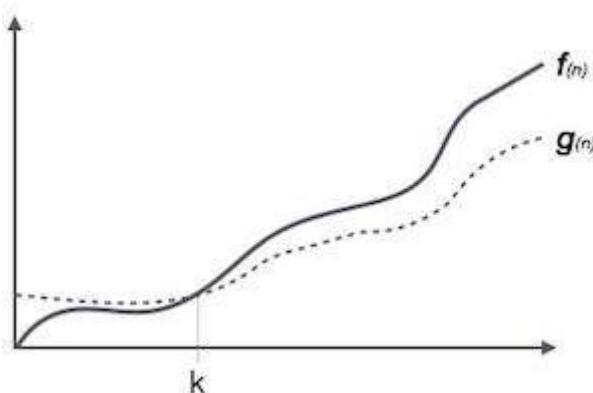


For example, for a function $f(n)$

$$O(f(n)) = \{ g(n) : \text{there exists } c > 0 \text{ and } n_0 \text{ such that } f(n) \leq c.g(n) \text{ for all } n > n_0. \}$$

Omega Notation, Ω

The notation $\Omega(n)$ is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.

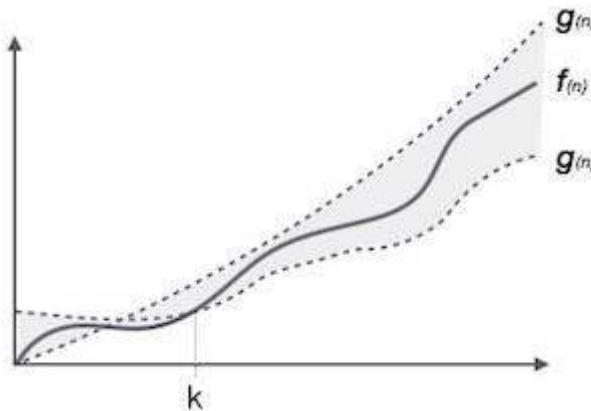


For example, for a function $f(n)$

$$\Omega(f(n)) \geq \{ g(n) : \text{there exists } c > 0 \text{ and } n_0 \text{ such that } g(n) \leq c.f(n) \text{ for all } n > n_0. \}$$

Theta Notation, Θ

The notation $\Theta(n)$ is the formal way to express both the lower bound and the upper bound of an algorithm's running time. It is represented as follows –



$\Theta(f(n)) = \{ g(n) \text{ if and only if } g(n) = O(f(n)) \text{ and } g(n) = \Omega(f(n)) \text{ for all } n > n_0. \}$

DATA STRUCTURES

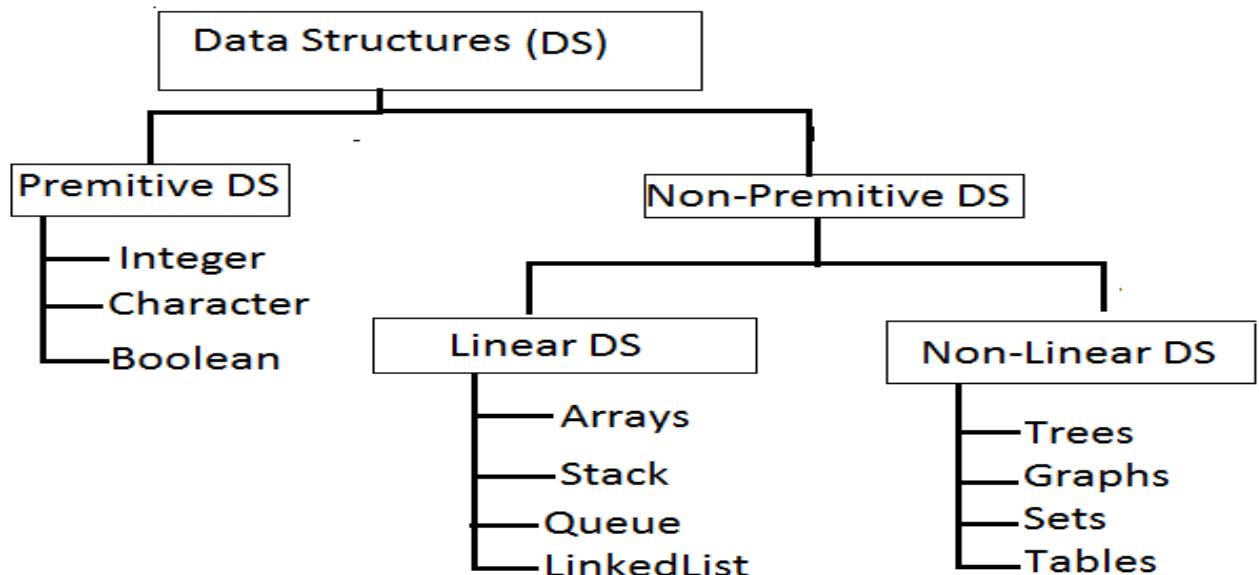
Data may be organized in many different ways logical or mathematical model of a program particularly organization of data. This organized data is called “Data Structure”.

Or

The organized collection of data is called a ‘Data Structure’.

Data Structure=Organized data +Allowed operations

Data Structure involves two complementary goals. The first goal is to identify and develop useful, mathematical entities and operations and to determine what class of problems can be solved by using these entities and operations. The second goal is to determine representation for those abstract entities to implement abstract operations on this concrete representation.



Primitive Data structures are directly supported by the language ie; any operation is directly performed in these data items.

Ex: integer, Character, Real numbers etc.

Non-primitive data types are not defined by the programming language, but are instead created by the programmer.

Linear data structures organize their data elements in a linear fashion, where data elements are attached one after the other. Linear data structures are very easy to implement, since the memory of the computer is also organized in a linear fashion. Some commonly used linear data structures are arrays, linked lists, stacks and queues.

In nonlinear data structures, data elements are not organized in a sequential fashion. Data structures like multidimensional arrays, trees, graphs, tables and sets are some examples of widely used nonlinear data structures.

Operations on the Data Structures:

Following operations can be performed on the data structures:

1. Traversing
2. Searching
3. Inserting
4. Deleting
5. Sorting
6. Merging

1. Traversing- It is used to access each data item exactly once so that it can be processed.

2. Searching- It is used to find out the location of the data item if it exists in the given collection of data items.

3. Inserting- It is used to add a new data item in the given collection of data items.

4. Deleting- It is used to delete an existing data item from the given collection of data items.

5. Sorting- It is used to arrange the data items in some order i.e. in ascending or descending order in case of numerical data and in dictionary order in case of alphanumeric data.

6. Merging- It is used to combine the data items of two sorted files into single file in the sorted form.

UNIT-II

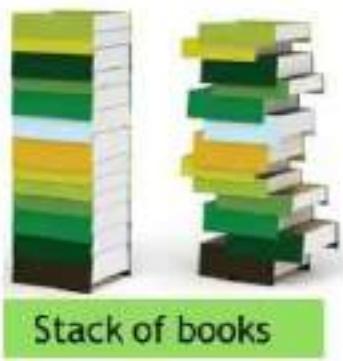
STACKS AND QUEUES

STACKS

A Stack is linear data structure. A stack is a list of elements in which an element may be inserted or deleted only at one end, called the **top of the stack**. Stack principle is **LIFO (last in, first out)**. Which element inserted last on to the stack that element deleted first from the stack.

As the items can be added or removed only from the top i.e. the last item to be added to a stack is the first item to be removed.

Real life examples of stacks are:



Stack of books



Stack of Plates



Stack of Toys

Operations on stack:

The two basic operations associated with stacks are:

1. Push
2. Pop

While performing push and pop operations the following test must be conducted on the stack.

- a) Stack is empty or not b) stack is full or not

1. Push: Push operation is used to add new elements in to the stack. At the time of addition first check the stack is full or not. If the stack is full it generates an error message "stack overflow".

2. Pop: Pop operation is used to delete elements from the stack. At the time of deletion first check the stack is empty or not. If the stack is empty it generates an error message "stack underflow".

All insertions and deletions take place at the same end, so the last element added to the stack will be the first element removed from the stack. When a stack is created, the stack base remains fixed while the stack top changes as elements are added and removed. The most accessible element is the top and the least accessible element is the bottom of the stack.

Representation of Stack (or) Implementation of stack:

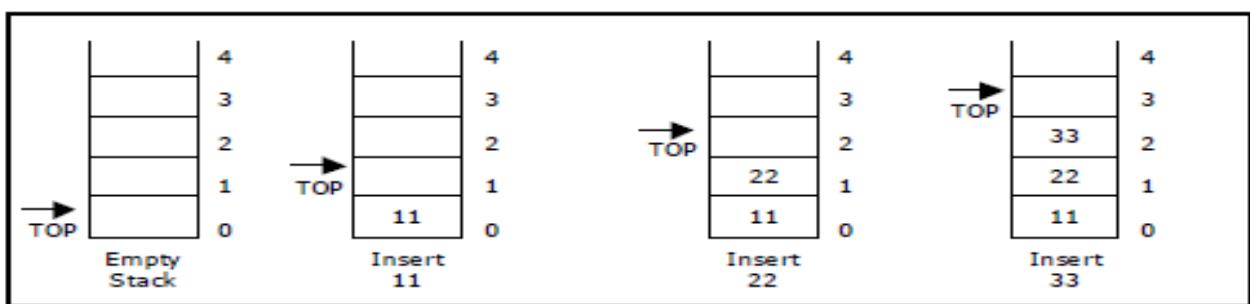
The stack should be represented in two ways:

1. Stack using array
2. Stack using linked list

1. Stack using array:

Let us consider a stack with 6 elements capacity. This is called as the size of the stack. The number of elements to be added should not exceed the maximum size of the stack. If we attempt to add new element beyond the maximum size, we will encounter a **stack overflow** condition. Similarly, you cannot remove elements beyond the base of the stack. If such is the case, we will reach a **stack underflow** condition.

1.push(): When an element is added to a stack, the operation is performed by push(). Below Figure shows the creation of a stack and addition of elements using push().



Initially **top=-1**, we can insert an element in to the stack, increment the top value i.e **top=top+1**. We can insert an element in to the stack first check the condition is stack is full or not. i.e **top>=size-1**. Otherwise add the element in to the stack.

```
void push()
{
    int x;
    if(top >= n-1)
    {
        printf("\n\nStack
Overflow..");
        return;
    }
    else
    {
        printf("\n\nEnter data: ");
        scanf("%d", &x);
        stack[top] = x;
        top = top + 1;
        printf("\n\nData Pushed into
the stack");
    }
}
```

Algorithm: Procedure for push():

Step 1: START
Step 2: if top>=size-1 then
 Write “ Stack is Overflow”
Step 3: Otherwise
 3.1: read data value ‘x’
 3.2: top=top+1;
 3.3: stack[top]=x;
Step 4: END

2.Pop(): When an element is taken off from the stack, the operation is performed by pop(). Below figure shows a stack initially with three elements and shows the deletion of elements using pop().

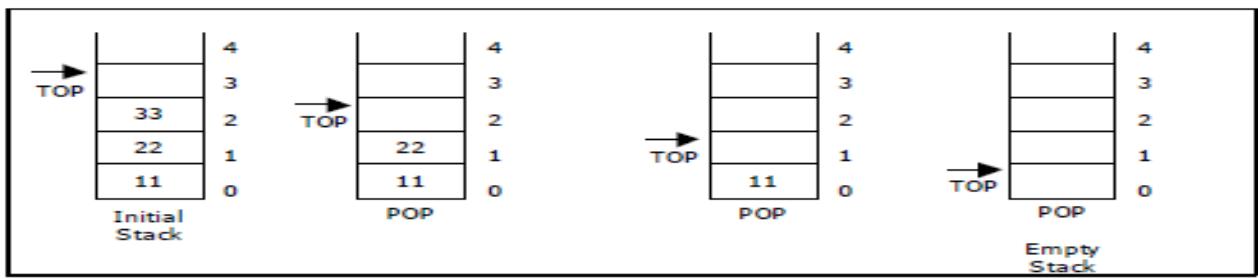


Figure Pop operations on stack

We can insert an element from the stack, decrement the top value i.e **top=top-1**.

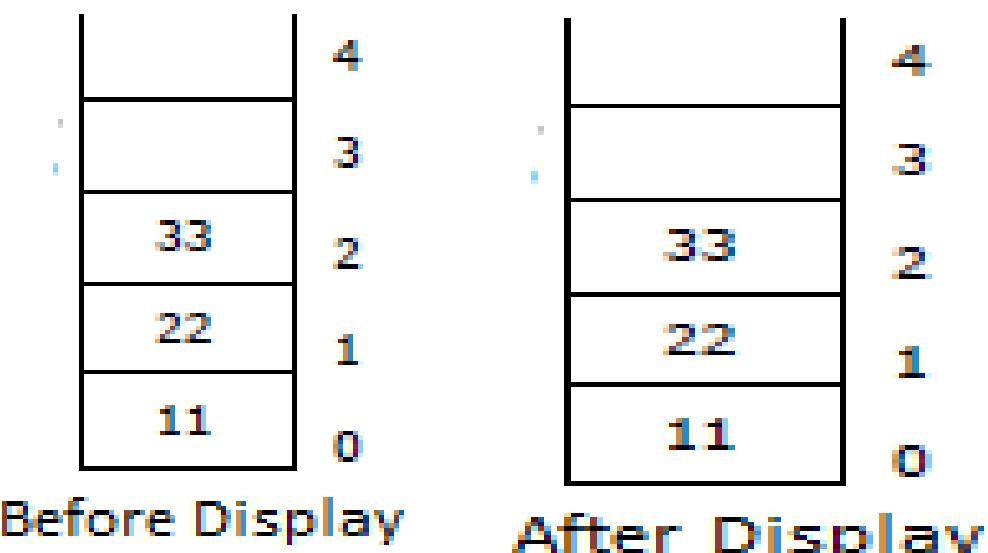
We can delete an element from the stack first check the condition is stack is empty or not. i.e **top== -1**. Otherwise remove the element from the stack.

```
Void pop()
{
    If(top== -1)
    {
        Printf("Stack is Underflow");
    }
    else
    {
        printf("Delete data %d",stack[top]);
        top=top-1;
    }
}
```

Algorithm: procedure pop():

```
Step 1: START
Step 2: if top== -1 then
        Write "Stack is Underflow"
Step 3: otherwise
        3.1: print "deleted element"
        3.2: top=top-1;
Step 4: END
```

3.display(): This operation performed display the elements in the stack. We display the element in the stack check the condition is stack is empty or not i.e **top== -1**.Otherwise display the list of elements in the stack.



```

void display()
{
    If(top== -1)
    {
        Printf("Stack is Underflow");
    }
    else
    {
        printf("Display elements are:");
        for(i=top;i>=0;i--)
            printf("%d",stack[i]);
    }
}

```

Algorithm: procedure pop():

Step 1: START
Step 2: if top== -1 then
 Write "Stack is Underflow"
Step 3: otherwise
 3.1: print "Display elements are"
 3.2: for top to 0
 Print 'stack[i]'
Step 4: END

Source code for stack operations, using array:

```

#include<stdio.h>
#include<conio.h>
int stack[100],choice,n,top,x,i;
void push(void);
void pop(void);
void display(void);
int main()
{
    //clrscr();
    top=-1;
    printf("\n Enter the size of STACK[MAX=100]:");
    scanf("%d",&n);
    printf("\n\t STACK OPERATIONS USING ARRAY");
    printf("\n\t-----");
    printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
    do
    {
        printf("\n Enter the Choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            {
                push();
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {

```

```

        display();
        break;
    }
    case 4:
    {
        printf("\n\t EXIT POINT ");
        break;
    }
    default:
    {
        printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
    }

}
}

while(choice!=4);
return 0;
}

void push()
{
    if(top>=n-1)
    {
        printf("\n\tSTACK is over flow");

    }
    else
    {
        printf(" Enter a value to be pushed:");
        scanf("%d",&x);
        top++;
        stack[top]=x;
    }
}

void pop()
{
    if(top<=-1)
    {
        printf("\n\t Stack is under flow");
    }
    else
    {
        printf("\n\t The popped elements is %d",stack[top]);
        top--;
    }
}

void display()
{
    if(top>=0)
    {

```

```

printf("\n The elements in STACK \n");
for(i=top; i>=0; i--)
    printf("\n%d",stack[i]);
printf("\n Press Next Choice");
}
else
{
    printf("\n The STACK is empty");
}
}

```

2. Stack using Linked List:

We can represent a stack as a linked list. In a stack push and pop operations are performed at one end called top. We can perform similar operations at one end of list using top pointer. The linked stack looks as shown in figure.

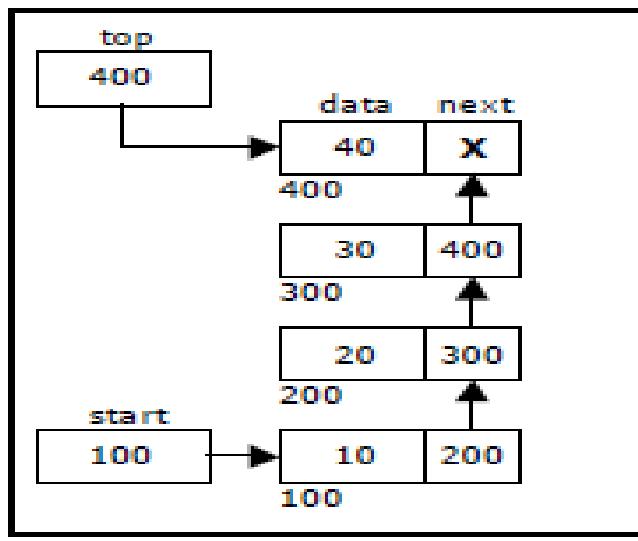


Figure **Linked stack representation**

Applications of stack:

1. Stack is used by compilers to check for balancing of parentheses, brackets and braces.
2. Stack is used to evaluate a postfix expression.
3. Stack is used to convert an infix expression into postfix/prefix form.
4. In recursion, all intermediate arguments and return values are stored on the processor's stack.
5. During a function call the return address and arguments are pushed onto a stack and on return they are popped off.

Converting and evaluating Algebraic expressions:

An **algebraic expression** is a legal combination of operators and operands. Operand is the quantity on which a mathematical operation is performed. Operand may be a variable like x, y, z or a constant like 5, 4, 6 etc. Operator is a symbol which signifies a mathematical or logical operation between the operands. Examples of familiar operators include +, -, *, /, ^ etc.

An algebraic expression can be represented using three different notations. They are infix, postfix and prefix notations:

Infix: It is the form of an arithmetic expression in which we fix (place) the arithmetic operator in between the two operands.

Example: A + B

Prefix: It is the form of an arithmetic notation in which we fix (place) the arithmetic operator before (pre) its two operands. The prefix notation is called as polish notation.

Example: + A B

Postfix: It is the form of an arithmetic expression in which we fix (place) the arithmetic operator after (post) its two operands. The postfix notation is called as *suffix notation* and is also referred to *reverse polish notation*.

Example: A B +

Conversion from infix to postfix:

Procedure to convert from infix expression to postfix expression is as follows:

1. Scan the infix expression from left to right.
2. a) If the scanned symbol is left parenthesis, push it onto the stack.
 - b) If the scanned symbol is an operand, then place directly in the postfix expression (output).
 - c) If the symbol scanned is a right parenthesis, then go on popping all the items from the stack and place them in the postfix expression till we get the matching left parenthesis.
 - d) If the scanned symbol is an operator, then go on removing all the operators from the stack and place them in the postfix expression, if and only if the precedence of the operator which is on the top of the stack is greater than (*or greater than or equal*) to the precedence of the scanned operator and push the scanned operator onto the stack otherwise, push the scanned operator onto the stack.

The three important features of postfix expression are:

1. The operands maintain the same order as in the equivalent infix expression.
2. The parentheses are not needed to designate the expression unambiguously.
3. While evaluating the postfix expression the priority of the operators is no longer relevant.

We consider five binary operations: +, -, *, / and \$ or \uparrow (exponentiation). For these binary operations, the following in the order of precedence (highest to lowest):

OPERATOR	PRECEDENCE	VALUE
Exponentiation (\$ or \uparrow or \wedge)	Highest	3
*, /	Next highest	2
+, -	Lowest	1

Example 1:

Convert $((A - (B + C)) * D) \uparrow (E + F)$ infix expression to postfix form:

SYMBOL	POSTFIX STRING	STACK	REMARKS
((
(((
A	A	((
-	A	((-	
(A	((- (
B	A B	((- (
+	A B	((- (+	
C	A B C	((- (+	
)	A B C +	((-	
)	A B C + -	(
*	A B C + -	(*	
D	A B C + - D	(*	
)	A B C + - D *		
\uparrow	A B C + - D *	\uparrow	
(A B C + - D *	\uparrow (
E	A B C + - D * E	\uparrow (
+	A B C + - D * E	\uparrow (+	
F	A B C + - D * E F	\uparrow (+	
)	A B C + - D * E F +	\uparrow	
End of string	A B C + - D * E F + \uparrow	The input is now empty. Pop the output symbols from the stack until it is empty.	

Example 2:

Convert the following infix expression $A + B * C - D / E * H$ into its equivalent postfix expression.

SYMBOL	POSTFIX STRING	STACK	REMARKS
A	A		
+	A	+	
B	A B	+	
*	A B	+ *	
C	A B C	+ *	
-	A B C * +	-	
D	A B C * + D	-	
/	A B C * + D	- /	
E	A B C * + D E	- /	
*	A B C * + D E /	- * /	
H	A B C * + D E / H	- * /	
End of string	A B C * + D E / H * -	The input is now empty. Pop the output symbols from the stack until it is empty.	

Evaluation of postfix expression:

The postfix expression is evaluated easily by the use of a stack.

1. When a number is seen, it is pushed onto the stack;
2. When an operator is seen, the operator is applied to the two numbers that are popped from the stack and the result is pushed onto the stack.
3. When an expression is given in postfix notation, there is no need to know any precedence rules; this is our obvious advantage.

Example 1:

Evaluate the postfix expression: 6 5 2 3 + 8 * + 3 + *

SYMBOL	OPERAND 1	OPERAND 2	VALUE	STACK	REMARKS
6				6	
5				6, 5	
2				6, 5, 2	
3				6, 5, 2, 3	The first four symbols are placed on the stack.
+	2	3	5	6, 5, 5	Next a '+' is read, so 3 and 2 are popped from the stack and their sum 5, is pushed
8	2	3	5	6, 5, 5, 8	Next 8 is pushed
*	5	8	40	6, 5, 40	Now a '*' is seen, so 8 and 5 are popped as $8 * 5 = 40$ is pushed
+	5	40	45	6, 45	Next, a '+' is seen, so 40 and 5 are popped and $40 + 5 = 45$ is pushed
3	5	40	45	6, 45, 3	Now, 3 is pushed
+	45	3	48	6, 48	Next, '+' pops 3 and 45 and pushes $45 + 3 = 48$ is pushed
*	6	48	288	288	Finally, a '*' is seen and 48 and 6 are popped, the result $6 * 48 = 288$ is pushed

Example 2:

Evaluate the following postfix expression: 6 2 3 + - 3 8 2 / + * 2 ↑ 3 +

SYMBOL	OPERAND 1	OPERAND 2	VALUE	STACK
6				6
2				6, 2
3				6, 2, 3
+	2	3	5	6, 5
-	6	5	1	1
3	6	5	1	1, 3
8	6	5	1	1, 3, 8
2	6	5	1	1, 3, 8, 2
/	8	2	4	1, 3, 4
+	3	4	7	1, 7
*	1	7	7	7
2	1	7	7	7, 2
↑	7	2	49	49
3	7	2	49	49, 3
+	49	3	52	52

QUEUE

A queue is linear data structure and collection of elements. A queue is another special kind of list, where items are inserted at one end called the **rear** and deleted at the other end called the **front**. The principle of queue is a “**FIFO**” or “**First-in-first-out**”.

Queue is an abstract data structure. A queue is a useful data structure in programming. It is similar to the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket.

A real-world example of queue can be a single-lane one-way road, where the vehicle enters first, exits first.



More real-world examples can be seen as queues at the ticket windows and bus-stops and our college library.



The operations for a queue are analogues to those for a stack; the difference is that the insertions go at the end of the list, rather than the beginning.

Operations on QUEUE:

A queue is an object or more specifically an abstract data structure (ADT) that allows the following operations:

- **Enqueue or insertion:** which inserts an element at the end of the queue.
- **Dequeue or deletion:** which deletes an element at the start of the queue.

Queue operations work as follows:

1. Two pointers called FRONT and REAR are used to keep track of the first and last elements in the queue.
2. When initializing the queue, we set the value of FRONT and REAR to 0.
3. On enqueueing an element, we increase the value of REAR index and place the new element in the position pointed to by REAR.
4. On dequeuing an element, we return the value pointed to by FRONT and increase the FRONT index.
5. Before enqueueing, we check if queue is already full.
6. Before dequeuing, we check if queue is already empty.
7. When enqueueing the first element, we set the value of FRONT to 1.
8. When dequeuing the last element, we reset the values of FRONT and REAR to 0.

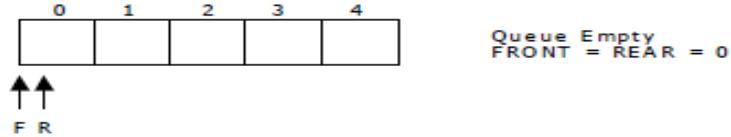
Representation of Queue (or) Implementation of Queue:

The queue can be represented in two ways:

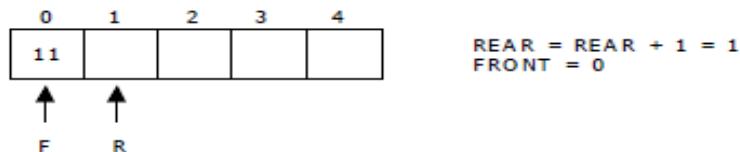
1. Queue using Array
2. Queue using Linked List

1.Queue using Array:

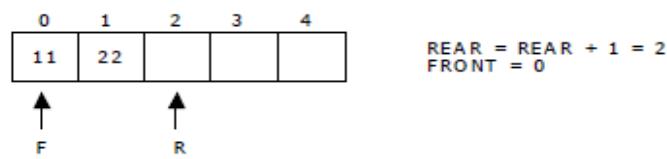
Let us consider a queue, which can hold maximum of five elements. Initially the queue is empty.



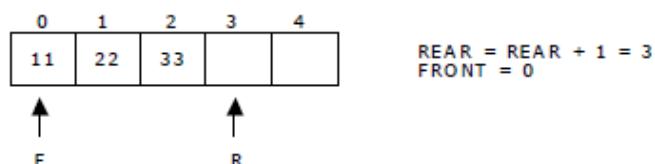
Now, insert 11 to the queue. Then queue status will be:



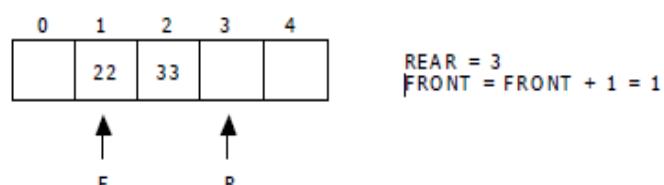
Next, insert 22 to the queue. Then the queue status is:



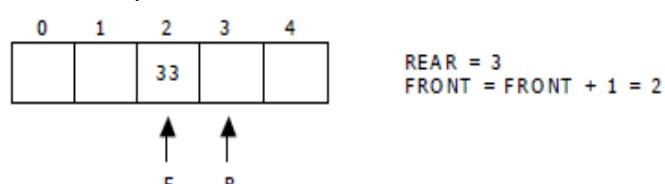
Again insert another element 33 to the queue. The status of the queue is:



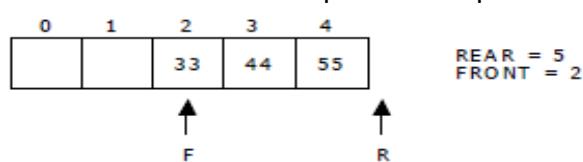
Now, delete an element. The element deleted is the element at the front of the queue. So the status of the queue is:



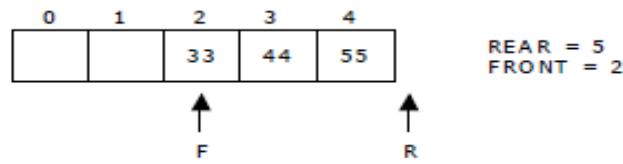
Again, delete an element. The element to be deleted is always pointed to by the FRONT pointer. So, 22 is deleted. The queue status is as follows:



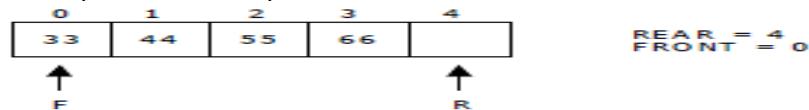
Now, insert new elements 44 and 55 into the queue. The queue status is:



Next insert another element, say 66 to the queue. We cannot insert 66 to the queue as the rear crossed the maximum size of the queue (i.e., 5). There will be queue full signal. The queue status is as follows:



Now it is not possible to insert an element 66 even though there are two vacant positions in the linear queue. To overcome this problem the elements of the queue are to be shifted towards the beginning of the queue so that it creates vacant position at the rear end. Then the FRONT and REAR are to be adjusted properly. The element 66 can be inserted at the rear end. After this operation, the queue status is as follows:



This difficulty can overcome if we treat queue position with index 0 as a position that comes after position with index 4 i.e., we treat the queue as a **circular queue**.

Queue operations using array:

a.**enqueue()** or **insertion()**: which inserts an element at the end of the queue.

```
void insertion()
{
    if(rear==max)
        printf("\n Queue is Full");
    else
    {
        printf("\n Enter no %d:",j++);
        scanf("%d",&queue[rear++]);
    }
}
```

Algorithm: Procedure for insertion():

Step-1:START
Step-2: if rear==max then
 Write 'Queue is full'
Step-3: otherwise
 3.1: read element 'queue[rear]'
Step-4:STOP

b.**dequeue()** or **deletion()**: which deletes an element at the start of the queue.

```
void deletion()
{
    if(front==rear)
    {
        printf("\n Queue is empty");
    }
    else
    {
        printf("\n Deleted Element is
               %d",queue[front++]);
        x++;
    }
}
```

Algorithm: procedure for deletion():

Step-1:START
Step-2: if front==rear then
 Write ' Queue is empty'
Step-3: otherwise
 3.1: print deleted element
Step-4:STOP

c.display(): which displays all elements in the queue.

```
void deletion()
{
    if(front==rear)
    {
        printf("\n Queue is empty");
    }
    else
    {
        for(i=front; i<rear; i++)
        {
            printf("%d",queue[i]);
            printf("\n");
        }
    }
}
```

Algorithm: procedure for deletion():
Step-1:START
Step-2: if front==rear then
 Write' Queue is empty'
Step-3: otherwise
 3.1: for i=front to rear then
 3.2: print 'queue[i]'
Step-4:STOP

2. Queue using Linked list:

We can represent a queue as a linked list. In a queue data is deleted from the front end and inserted at the rear end. We can perform similar operations on the two ends of a list. We use two pointers *front* and *rear* for our linked queue implementation.

The linked queue looks as shown in figure:

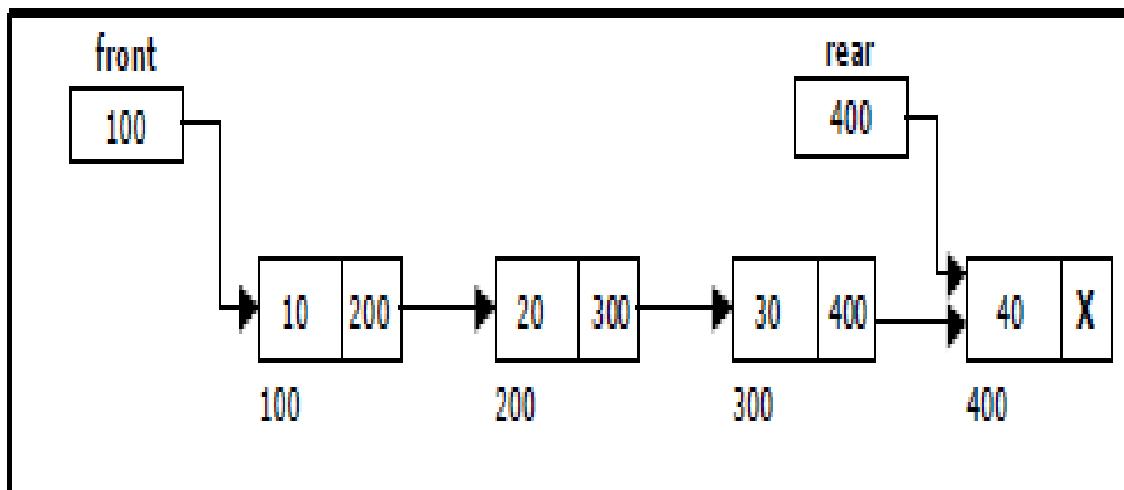


Figure : Linked Queue representation

Applications of Queue:

1. It is used to schedule the jobs to be processed by the CPU.
2. When multiple users send print jobs to a printer, each printing job is kept in the printing queue. Then the printer prints those jobs according to first in first out (FIFO) basis.
3. Breadth first search uses a queue data structure to find an element from a graph.

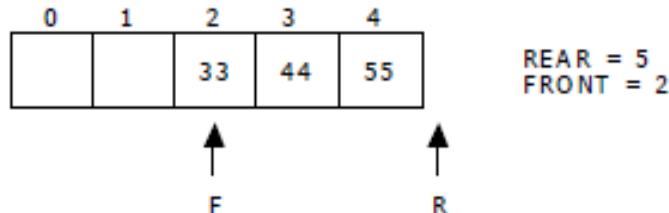
CIRCULAR QUEUE

A more efficient queue representation is obtained by regarding the array Q[MAX] as circular. Any number of items could be placed on the queue. This implementation of a queue is called a circular queue because it uses its storage array as if it were a circle instead of a linear list.

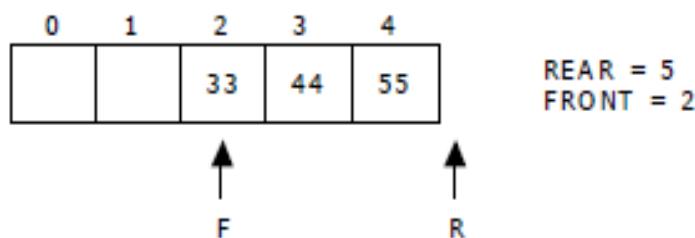
There are two problems associated with linear queue. They are:

- Time consuming: linear time to be spent in shifting the elements to the beginning of the queue.
- Signaling queue full: even if the queue is having vacant position.

For example, let us consider a linear queue status as follows:



Next insert another element, say 66 to the queue. We cannot insert 66 to the queue as the rear crossed the maximum size of the queue (i.e., 5). There will be queue full signal. The queue status is as follows:

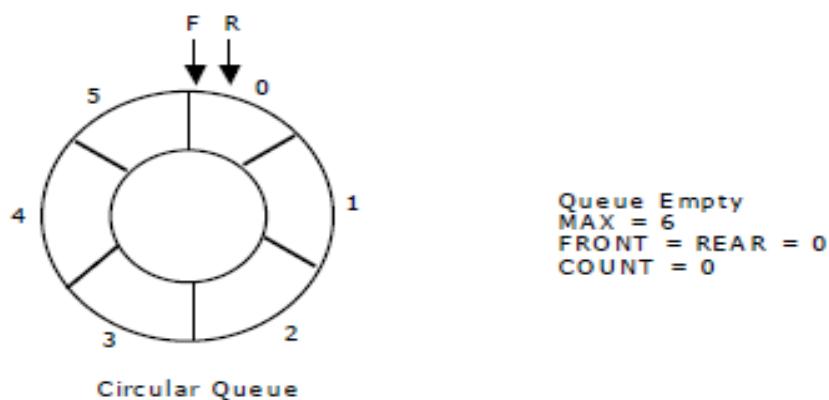


This difficulty can be overcome if we treat queue position with index zero as a position that comes after position with index four then we treat the queue as a **circular queue**.

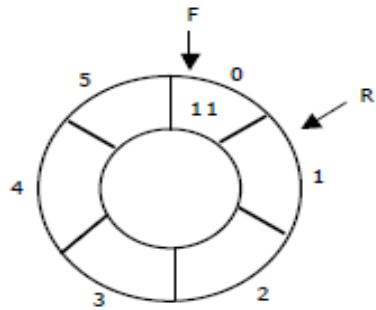
In circular queue if we reach the end for inserting elements to it, it is possible to insert new elements if the slots at the beginning of the circular queue are empty.

Representation of Circular Queue:

Let us consider a circular queue, which can hold maximum (MAX) of six elements. Initially the queue is empty.



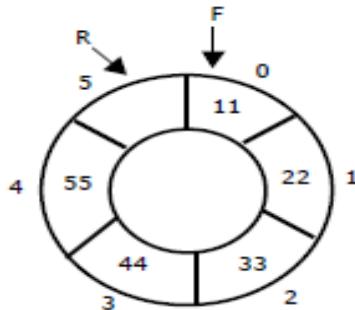
Now, insert 11 to the circular queue. Then circular queue status will be:



`FRONT = 0
REAR = (REAR + 1) % 6 = 1
COUNT = 1`

Circular Queue

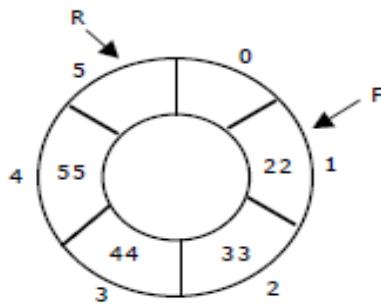
Insert new elements 22, 33, 44 and 55 into the circular queue. The circular queue status is:



`FRONT = 0
REAR = (REAR + 1) % 6 = 5
COUNT = 5`

Circular Queue

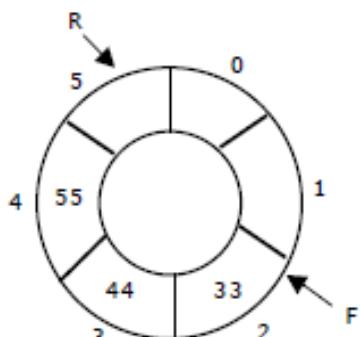
Now, delete an element. The element deleted is the element at the front of the circular queue. So, 11 is deleted. The circular queue status is as follows:



`FRONT = (FRONT + 1) % 6 = 1
REAR = 5
COUNT = COUNT - 1 = 4`

Circular Queue

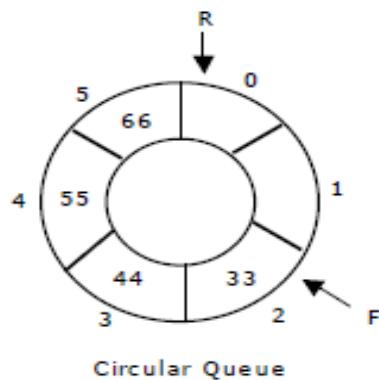
Again, delete an element. The element to be deleted is always pointed to by the FRONT pointer. So, 22 is deleted. The circular queue status is as follows:



`FRONT = (FRONT + 1) % 6 = 2
REAR = 5
COUNT = COUNT - 1 = 3`

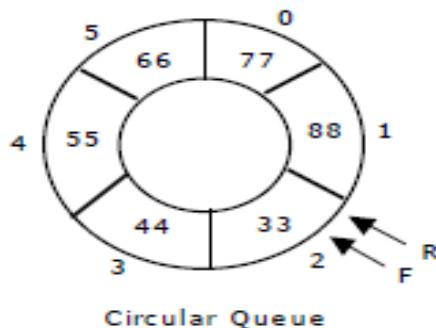
Circular Queue

Again, insert another element 66 to the circular queue. The status of the circular queue is:



```
FRONT = 2
REAR = (REAR + 1) % 6 = 0
COUNT = COUNT + 1 = 4
```

Now, insert new elements 77 and 88 into the circular queue. The circular queue status is:



```
FRONT = 2, REAR = 2
REAR = REAR % 6 = 2
COUNT = 6
```

Now, if we insert an element to the circular queue, as COUNT = MAX we cannot add the element to circular queue. So, the circular queue is *full*.

Operations on Circular queue:

a.enqueue() or insertion(): This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at Rear position.

```
void insertCQ()
{
    int data;
    if(count ==MAX)
    {
        printf("\n Circular Queue is Full");
    }
    else
    {
        printf("\n Enter data: ");
        scanf("%d", &data);
        CQ[rear] = data;
        rear = (rear + 1) % MAX;
        count++;
    }
    printf("\n Data Inserted in the Circular Queue ");
}
```

Algorithm: procedure of insertCQ():

```
Step-1:START
Step-2: if count==MAX then
        Write "Circular queue is full"
Step-3:otherwise
    3.1: read the data element
    3.2: CQ[rear]=data
    3.3: rear=(rear+1)%MAX
    3.4: count=count+1
Step-4:STOP
```

b.dequeue() or deletion():This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from front position.

```
void deleteCQ()
{
if(count ==0)
{
printf("\n\nCircular Queue is Empty..");
}
else
{
printf("\n Deleted element from Circular
Queue is %d ", CQ[front]);
front = (front + 1) % MAX;
count--;
}
}
```

Algorithm: procedure of deleteCQ():

Step-1:START
Step-2: if count==0 then
 Write “Circular queue is empty”
Step-3:otherwise
 3.1: print the deleted element
 3.2: front=(front+1)%MAX
 3.3: count=count-1
Step-4:STOP

c.display():This function is used to display the list of elements in the circular queue.

```
void displayCQ()
{
int i, j;
if(count ==0)
{
printf("\n\n\t Circular Queue is Empty ");
}
else
{
printf("\n Elements in Circular Queue are:
");
j = count;
for(i = front; j != 0; j--)
{
printf("%d\t", CQ[i]);
i = (i + 1) % MAX;
}
}
}
```

Algorithm: procedure of displayCQ():

Step-1:START
Step-2: if count==0 then
 Write “Circular queue is empty”
Step-3:otherwise
 3.1: print the list of elements
 3.2: for i=front to j!=0
 3.3: print CQ[i]
 3.4: i=(i+1)%MAX
Step-4:STOP

Deque:

In the preceding section we saw that a queue in which we insert items at one end and from which we remove items at the other end. In this section we examine an extension of the queue, which provides a means to insert and remove items at both ends of the queue. This data structure is a **deque**. The word **deque** is an acronym derived from **double-ended queue**. Below figure shows the representation of a deque.

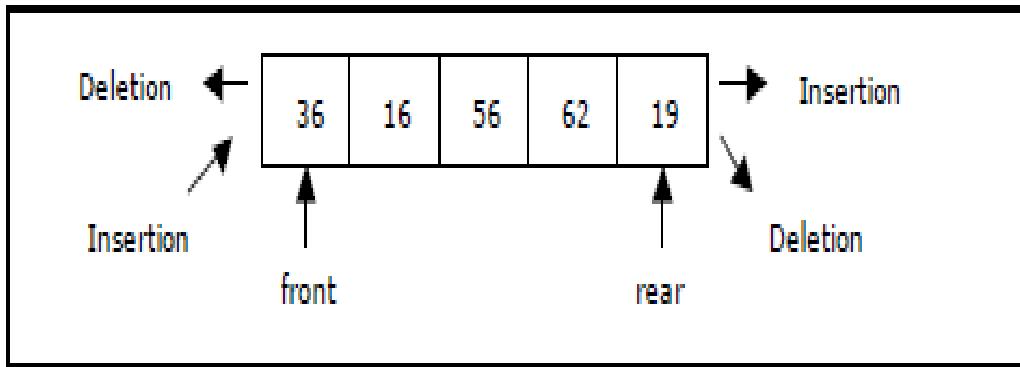


Figure Representation of a deque.

deque provides four operations. Below Figure shows the basic operations on a deque.

- enqueue_front: insert an element at front.
- dequeue_front: delete an element at front.
- enqueue_rear: insert element at rear.
- dequeue_rear: delete element at rear.

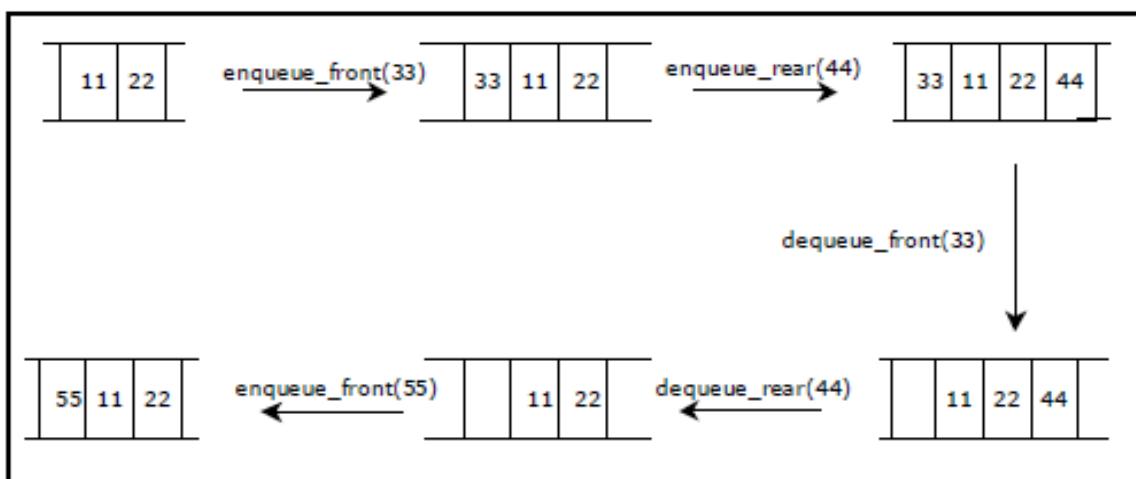


Figure 7. Basic operations on deque

There are two variations of deque. They are:

- Input restricted deque (IRD)
- Output restricted deque (ORD)

An Input restricted deque is a deque, which allows insertions at one end but allows deletions at both ends of the list.

An output restricted deque is a deque, which allows deletions at one end but allows insertions at both ends of the list.

Priority Queue:

A **priority queue** is a collection of elements such that each element has been assigned a priority. We can insert an element in priority queue at the rare position. We can delete an element from the priority queue based on the elements priority and such that the order in which elements are deleted and processed comes from the following rules:

1. An element of higher priority is processed before any element of lower priority.
2. Two elements with same priority are processed according to the order in which they were added to the queue. It follows FIFO or FCFS(First Comes First serve) rules.

We always remove an element with the highest priority, which is given by the minimal integer priority assigned.



A prototype of a priority queue is time sharing system: programs of high priority are processed first, and programs with the same priority form a standard queue. An efficient implementation for the Priority Queue is to use heap, which in turn can be used for sorting purpose called heap sort

Priority queues are two types:

1. Ascending order priority queue
2. Descending order priority queue

1. Ascending order priority queue: It is Lower priority number to high priority number.

Examples: order is 1,2,3,4,5,6,7,8,9,10

2. Descending order priority queue: It is high priority number to lowest priority number.

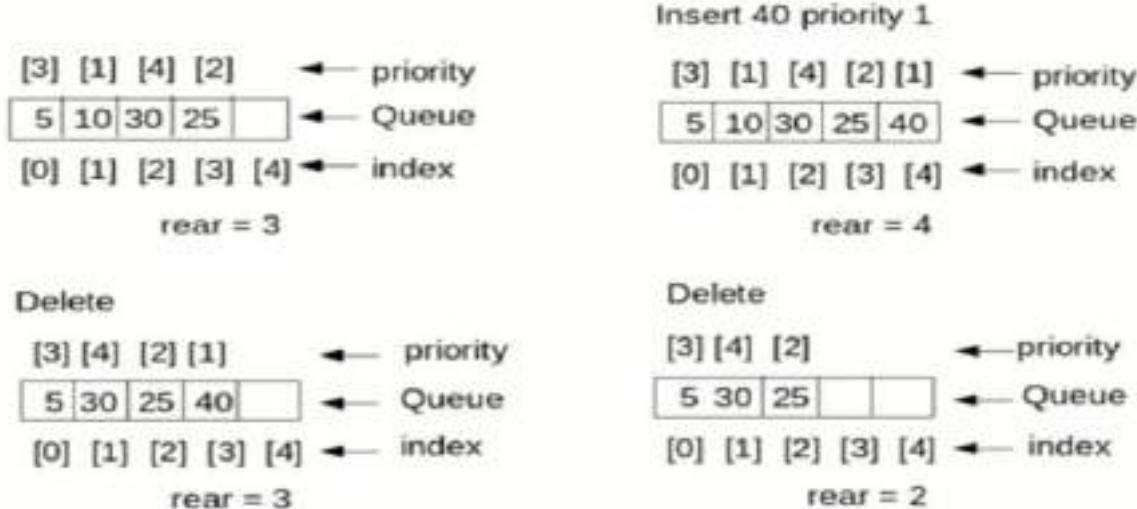
Examples: Order is 10,9,8,7,6,5,4,3,2,1

Implementation of Priority Queue:

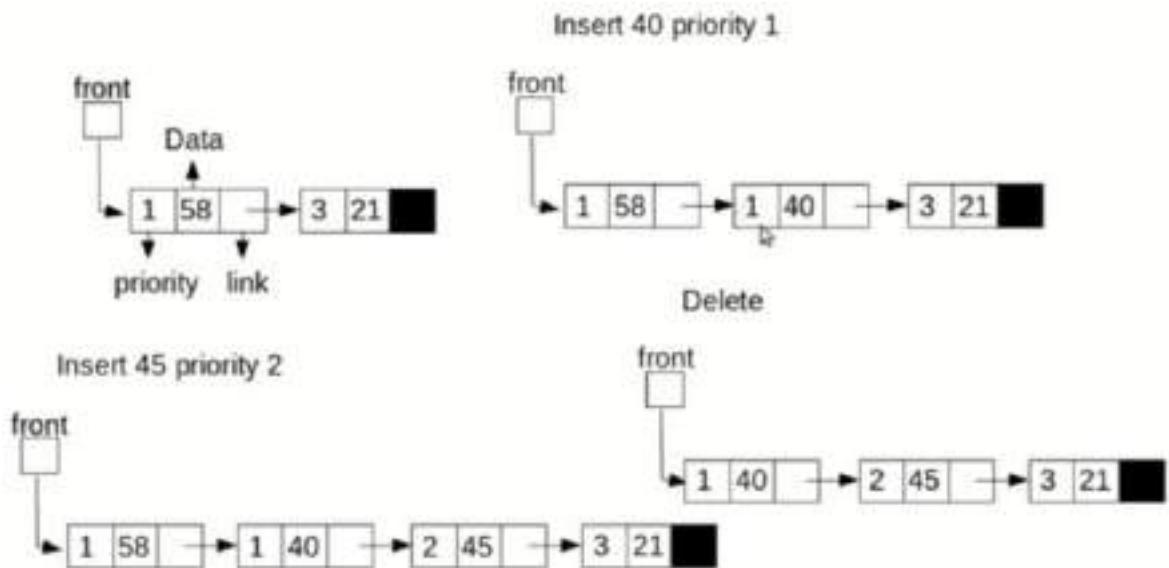
Implementation of priority queues are two types:

1. Through Queue(Using Array)
2. Through Sorted List(Using Linked List)

1. Through Queue (Using Array): In this case element is simply added at the rear end as usual. For deletion, the element with highest priority is searched and then deleted.



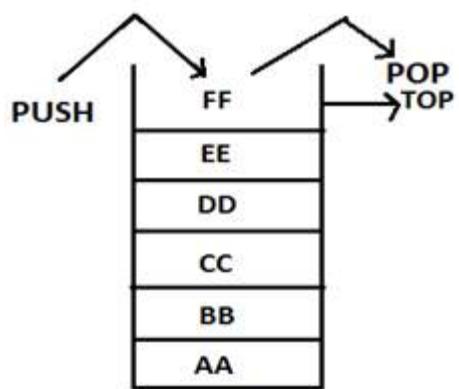
2. Through sorted List (Using Linked List): In this case insertion is costly because the element insert at the proper place in the list based on the priority. Here deletion is easy since the element with highest priority will always be in the beginning of the list.



1. Difference between stacks and Queues?

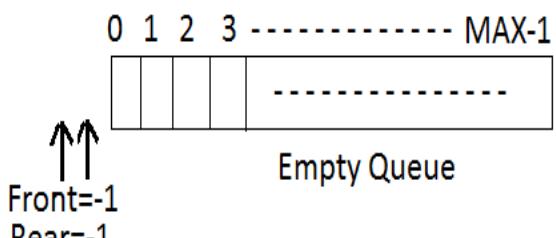
stacks	Queues
<p>1.A stack is a linear list of elements in which the element may be inserted or deleted at one end.</p> <p>2. In stacks, elements which are inserted last is the first element to be deleted.</p> <p>3.Stacks are called LIFO (Last In First Out)list</p> <p>4.In stack elements are removed in reverse order in which thy are inserted.</p> <p>5.suppose the elements a,b,c,d,e are inserted in the stack, the deletion of elements will be e,d,c,b,a.</p> <p>6.In stack there is only one pointer to insert and delete called "Top".</p> <p>7.Initially top=-1 indicates a stack is empty.</p> <p>8.Stack is full represented by the condition TOP=MAX-1(if array index starts from '0').</p> <p>9.To push an element into a stack, Top is incremented by one</p> <p>10.To POP an element from stack,top is decremented by one.</p>	<p>1.A Queue is a linerar list of elements in which the elements are added at one end and deletes the elements at another end.</p> <p>2.. In Queue the element which is inserted first is the element deleted first.</p> <p>3. Queues are called FIFO (First In First Out)list.</p> <p>4. In Queue elements are removed in the same order in which thy are inserted.</p> <p>5. Suppose the elements a,b,c,d,e are inserted in the Queue, the deletion of elements will be in the same order in which thy are inserted.</p> <p>6. In Queue there are two pointers one for insertion called "Rear" and another for deletion called "Front".</p> <p>7. Initially Rear=Front=-1 indicates a Queue is empty.</p> <p>8.Queue is full represented by the condition Rear=Max-1.</p> <p>9.To insert an element into Queue, Rear is incremented by one.</p> <p>10.To delete an element from Queue, Front is</p>

11.The conceptual view of Stack is as follows:



incremented by one.

11.The conceptual view of Queue is as follows:



UNIT-III

LINEAR LIST

INTRODUCTION

Linear Data Structures:

Linear data structures are those data structures in which data elements are accessed (read and written) in sequential fashion (one by one). Ex: Stacks, Queues, Lists, Arrays

Non Linear Data Structures:

Non Linear Data Structures are those in which data elements are not accessed in sequential fashion.

Ex: trees, graphs

Difference between Linear and Nonlinear Data Structures

Main difference between linear and nonlinear data structures lie in the way they organize data elements. In linear data structures, data elements are organized sequentially and therefore they are easy to implement in the computer's memory. In nonlinear data structures, a data element can be attached to several other data elements to represent specific relationships that exist among them. Due to this nonlinear structure, they might be difficult to be implemented in computer's linear memory compared to implementing linear data structures. Selecting one data structure type over the other should be done carefully by considering the relationship among the data elements that needs to be stored.

LINEAR LIST

A data structure is said to be linear if its elements form a sequence. A linear list is a list that displays the relationship of adjacency between elements.

A Linear list can be defined as a data object whose instances are of the form $(e_1, e_2, e_3 \dots e_n)$ where n is a finite natural number. The e_i terms are the elements of the list and n is its length. The elements may be viewed as atomic as their individual structure is not relevant to the structure of the list. When $n=0$, the list is empty. When $n>0$, e_1 is the first element and e_n the last. I.e; e_1 comes before e_2 , e_2 comes before e_3 and so on.

Some examples of the Linear List are

- An alphabetized list of students in a class
- A list of exam scores in non decreasing order
- A list of gold medal winners in the Olympics
- An alphabetized list of members of Congress

The following are the operations that performed on the Linear List

- ✓ Create a Linear List
- ✓ Destroy a Linear List
- ✓ Determine whether the list is empty
- ✓ Determine the size of the List
- ✓ Find the element with a given index
- ✓ Find the index of a given number
- ✓ Delete, erase or remove an element given its index
- ✓ Insert a new element so that it has a given index

A Linear List may be specified as an abstract Data type (ADT) in which we provide a specification of the instance as well as of the operations that are to be performed. The below abstract data type omitted specifying operations to create and destroy instance of the data type. All ADT specifications implicitly include an operation to create an empty instance and optionally, an operation to destroy an instance.

```

AbstractDataType linearList
{
    instances
        ordered finite collections of zero or more elements

    operations
        empty() : return true if the list is empty, false otherwise
        size() : return the list size (i.e., number of elements in the list)
        get(index): return the indexth element of the list
        indexOf(x): return the index of the first occurrence of x in the list,
                        return -1 if x is not in the list
        erase(index): remove/delete the indexth element, elements with higher in-
                        dex have their index reduced by 1
        insert(index, x): insert x as the indexth element, elements with index  $\geq$  index
                            have their index increased by 1
        output(): output the list elements from left to right
}

```

Abstract data type specification of a linear list

Array Representation: (Formula Based Representation)

A formula based representation uses an array to represent the instance of an object. Each position of the Array is called a Cell or Node and is large enough to hold one of the elements that make up an instance, while in other cases one array can represent several instances. Individual elements of an instance are located in the array using a mathematical formula.

Suppose one array is used for each list to be represented. We need to map the elements of a list to positions in the array used to represent it. In a formula based representation, a mathematical formula determines the location of each element. A simple mapping formulas is

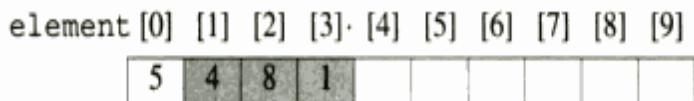
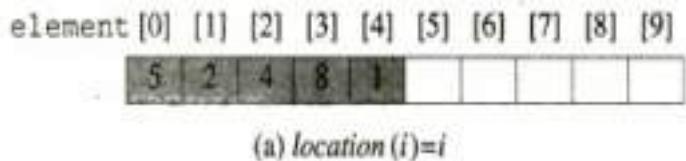
$$\boxed{\text{Location (i)} = i-1}$$

This equation states that the i^{th} element of the list is in position $i-1$ of the array. The below figure shows a five element list represented in the array element using the mapping of equation.

To completely specify the list we need to know its current length or size. For this purpose we use variable length. Length is zero when list is empty. Program gives the resulting C++ class definition. Since the data type of the list element may vary from application to application, we have defined a template class in which the user specifies the element data type T. the data members length, MaxSize and element are private members are private members, while the remaining members are public. Insert and delete have been defined to return a reference to a linear list.

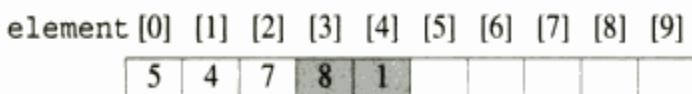
Insertion and Deletion of a Linear List:

Suppose we want to remove an element e_i from the list by moving to its right down by 1. For example, to remove an element $e_1=2$ from the list, we have to move the elements $e_2=4$, $e_3=8$, and $e_4=1$, which are to the right of e_1 , to positions 1, 2 and 3 of the array element. The below figure shows this result. The shaded elements are moved.



(a) 2 removed from element [1], listSize = 4

To insert an element so that it becomes element i of a list, must move the existing element e_i and all elements to its right one position right and then put the new element into position i of the array. For example to insert 7 as the second element of the list, we first move elements e_2 and e_3 to the right by 1 and then put 7 in to second position 2 of the array. The below figure shows this result. The shaded elements were moved.

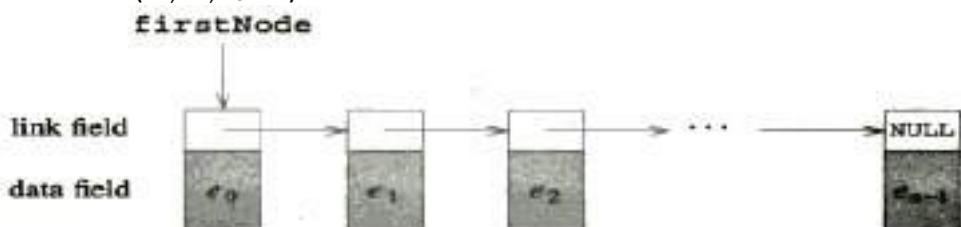


(b) 7 inserted at element [2], listSize = 5

Linked Representation And Chains

In a linked list representation each element of an instance of a data object is represented in a cell or node. The nodes however need not be component of an array and no formula is used to locate individual elements. Instead of each node keeps explicit information about the location of other relevant nodes. This explicit information about the location of another node is called Link or Pointer.

Let $L=(e_1, e_2, e_3\dots e_n)$ be a linear List. In one possible linked representation for this list, each element e_i is represented in a separate node. Each node has exactly one link field that is used to locate the next element in the linear list. So the node for e_i links to that for e_{i+1} , $0 \leq i < n-1$. The node for e_{n-1} has no need to link to and so its link field is NULL. The pointer variables first locate the first node in the representation. The below figure shows the linked representation of a List= $(e_1, e_2, e_3\dots e_n)$.

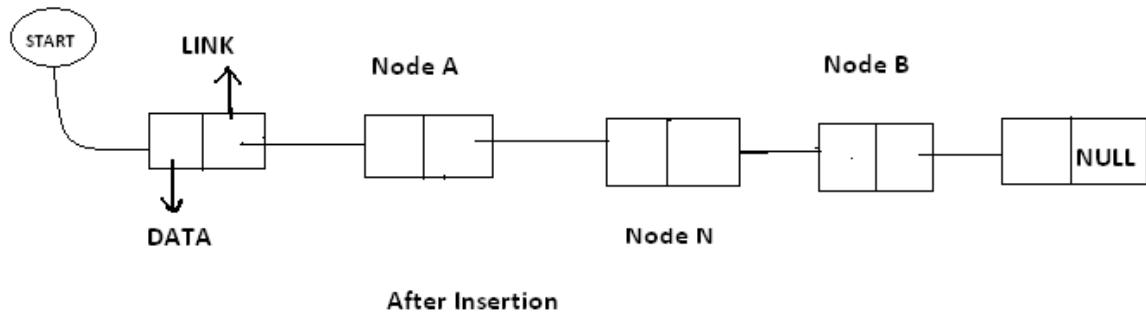
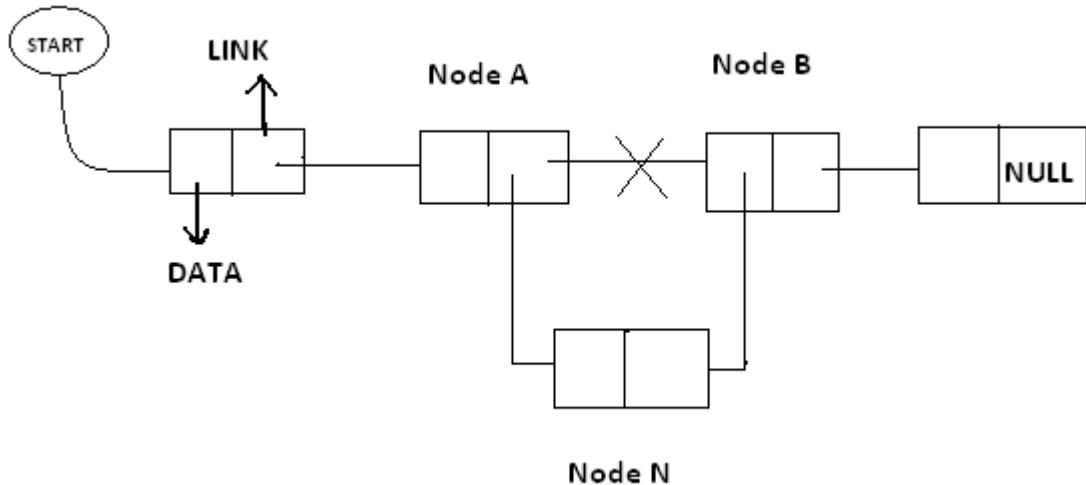


Linked representation of a linear list

Since each node in the Linked representation of the above figure has exactly one link, the structure of this figure is called a '**Single Linked List**'. The nodes are ordered from left to right with each node (other than last one) linking to the next, and the last node has a NULL link, the structure is also called a **chain**.

Insertion and Deletion of a Single Linked List:

Insertion Let the list be a Linked list with successive nodes A and B as shown in below figure. Suppose a node N is to be inserted into the list between the node A and B.

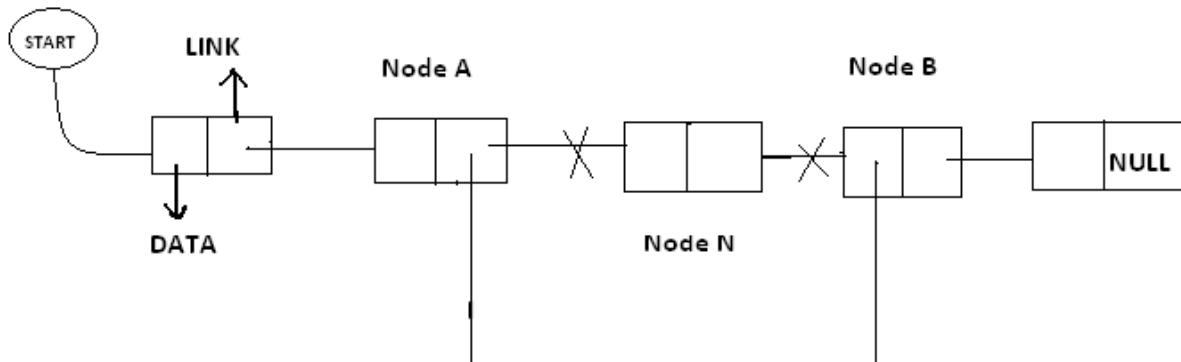


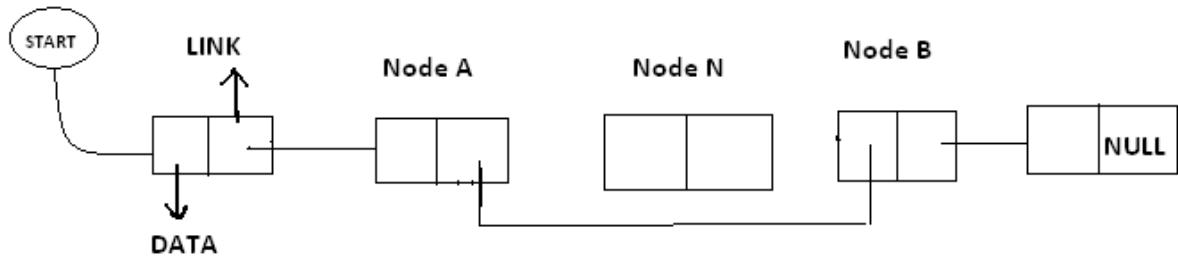
After Insertion

In the New list the Node A points to the new Node N and the new node N points to the node B to which Node A previously pointed.

Deletion:

Let list be a Linked list with node N between Nodes A and B is as shown in the following figure.



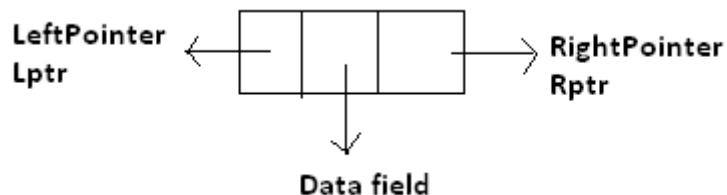


After Deletion Node N in Between Node A abd Node B

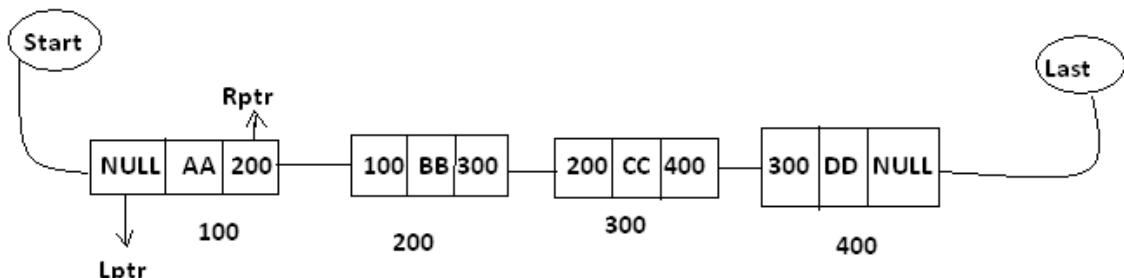
In the new list the node N is to be deleted from the Linked List. The deletion occurs as the link field in the Node A is made to point node B this excluding node N from its path.

DOUBLE LINKED LIST (Or) TWO WAY LINKED LIST

In certain applications it is very desirable that list be traversed in either forward direction or Back word direction. The property of Double Linked List implies that each node must contain two link fields instead of one. The links are used to denote the preceding and succeeding of the node. The link denoting the preceding of a node is called Left Link. The link denoting succeeding of a node is called Right Link. The list contain this type of node is called a “Double Linked List” or “Two Way List”. The Node structure in the Double Linked List is as follows:

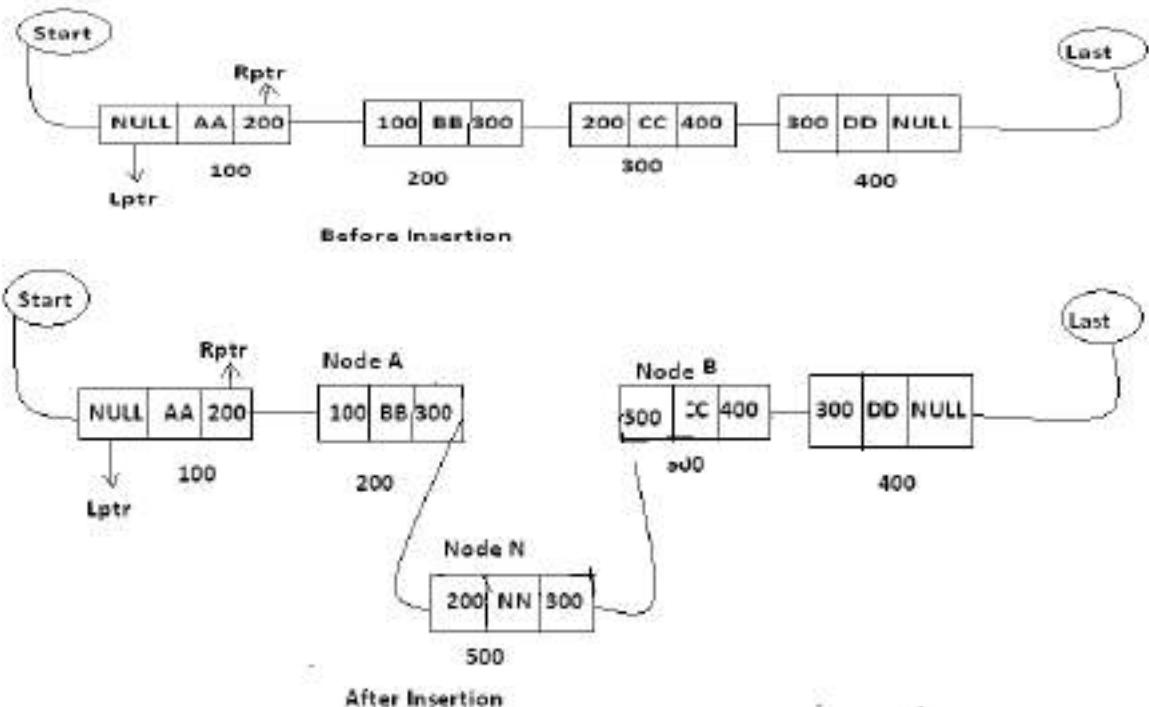


Lptr contains the address of the before node. Rptr contains the address of next node. Data Contains the Linked List is as follows.



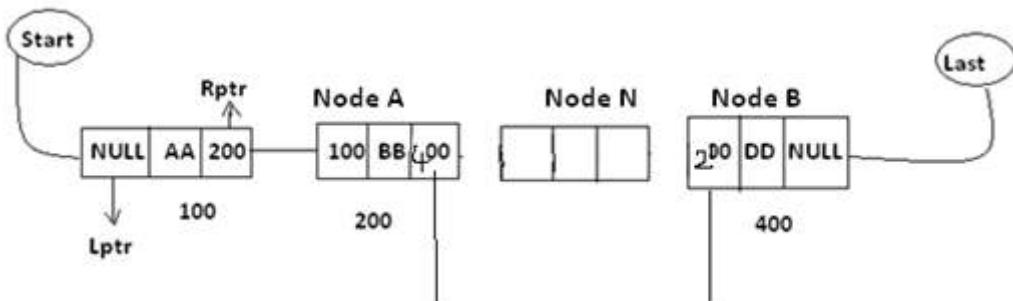
In the above diagram Last and Start are pointer variables which contains the address of last node and starting node respectively.

Insertion in to the Double Linked List: Let list be a double linked list with successive nodes A and B as shown in the following diagram. Suppose a node N is to be inserted into the list between the node s A and B this is shown in the following diagram.



As in the new list the right pointer of node A points to the new node N ,the Lptr of the node 'N' points to the node A and Rptr of node 'N' points to the node 'B' and Lpts of node B points the new node 'N'

Deletion Of Double Linked List :- Let list be a linked list contains node N between the nodes A and B as shown in the following diagram.

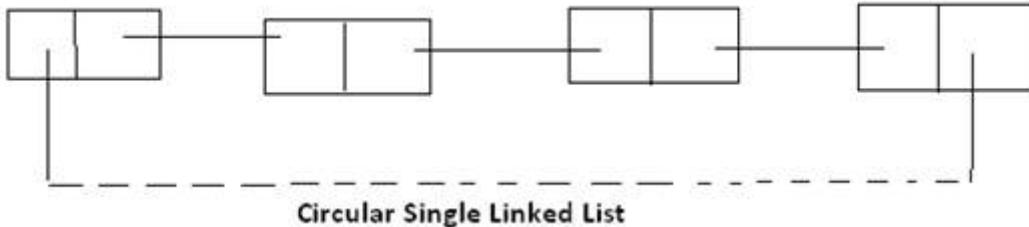


Support node N is to be deleted from the list diagram will appear as the above mention double linked list. The deletion occurs as soon as the right pointer field of node A charged, so that it points to node B and the lift point field of node B is changed. So that it pointes to node A.

Circular Linked List:- Circular Linked List is a special type of linked list in which all the nodes are linked in continuous circle. Circular list can be singly or doubly linked list. Note that, there are no Nulls in Circular Linked Lists. In these types of lists, elements can be added to the back of the list and removed from the front in constant time.

Both types of circularly-linked lists benefit from the ability to traverse the full list beginning at any given node. This avoids the necessity of storing first Node and last node, but we need a special representation for the empty list, such as a last node variable which points to some node in the list or is null if it's empty. This representation significantly simplifies adding and removing nodes with a non-empty list, but empty lists are then a special case. Circular linked lists are most useful for describing naturally circular structures, and have the advantage of being able to traverse the list starting at any point. They also allow quick access to the first and last records through a single pointer (the address of the last element)

Circular single linked list:



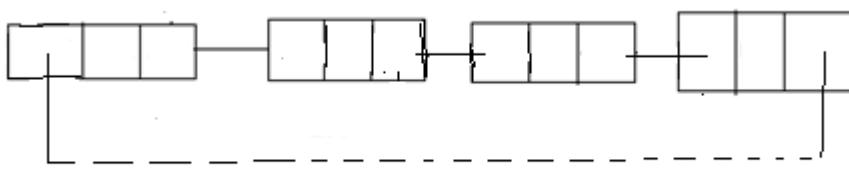
Circular linked list are one they of liner linked list. In which the link fields of last node of the list contains the address of the first node of the list instead of contains a null pointer.

Advantages:- Circular list are frequency used instead of ordinary linked list because in circular list all nodes contain a valid address. The important feature of circular list is as follows.

- (1) In a circular list every node is accessible from a given node.
- (2) Certain operations like concatenation and splitting becomes more efficient in circular list.

Disadvantages: Without some conditions in processing it is possible to get into an infinite Loop.

Circular Double Linked List :- These are one type of double linked list. In which the rpt field of the last node of the list contain the address of the first node ad the left points of the first node contains the address of the last node of the list instead of containing null pointer.



Circular Double Linked List

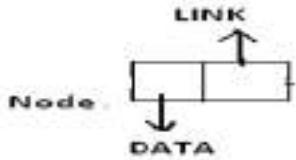
Advantages:- circular list are frequently used instead of ordinary linked list because in circular list all nodes contained a valid address. The important feature of circular list is as follows.

- (1) In a circular list every node is accessible from a given node.
- (2) Certain operations like concatenation and splitting becomes more efficient in circular list.

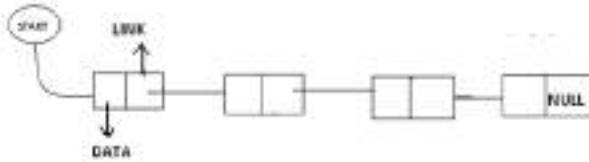
*Disadvantage:-*Without some conditions in processes it is possible to get in to an infant glad.

Difference between single linked list and double linked list?

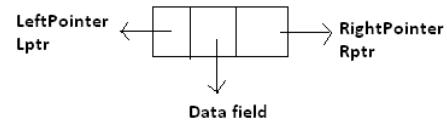
Single linked list(SLL)	Double linked list(DLL)
1.In Single Linked List the list will be traversed in only one way ie; in forward. 2. In Single Linked List the node contains one link field only. 3. Every node contains the address of next node. 4.The node structure in Single linked list is as follows:	1. In Double Linked List the list will be traversed in two way ie; either forward and backward 2. In Double Linked List the node contains two link fields. 3. Every node contains the address of next node as well as preceding node. 4.the node structure in double linked list is as follows:



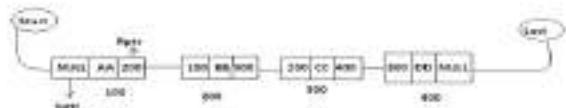
5. The conceptual view of SLL is as follows:



6. SLL are maintained in memory by using two arrays.



5. The conceptual view of DLL is as follows:



6. DLL is maintained in memory by using three arrays.

2. Difference between sequential allocation and linked allocation?

OR

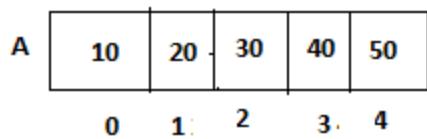
Difference between Linear List and Linked List?

OR

Difference between Arrays and Linked List?

Arrays	Linked List
<p>1. Arrays are used in the predictable storage requirement ie; exact amount of data storage required by the program can be determined.</p> <p>2. In arrays the operations such as insertion and deletion are done in an inefficient manner.</p> <p>3. The insertion and deletion are done by moving the elements either up or down.</p> <p>4. Successive elements occupy adjacent space on memory.</p> <p>5. In arrays each location contains DATA only</p> <p>6. The linear relationship between the data elements of an array is reflected by the physical relationship of data in the memory.</p> <p>7. In array declaration a block of memory space is required.</p> <p>8. There is no need of storage of pointer or links</p>	<p>1. Linked List are used in the unpredictable storage requirement ie; exact amount of data storage required by the program can't be determined.</p> <p>2. In Linked List the operations such as insertion and deletion are done more efficient manner ie; only by changing the pointer.</p> <p>3. The insertion and deletion are done by only changing the pointers.</p> <p>4. Successive elements need not occupy adjacent space.</p> <p>5. In linked list each location contains data and pointer to denote whether the next element present in the memory.</p> <p>6. The linear relationship between the data elements of a Linked List is reflected by the Linked field of the node.</p> <p>7. In Linked list there is no need of such thing.</p>

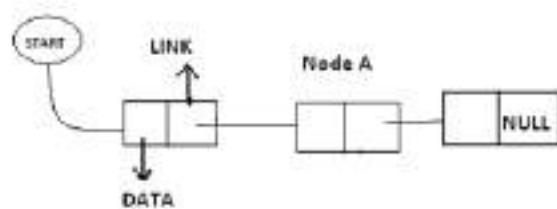
9.The Conceptual view of an Array is as follows:



10.In array there is no need for an element to specify whether the next is stored

8. In Linked list a pointer is stored along into the element.

9. The Conceptual view of Linked list is as follows:



10. There is need for an element (node) to specify whether the next node is formed.

UNIT-IV

SORTING AND SEARCHING

SORTING-INTRODUCTION

Sorting is a technique of organizing the data. It is a process of arranging the records, either in ascending or descending order i.e. bringing some order lines in the data. Sort methods are very important in Data structures.

Sorting can be performed on any one or combination of one or more attributes present in each record. It is very easy and efficient to perform searching, if data is stored in sorting order. The sorting is performed according to the key value of each record. Depending up on the makeup of key, records can be stored either numerically or alphanumerically. In numerical sorting, the records arranged in ascending or descending order according to the numeric value of the key.

Let A be a list of n elements $A_1, A_2, A_3 \dots \dots \dots A_n$ in memory. Sorting A refers to the operation of rearranging the contents of A so that they are increasing in order, that is, so that $A_1 \leq A_2 \leq A_3 \leq \dots \leq A_n$. Since A has n elements, there are $n!$ Ways that the contents can appear in A. these ways corresponding precisely to the $n!$ Permutations of $1, 2, 3, \dots, n$. accordingly each sorting algorithm must take care of these $n!$ Possibilities.

Ex: suppose an array DATA contains 8elements as follows:

DATA: 70, 30,40,10,80,20,60,50.

After sorting DATA must appear in memory as follows:

DATA: 10 20 30 40 50 60 70 80

Since DATA consists of 8 elements, there are $8!=40320$ ways that the numbers 10,20,30,40,50,60,70,80 can appear in DATA.

The factors to be considered while choosing sorting techniques are:

- Programming Time
- Execution Time
- Number of Comparisons
- Memory Utilization
- Computational Complexity

Types of Sorting Techniques:

Sorting techniques are categorized into 2 types. They are Internal Sorting and External Sorting.

Internal Sorting: Internal sorting method is used when small amount of data has to be sorted. In this method , the data to be sorted is stored in the main memory (RAM).Internal sorting method can access records randomly. EX: Bubble Sort, Insertion Sort, Selection Sort, Shell sort, Quick Sort, Radix Sort, Heap Sort etc.

External Sorting: External sorting method is used when large amount of data has to be sorted. In this method, the data to be sorted is stored in the main memory as well as in the secondary memory such as disk. External sorting methods an access records only in a sequential order. Ex: Merge Sort, Multi way Mage Sort.

Complexity of sorting Algorithms: The complexity of sorting algorithm measures the running time as a function of the number n of items to be stored. Each sorting algorithm S will be made up of the following operations, where $A_1, A_2, A_3 \dots \dots \dots A_n$ contain the items to be sorted and B is an auxiliary location.

- Comparisons, which test whether $A_i < A_j$ or test whether $A_i < B$.
- Interchanges which switch the contents of A_i and A_j or of A_i and B .
- Assignment which set $B: A_i$ and then set $A_j := B$ or $A_j := A_i$

Normally, the complexity function measures only the number of comparisons, since the number of other operations is at most a constant factor of the number of comparisons.

SELECTION SORT

In selection sort, the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array. First, find the smallest element of the array and place it on the first position. Then, find the second smallest element of the array and place it on the second position. The process continues until we get the sorted array. The array with n elements is sorted by using $n-1$ pass of selection sort algorithm.

- In 1st pass, smallest element of the array is to be found along with its index pos. then, swap $A[0]$ and $A[pos]$. Thus $A[0]$ is sorted, we now have $n - 1$ elements which are to be sorted.
- In 2nd pass, position pos of the smallest element present in the sub-array $A[n-1]$ is found. Then, swap, $A[1]$ and $A[pos]$. Thus $A[0]$ and $A[1]$ are sorted, we now left with $n-2$ unsorted elements.
- In $n-1$ th pass, position pos of the smaller element between $A[n-1]$ and $A[n-2]$ is to be found. Then, swap, $A[pos]$ and $A[n-1]$.

Therefore, by following the above explained process, the elements $A[0]$, $A[1]$, $A[2]$, ..., $A[n-1]$ are sorted.

Example: Consider the following array with 6 elements. Sort the elements of the array by using selection sort.

$$A = \{10, 2, 3, 90, 43, 56\}.$$

Pass	Pos	$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$	$A[5]$
1	1	2	10	3	90	43	56
2	2	2	3	10	90	43	56
3	3	2	3	10	90	43	56
4	4	2	3	10	43	90	56
5	5	2	3	10	43	56	90

$$\text{Sorted } A = \{2, 3, 10, 43, 56, 90\}$$

Complexity

Complexity	Best Case	Average Case	Worst Case
Time	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Space			$O(1)$

Algorithm

SELECTION SORT (ARR, N)

Step 1: Repeat Steps 2 and 3 for K = 1 to N-1

Step 2: CALL SMALLEST(A, K, N, POS)

Step 3: SWAP A[K] with
A[POS] [END OF LOOP]

Step 4: EXIT

BUBBLE SORT

Bubble Sort: This sorting technique is also known as exchange sort, which arranges values by iterating over the list several times and in each iteration the larger value gets bubble up to the end of the list. This algorithm uses multiple passes and in each pass the first and second data items are compared. if the first data item is bigger than the second, then the two items are swapped. Next the items in second and third position are compared and if the first one is larger than the second, then they are swapped, otherwise no change in their order. This process continues for each successive pair of data items until all items are sorted.

Bubble Sort Algorithm:

Step 1: Repeat Steps 2 and 3 for i=1 to 10

Step 2: Set j=1

Step 3: Repeat while j<=n

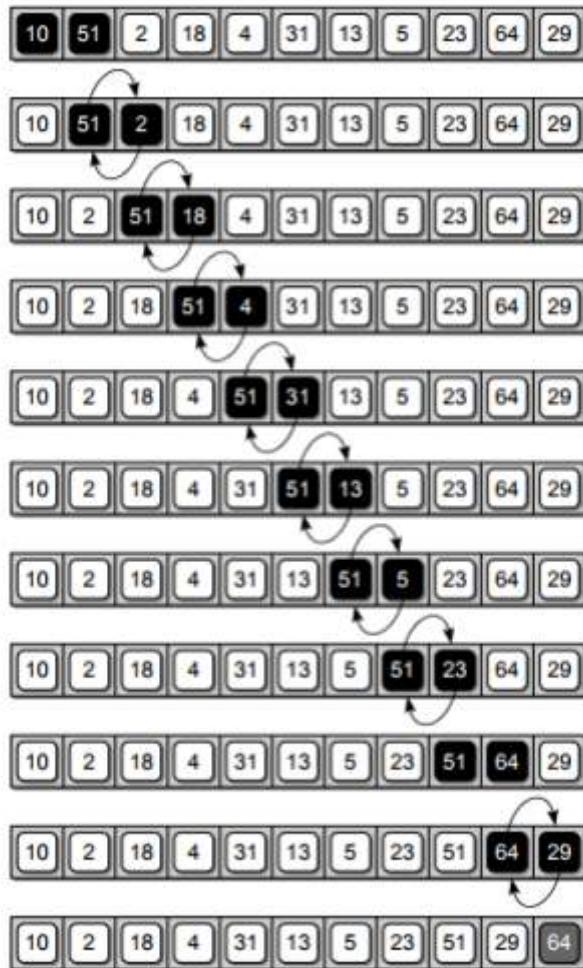
(A)

 if a[i] < a[j] Then
 interchange a[i] and a[j]
 [End of if]

(B) Set j = j+1

 [End of Inner Loop]
 [End of Step 1 Outer Loop]

Step 4: Exit



Various Passes of Bubble Sort

INSERTION SORT

Insertion sort is one of the best sorting techniques. It is twice as fast as Bubble sort. In Insertion sort the elements comparisons are as less as compared to bubble sort. In this comparison the value until all prior elements are less than the compared values is not found. This means that all the previous values are lesser than compared value. Insertion sort is good choice for small values and for nearly sorted values.

Working of Insertion sort:

The Insertion sort algorithm selects each element and inserts it at its proper position in a sub list sorted earlier. In a first pass the elements A_1 is compared with A_0 and if $A[1]$ and $A[0]$ are not sorted they are swapped.

In the second pass the element $[2]$ is compared with $A[0]$ and $A[1]$. And it is inserted at its proper position in the sorted sub list containing the elements $A[0]$ and $A[1]$. Similarly doing i^{th} iteration the element $A[i]$ is placed at its proper position in the sorted sub list, containing the elements $A[0], A[1], A[2], \dots, A[i-1]$.

To understand the insertion sort consider the unsorted Array $A = \{7, 33, 20, 11, 6\}$.

The steps to sort the values stored in the array in ascending order using Insertion sort are given below:

7	33	20	11	6
---	----	----	----	---

Step 1: The first value i.e; 7 is trivially sorted by itself.

Step 2: the second value 33 is compared with the first value 7. Since 33 is greater than 7, so no changes are made.

Step 3: Next the third element 20 is compared with its previous element (towards left).Here 20 is less than 33.but 20 is greater than 7. So it is inserted at second position. For this 33 is shifted towards right and 20 is placed at its appropriate position.

7	33	20	11	6
---	----	----	----	---

7	20	33	11	6
---	----	----	----	---

Step 4: Then the fourth element 11 is compared with its previous elements. Since 11 is less than 33 and 20 ; and greater than 7. So it is placed in between 7 and 20. For this the elements 20 and 33 are shifted one position towards the right.

7	20	33	11	6
---	----	----	----	---

7	11	20	33	6
---	----	----	----	---

Step5: Finally the last element 6 is compared with all the elements preceding it. Since it is smaller than all other elements, so they are shifted one position towards right and 6 is inserted at the first position in the array. After this pass, the Array is sorted.

7	11	20	33	6
---	----	----	----	---

6	7	11	20	33
---	---	----	----	----

Step 6: Finally the sorted Array is as follows:

6	7	11	20	33
---	---	----	----	----

ALGORITHM:

Insertion_sort(ARR,SIZE)

Step 1: Set i=1;

Step 2: while($i < SIZE$)

 Set temp=ARR[i]

 J=i-1;

 While(Temp<=ARR[j] and j>=0)

 Set ARR[j+1]=ARR[i]

 Set j=j-1

 End While

 SET ARR(j+1)=Temp;

Print ARR after i^{th} pass

Set $i=i+1$

End while

Step 3: print no.of passes $i-1$

Step 4: end

Advantages of Insertion Sort:

- It is simple sorting algorithm, in which the elements are sorted by considering one item at a time. The implementation is simple.
- It is efficient for smaller data set and for data set that has been substantially sorted before.
- It does not change the relative order of elements with equal keys
- It reduces unnecessary travels through the array
- It requires constant amount of extra memory space.

Disadvantages:-

- It is less efficient on list containing more number of elements.
- As the number of elements increases the performance of program would be slow

Complexity of Insertion Sort:

BEST CASE:-

Only one comparison is made in each pass.

The Time complexity is $O(n^2)$.

WORST CASE:- In the worst case i.e; if the list is arranged in descending order, the number of comparisons required by the insertion sort is given by:

$$\begin{aligned} 1+2+3+\dots+(n-2)+(n-1) &= (n*(n-1))/2; \\ &= (n^2-n)/2. \end{aligned}$$

The number of Comparisons are $O(n^2)$.

AVERAGE CASE:- In average case the numer of comparisons is given by

$$\frac{1}{2} + \frac{2}{2} + \frac{3}{3} + \dots + \frac{(n-2)}{2} + \frac{(n-1)}{2} = \frac{n*(n-1)}{2*2} = (n^2-n)/4 = O(n^2).$$

Program:

```
/* Program to implement insertion sort*/
#include<iostream.h>
#include<conio.h>
main()
{
int a[10],i,j,n,t;
clrscr();
cout<<"\n Enter number of elements to be Sort:";
cin>>n;
cout<<"\n Enter the elements to be Sorted:";
for(i=0;i<n;i++)
cin>>a[i];
for(i=0;i<n;i++)
{ t=a[i];
J=i;
while((j>0)&&(a[j-1]>t))
{ a[j]=a[j-1];
```

```

J=j-1;
}
a[j]=t;
}
cout<<"Array after Insertion sort:";
for(i=0;i<n;i++)
cout<"\n a[i]";
getch();
}

OUTPUT:

```

Enter number of elements to sort:5

Enter number of elements to sorted: 7 33 20 11 6

Array after Insertion sort: 6 7 11 20 33.

QUICK SORT

The Quick Sort algorithm follows the principle of divide and Conquer. It first picks up the partition element called ‘Pivot’, which divides the list into two sub lists such that all the elements in the left sub list are smaller than pivot and all the elements in the right sub list are greater than the pivot. The same process is applied on the left and right sub lists separately. This process is repeated recursively until each sub list containing more than one element.

Working of Quick Sort:

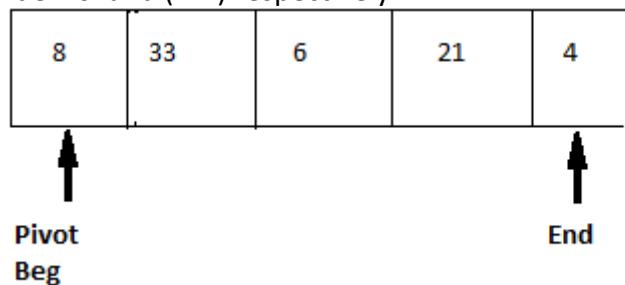
The main task in Quick Sort is to find the pivot that partitions the given list into two halves, so that the pivot is placed at its appropriate position in the array. The choice of pivot as a significant effect on the efficiency of Quick Sort algorithm. The simplest way is to choose the first element as the Pivot. However the first element is not good choice, especially if the given list is ordered or nearly ordered .For better efficiency the middle element can be chosen as Pivot.

Initially three elements Pivot, Beg and End are taken, such that both Pivot and Beg refers to 0th position and End refers to the (n-1)th position in the list. The first pass terminates when Pivot, Beg and End all refers to the same array element. This indicates that the Pivot element is placed at its final position. The elements to the left of Pivot are smaller than this element and the elements to its right are greater.

To understand the Quick Sort algorithm, consider an unsorted array as follows. The steps to sort the values stored in the array in the ascending order using Quick Sort are given below.

8	33	6	21	4
---	----	---	----	---

Step 1: Initially the index ‘0’ in the list is chosen as Pivot and the index variable Beg and End are initiated with index ‘0’ and (n-1) respectively.

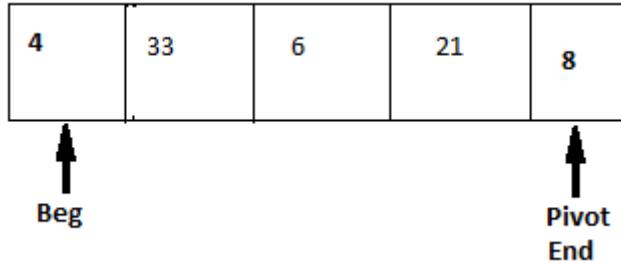


Step 2: The scanning of the element starts from the end of the list.

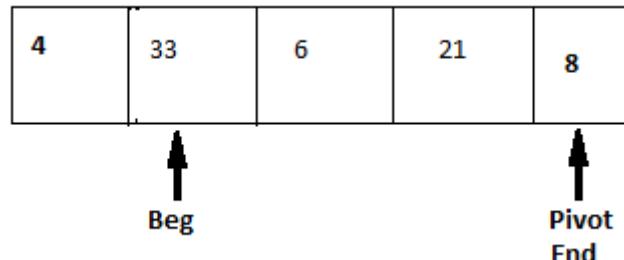
$$A[\text{Pivot}] > A[\text{End}]$$

i.e; 8 > 4

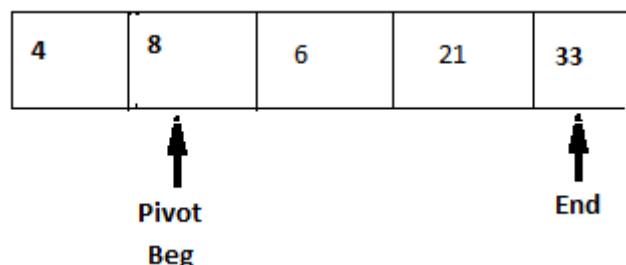
so they are swapped.



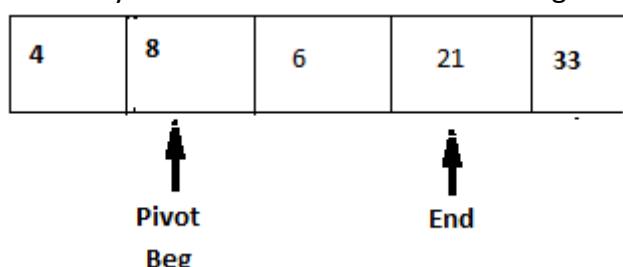
Step 3: Now the scanning of the elements starts from the beginning of the list. Since $A[\text{Pivot}] > A[\text{Beg}]$. So Beg is incremented by one and the list remains unchanged.



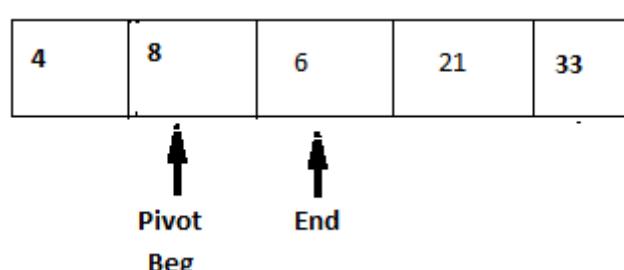
Step 4: The element $A[\text{Pivot}]$ is smaller than $A[\text{Beg}]$. So they are swapped.



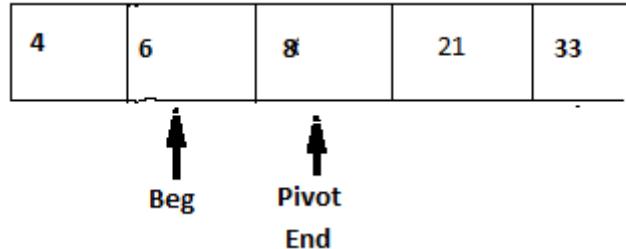
Step 5: Again the list is scanned from right to left. Since $A[\text{Pivot}]$ is smaller than $A[\text{End}]$, so the value of End is decreased by one and the list remains unchanged.



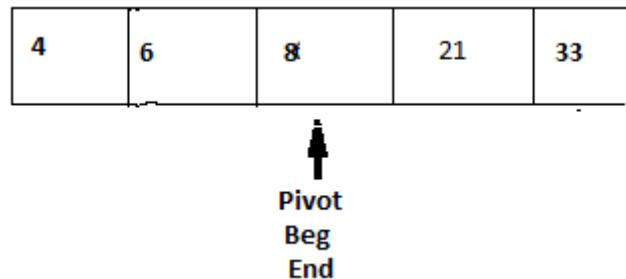
Step 6: Next the element $A[\text{Pivot}]$ is smaller than $A[\text{End}]$, the value of End is increased by one. and the list remains unchanged.



Step 7: A[Pivot]>>A[End] so they are swapped.



Step 8: Now the list is scanned from left to right. Since A[Pivot]>A[Beg], value of Beg is increased by one and the list remains unchanged.



At this point the variable Pivot, Beg, End all refers to same element, the first pass is terminated and the value 8 is placed at its appropriate position. The elements to its left are smaller than 8 and the elements to its right are greater than 8. The same process is applied on left and right sub lists.

ALGORITHM

Step 1: Select first element of array as Pivot

Step 2: Initialize i and j to Beg and End elements respectively

Step 3: Increment i until A[i]>Pivot.

Stop

Step 4: Decrement j until A[j]>Pivot

Stop

Step 5: if i<j interchange A[i] with A[j].

Step 6: Repeat steps 3,4,5 until i>j i.e: i crossed j.

Step 7: Exchange the Pivot element with element placed at j, which is correct place for Pivot.

Advantages of Quick Sort:

- This is fastest sorting technique among all.
- Its efficiency is also relatively good.
- It requires small amount of memory

Disadvantages:

- It is somewhat complex method for sorting.
- It is little hard to implement than other sorting methods
- It does not perform well in the case of small group of elements.

Complexities of Quick Sort:

Average Case: The running time complexity is $O(n \log n)$.

Worst Case : Input array is not evenly divided. So the running time complexity is $O(n^2)$.

Best Case: Input array is evenly divided. So the running time complexity is $O(n \log n)$.

MERGE SORT

The Merge Sort algorithm is based on the fact that it is easier and faster to sort two smaller arrays than one large array. It follows the principle of “Divide and Conquered”. In this sorting the list is first divided into two halves. The left and right sub lists obtained are recursively divided into two sub lists until each sub list contains not more than one element. The sub list containing only one element do not require any sorting. After that merge the two sorted sub lists to form a combined list and recursively applies the merging process till the sorted array is achieved.

Let us apply the Merge Sort to sort the following list:

13	42	36	20	63	23	12
----	----	----	----	----	----	----

Step 1: First divide the combined list into two sub lists as follows.

13	42	36	20	63	23	12
----	----	----	----	----	----	----

Step 2: Now Divide the left sub list into smaller sub list

13	42	36	20
----	----	----	----

Step 3: Similarly divide the sub lists till one element is left in the sub list.

13	42	36	20
----	----	----	----

Step 4: Next sort the elements in their appropriate positions and then combined the sub lists.

13	42	20	36
----	----	----	----

Step 5: Now these two sub lists are again merged to give the following sorted sub list of size 4.

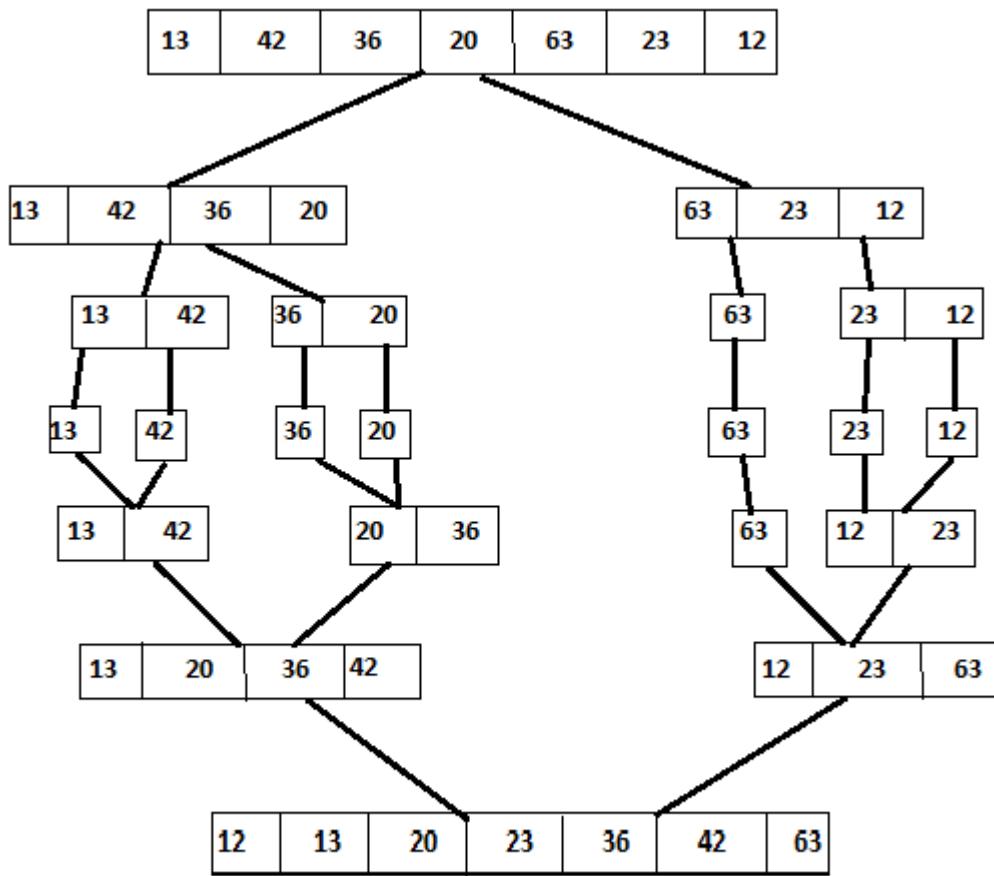
13	20	36	42
----	----	----	----

Step 6: After sorting the left half of the array, containing the same process for the right sub list also. Then the sorted array of right half of the list is as follows.

12	23	63
----	----	----

Step 7: Finally the left and right halves of the array are merged to give the sorted array as follows.

12	13	20	23	36	42	63
----	----	----	----	----	----	----



Merge Sort

Advantages:

- Merge sort is stable sort
- It is easy to understand
- It gives better performance.

Disadvantages:

- It requires extra memory space
- Copy of elements to temporary array
- It requires additional array
- It is slow process.

Complexity of Merge Sort: The merge sort algorithm passes over the entire list and requires at most $\log n$ passes and merges n elements in each pass. The total number of comparisons required by the merge sort is given by $O(n \log n)$.

External searching: When the records are stored in disk, tape, any secondary storage then that searching is known as 'External Searching'.

Internal Searching: When the records are to be searched or stored entirely within the computer memory then it is known as 'Internal Searching'.

LINEAR SEARCH

The Linear search or Sequential Search is most simple searching method. It does not expect the list to be sorted. The Key which to be searched is compared with each element of the list one by one. If a match exists, the search is terminated. If the end of the list is reached, it means that the search has failed and the Key has no matching element in the list.

Ex: consider the following Array A

23 15 18 17 42 96 103

Now let us search for 17 by Linear search. The searching starts from the first position.

Since $A[0] \neq 17$.

The search proceeds to the next position i.e; second position $A[1] \neq 17$.

The above process continuous until the search element is found such as $A[3]=17$.

Here the searching element is found in the position 4.

Algorithm: **LINEAR(DATA, N, ITEM, LOC)**

Here DATA is a linear Array with N elements. And ITEM is a given item of information. This algorithm finds the location LOC of an ITEM in DATA. LOC=-1 if the search is unsuccessful.

Step 1: Set $DATA[N+1]=ITEM$

Step 2: Set $LOC=1$

Step 3: Repeat while ($DATA[LOC] \neq ITEM$)

 Set $LOC=LOC+1$

Step 4: if $LOC=N+1$ then

 Set $LOC= -1$.

Step 5: Exit

Advantages:

- It is simplest known technique.
- The elements in the list can be in any order.

Disadvantages:

This method is in efficient when large numbers of elements are present in list because time taken for searching is more.

Complexity of Linear Search: The worst and average case complexity of Linear search is $O(n)$, where 'n' is the total number of elements present in the list.

BINARY SEARCH

Suppose DATA is an array which is stored in increasing order then there is an extremely efficient searching algorithm called "Binary Search". Binary Search can be used to find the location of the given ITEM of information in DATA.

Working of Binary Search Algorithm:

During each stage of algorithm search for ITEM is reduced to a segment of elements of $DATA[BEG], DATA[BEG+1], DATA[BEG+2], \dots, DATA[END]$.

Here BEG and END denotes beginning and ending locations of the segment under considerations. The algorithm compares ITEM with middle element $DATA[MID]$ of a segment, where $MID=[BEG+END]/2$. If $DATA[MID]=ITEM$ then the search is successful. and we said that $LOC=MID$. Otherwise a new segment of data is obtained as follows:

- i. If $ITEM < DATA[MID]$ then item can appear only in the left half of the segment.
 $DATA[BEG], DATA[BEG+1], DATA[BEG+2]$
So we reset $END=MID-1$. And begin the search again.

- ii. If ITEM>DATA[MID] then ITEM can appear only in right half of the segment i.e. DATA[MID+1], DATA[MID+2],.....DATA[END].

So we reset BEG=MID+1. And begin the search again.

Initially we begin with the entire array DATA i.e. we begin with BEG=1 and END=n

Or

BEG=lb(Lower Bound)

END=ub(Upper Bound)

If ITEM is not in DATA then eventually we obtained END<BEG. This condition signals that the searching is Unsuccessful.

The precondition for using Binary Search is that the list must be sorted one.

Ex: consider a list of sorted elements stored in an Array A is

2	12	30	35	46	53	60	70	75
↑ lb=1							↑ ub=9	

Let the key element which is to be searched is 35.

Key=35

The number of elements in the list n=9.

Step 1: MID= [lb+ub]/2

$$\begin{aligned} &= (1+9)/2 \\ &= 5 \end{aligned}$$

2	12	30	35	46	53	60	70	75
↑ lb=1				↑ MID				↑ ub=9

Key<A[MID]

i.e. 35<46.

So search continues at lower half of the array.

Ub=MID-1

$$\begin{aligned} &= 5-1 \\ &= 4. \end{aligned}$$

Step 2: MID= [lb+ub]/2

$$\begin{aligned} &= (1+4)/2 \\ &= 2. \end{aligned}$$

2	12	30	35	46	53	60	70	75
↑ lb=1	↑ MID		↑ ub=4					

Key>A[MID]

i.e. 35>12.

So search continues at Upper Half of the array.

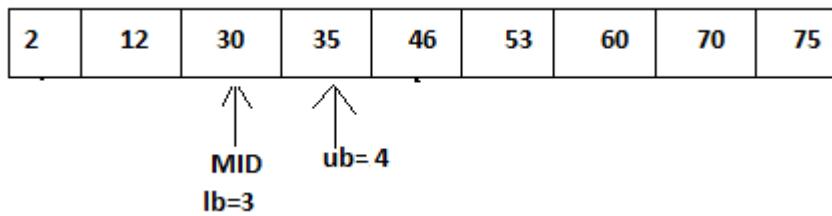
Lb=MID+1

$$\begin{aligned} &= 2+1 \\ &= 3. \end{aligned}$$

Step 3: $MID = [lb+ub]/2$

$$= (3+4)/2$$

$$= 3.$$



Key > A[MID]

i.e. 35 > 30.

So search continues at Upper Half of the array.

Lb = MID + 1

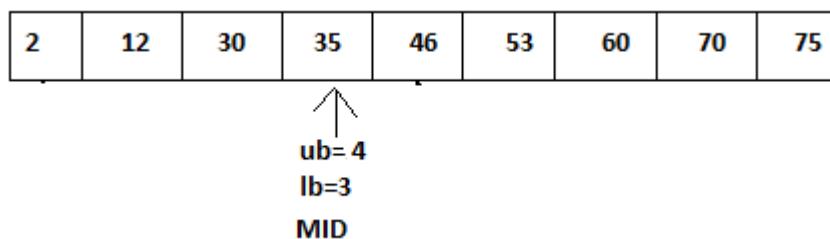
$$= 3 + 1$$

$$= 4.$$

Step 4: $MID = [lb+ub]/2$

$$= (4+4)/2$$

$$= 4.$$



ALGORITHM:

BINARY SEARCH[A,N,KEY]

Step 1: begin

Step 2: [Initialization]

 Lb=1; ub=n;

Step 3: [Search for the ITEM]

 Repeat through step 4, while Lower bound is less than Upper Bound.

Step 4: [Obtain the index of middle value]

$MID = (lb+ub)/2$

Step 5: [Compare to search for ITEM]

 If Key < A[MID] then

 Ub=MID-1

 Otherwise if Key > A[MID] then

 Lb=MID+1

 Otherwise write "Match Found"

 Return Middle.

Step 6: [Unsuccessful Search]

 write "Match Not Found"

Step 7: Stop.

Advantages: When the number of elements in the list is large, Binary Search executed faster than linear search. Hence this method is efficient when number of elements is large.

Disadvantages: To implement Binary Search method the elements in the list must be in sorted order, otherwise it fails.

Define sorting? What is the difference between internal and external sorting methods?

Ans:- Sorting is a technique of organizing data. It is a process of arranging the elements either may be ascending or descending order, ie; bringing some order lines with data.

Internal sorting	External sorting
1. Internal Sorting takes place in the main memory of a computer.	1. External sorting is done with additional external memory like magnetic tape or hard disk
2. The internal sorting methods are applied to small collection of data.	2. The External sorting methods are applied only when the number of data elements to be sorted is too large.
3. Internal sorting takes small input	3. External sorting can take as much as large input.
4. It means that, the entire collection of data to be sorted is small enough that the sorting can take place within main memory.	4. External sorting typically uses a sort-merge strategy, and requires auxiliary storage.
5. For sorting larger datasets, it may be necessary to hold only a chunk of data in memory at a time, since it won't all fit.	5. In the sorting phase, chunks of data small enough to fit in main memory are read, sorted, and written out to a temporary file.
6. Example of Internal Sorting algorithms are :- Bubble Sort, Insertion Sort, Quick Sort, Heap Sort, Binary Sort, Radix Sort, Selection sort.	6. Example of External sorting algorithms are: - Merge Sort, Two-way merge sort.
7. Internal sorting does not make use of extra resources.	7. External sorting make use of extra resources.

Justify the fact that the efficiency of Quick sort is $O(n \log n)$ under best case?

Ans:- Best Case:-

The best case in quick sort arises when the pivot element divides the lists into two exactly equal sub lists. Accordingly

- i) Reducing the initial list places '1' element and produces two equal sub lists.
- ii) Reducing the two sub lists places '2' elements and produces four equal sub lists and so on.

Observe that the reduction step in the k^{th} level finalizes the location of $2^{(k-1)}$ elements, hence there will be approximately $\log n$ levels of reduction. Further, each level uses at most ' n ' comparisons, So $f(n) = O(n \log n)$. Hence the efficiency of quick sort algorithm is $O(n \log n)$ under the best case.

Mathematical Proof:- Hence from the above, the recurrence relation for quick sort under best case is given by

$$T(n) = 2T(n/2) + kn$$

By using substitution method , we get

$$\begin{aligned} T(n) &= 2T(n/2) + kn \\ &= 2\{2T(n/4) + k.n/2\} + kn \\ &= 4T(n/4) + 2kn \end{aligned}$$

In general

$$T(n) = 2^k T(n/2^k) + a kn // \text{after } k \text{ substitutions}$$

The above recurrence relation continues until $n=2^k$, $k=\log n$

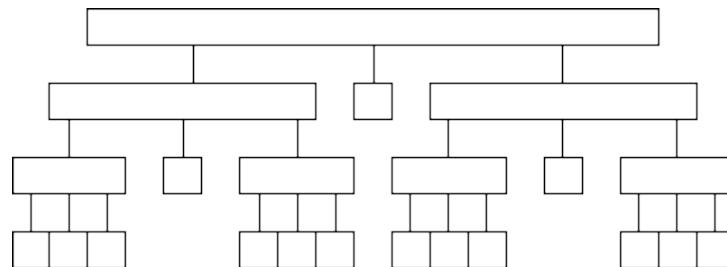
By substituting the above values , we get

$$T(n) \text{ is } O(n \log n)$$

Quick sort, or partition-exchange sort, is a sorting algorithm that, on average, makes $O(n \log n)$ comparisons to sort n items. In the worst case, it makes $O(n^2)$ comparisons, though this behavior is rare. Quick sort is often faster in practice than other $O(n \log n)$ algorithms. Additionally, quick sort's sequential and localized memory references work well with a cache. Quick sort is a comparison sort and, in efficient implementations, is not a stable sort. Quick sort can be implemented with an in-place partitioning algorithm, so the entire sort can be done with only $O(\log n)$ additional space used by the stack during the recursion. Since each element ultimately ends up in the correct position, the algorithm correctly sorts. But how long does it take.

The best case for divide-and-conquer algorithms comes when we split the input as evenly as possible. Thus in the best case, each sub problem is of size $n/2$.The partition step on each sub problem is linear in its size. Thus the total effort in partitioning the 2^k problems of size $n/2^k$ is $O(n)$.

The recursion tree for the best case looks like this:



The total partitioning on each level is $O(n)$, and it takes $\log n$ levels of perfect partitions to get to single element sub problems. When we are down to single elements, the problems are sorted. Thus the total time in the best case is $O(n \log n)$.

UNIT-V

TREES AND BINARY TREES

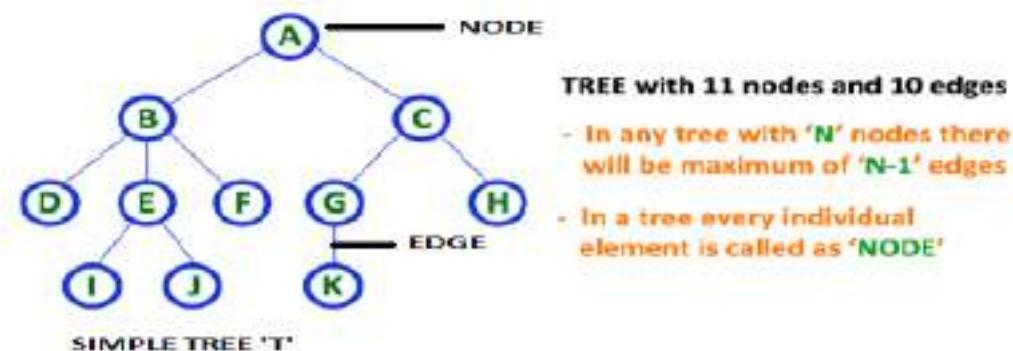
TREES

INTRODUCTION

In linear data structure data is organized in sequential order and in non-linear data structure data is organized in random order. A tree is a very popular non-linear data structure used in a wide range of applications. Tree is a non-linear data structure which organizes data in hierarchical structure and this is a recursive definition.

DEFINITION OF TREE:

Tree is collection of nodes (or) vertices and their edges (or) links. In tree data structure, every individual element is called as **Node**. Node in a tree data structure stores the actual data of that particular element and link to next element in hierarchical structure.

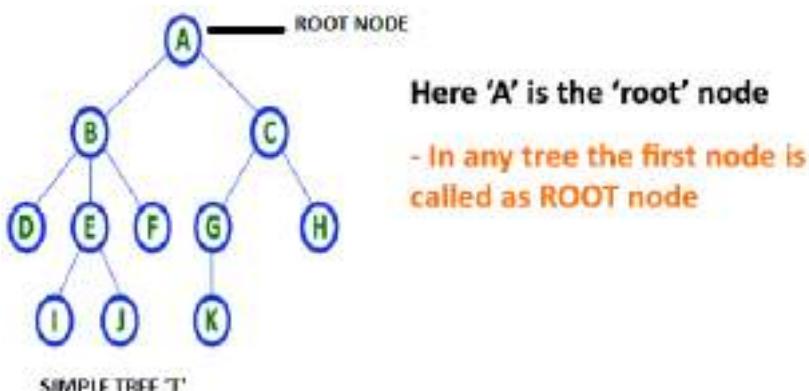


Note: 1. In a **Tree**, if we have N number of nodes then we can have a maximum of $N-1$ number of links or edges.

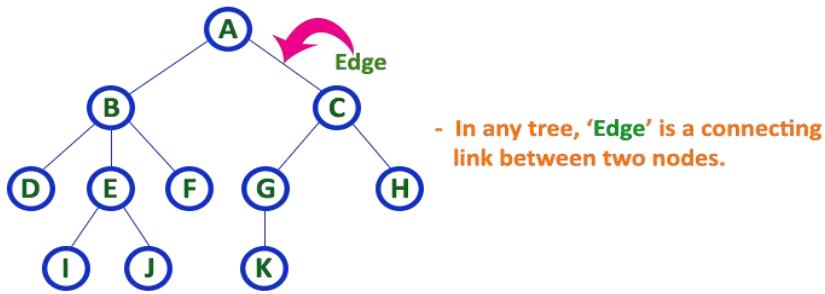
2. **Tree** has no cycles.

TREE TERMINOLOGIES:

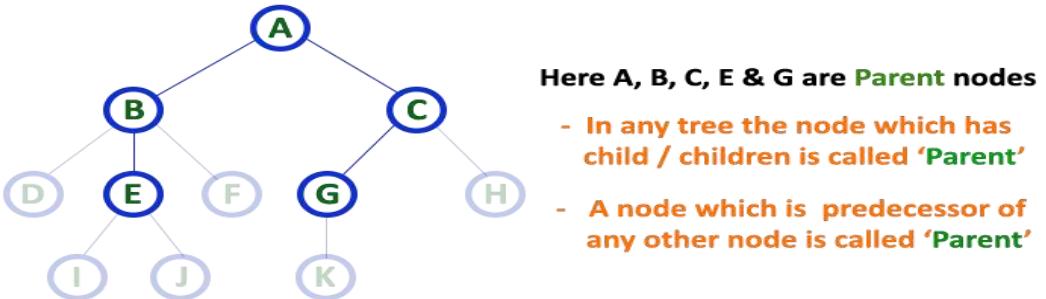
1. **Root Node:** In a **Tree** data structure, the first node is called as **Root Node**. Every tree must have a root node. We can say that the root node is the origin of the tree data structure. In any tree, there must be only one root node. We never have multiple root nodes in a tree.



2. **Edge:** In a **Tree**, the connecting link between any two nodes is called as **EDGE**. In a tree with ' N ' number of nodes there will be a maximum of ' $N-1$ ' number of edges.

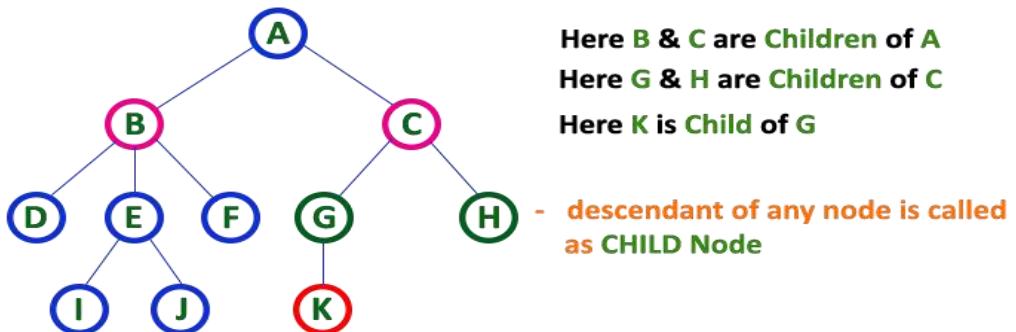


3. Parent Node: In a **Tree**, the node which is a predecessor of any node is called as **PARENT NODE**. In simple words, the node which has a branch from it to any other node is called a parent node. Parent node can also be defined as "The node which has child / children".

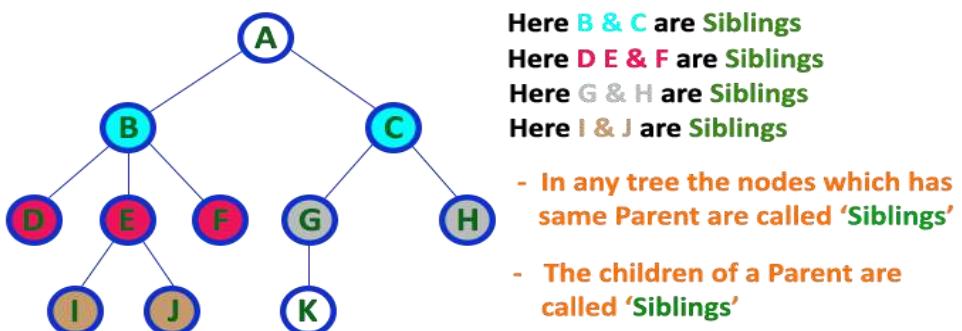


Here, A is parent of B&C. B is the parent of D,E&F and so on...

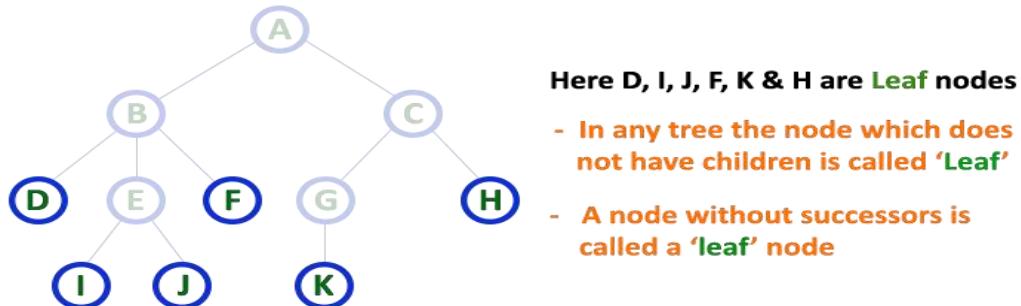
4. Child Node: In a **Tree** data structure, the node which is descendant of any node is called as **CHILD Node**. In simple words, the node which has a link from its parent node is called as child node. In a tree, any parent node can have any number of child nodes. In a tree, all the nodes except root are child nodes.



5. Siblings: In a **Tree** data structure, nodes which belong to same Parent are called as **SIBLINGS**. In simple words, the nodes with the same parent are called Sibling nodes.

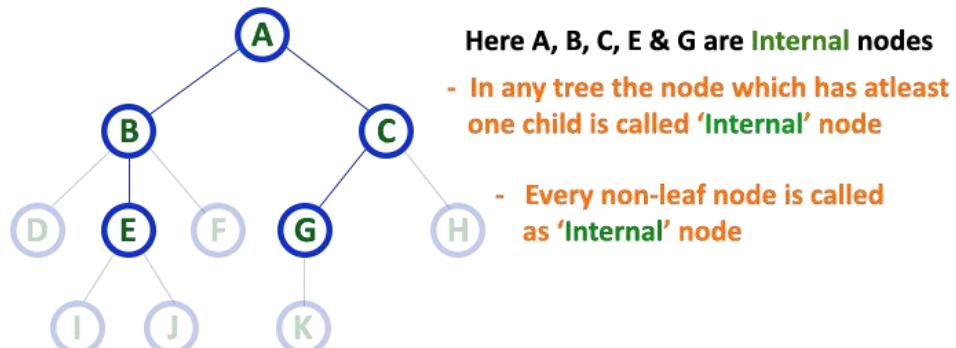


6. Leaf Node: In a **Tree** data structure, the node which does not have a child is called as **LEAF Node**. In simple words, a leaf is a node with no child. In a tree data structure, the leaf nodes are also called as External Nodes. External node is also a node with no child. In a tree, leaf node is also called as 'Terminal' node.

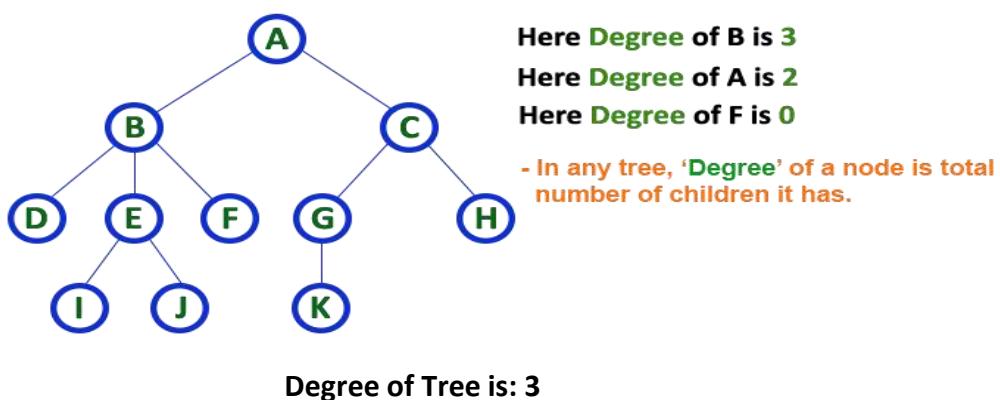


7. Internal Nodes: In a **Tree** data structure, the node which has atleast one child is called as **INTERNAL Node**. In simple words, an internal node is a node with atleast one child.

In a **Tree** data structure, nodes other than leaf nodes are called as Internal Nodes. The root node is also said to be Internal Node if the tree has more than one node. Internal nodes are also called as 'Non-Terminal' nodes.

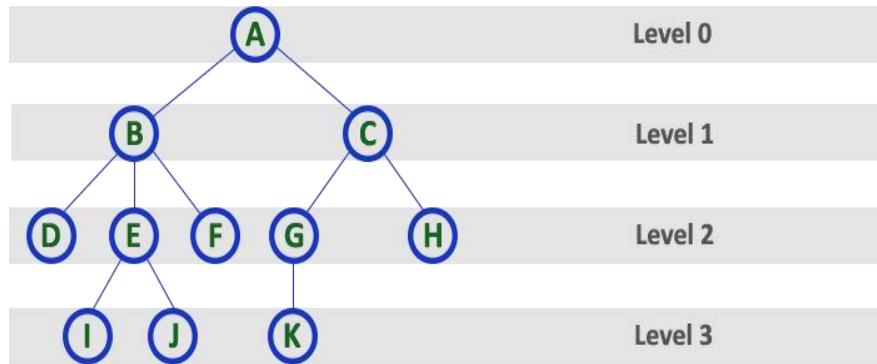


8. Degree: In a **Tree** data structure, the total number of children of a node is called as **DEGREE** of that Node. In simple words, the Degree of a node is total number of children it has. The highest degree of a node among all the nodes in a tree is called as '**Degree of Tree**'

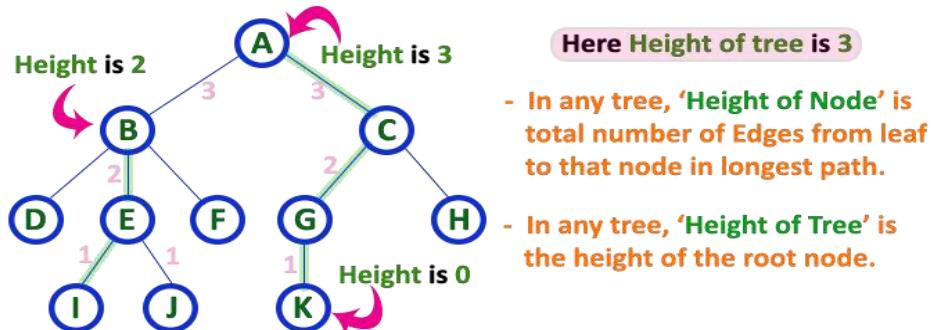


9. Level: In a **Tree** data structure, the root node is said to be at Level 0 and the children of root node are at Level 1 and the children of the nodes which are at Level 1 will be at Level 2

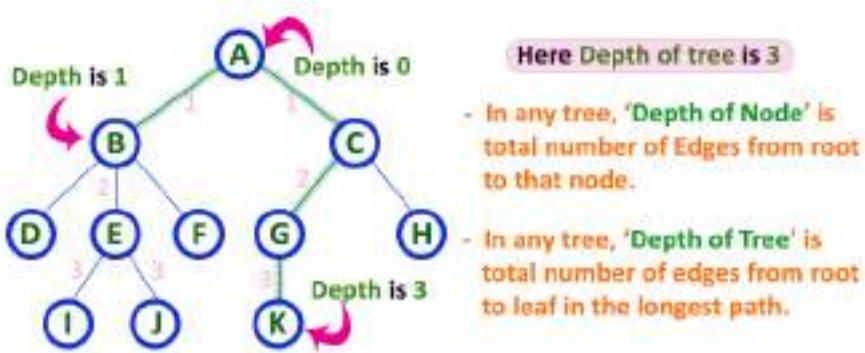
and so on... In simple words, in a tree each step from top to bottom is called as a Level and the Level count starts with '0' and incremented by one at each level (Step).



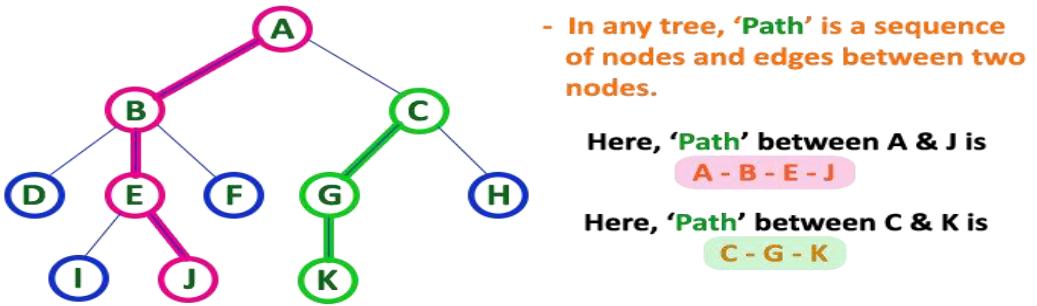
10. Height: In a **Tree** data structure, the total number of edges from leaf node to a particular node in the longest path is called as **HEIGHT** of that Node. In a tree, height of the root node is said to be height of the tree. In a tree, height of all leaf nodes is '0'.



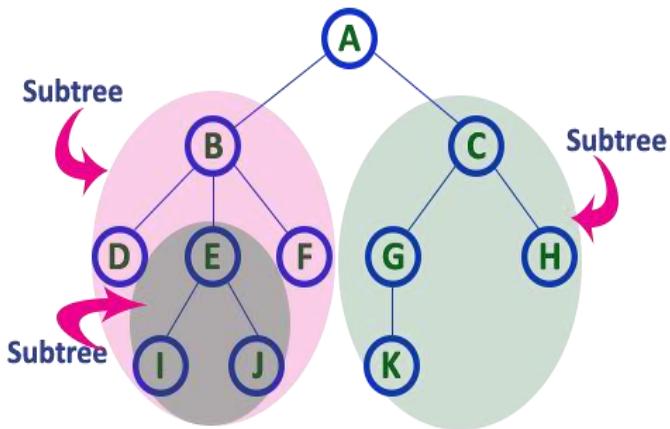
11. Depth: In a **Tree** data structure, the total number of edges from root node to a particular node is called as **DEPTH** of that Node. In a tree, the total number of edges from root node to a leaf node in the longest path is said to be **Depth of the tree**. In simple words, the highest depth of any leaf node in a tree is said to be depth of that tree. In a tree, **depth of the root node is '0'**.



12. Path: In a **Tree** data structure, the sequence of Nodes and Edges from one node to another node is called as **PATH** between that two Nodes. **Length of a Path** is total number of nodes in that path. In below example **the path A - B - E - J has length 4**.



13. Sub Tree: In a **Tree** data structure, each child from a node forms a subtree recursively. Every child node will form a subtree on its parent node.

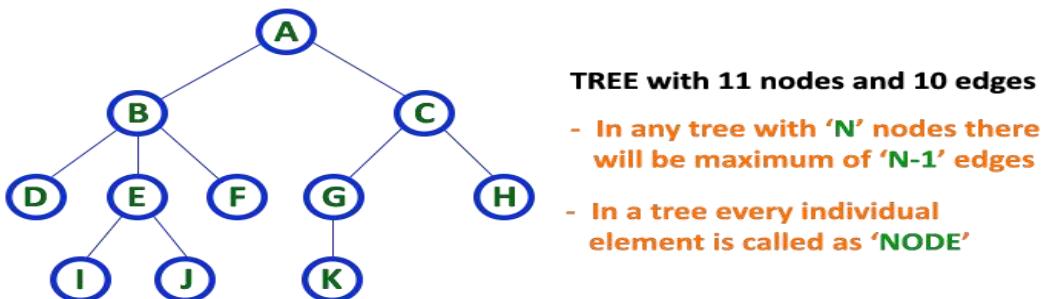


TREE REPRESENTATIONS:

A tree data structure can be represented in two methods. Those methods are as follows...

1. List Representation
2. Left Child - Right Sibling Representation

Consider the following tree...

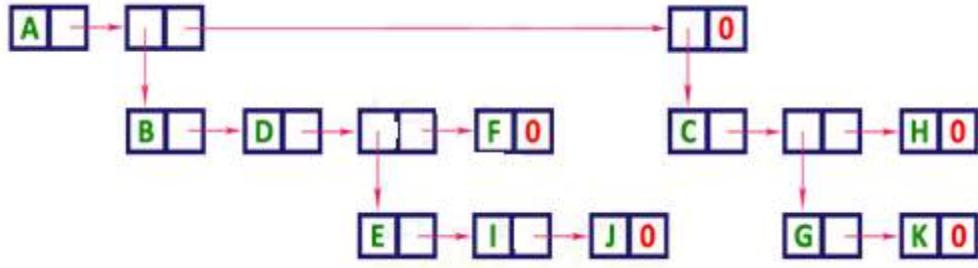


1. List Representation

In this representation, we use two types of nodes one for representing the node with data called 'data node' and another for representing only references called 'reference node'. We start with a 'data node' from the root node in the tree. Then it is linked to an internal node

through a 'reference node' which is further linked to any other node directly. This process repeats for all the nodes in the tree.

The above example tree can be represented using List representation as follows...



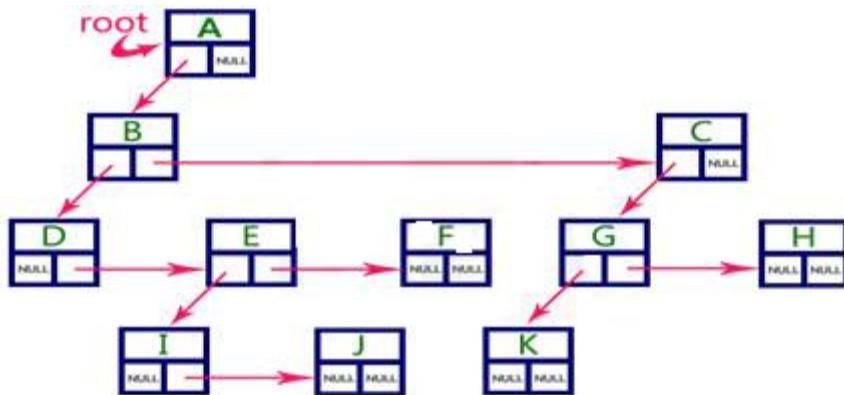
2. Left Child - Right Sibling Representation

In this representation, we use a list with one type of node which consists of three fields namely Data field, Left child reference field and Right sibling reference field. Data field stores the actual value of a node, left reference field stores the address of the left child and right reference field stores the address of the right sibling node. Graphical representation of that node is as follows...

Data	
Left Child	Right Sibling

In this representation, every node's data field stores the actual value of that node. If that node has left a child, then left reference field stores the address of that left child node otherwise stores NULL. If that node has the right sibling, then right reference field stores the address of right sibling node otherwise stores NULL.

The above example tree can be represented using Left Child - Right Sibling representation as follows...



BINARY TREE:

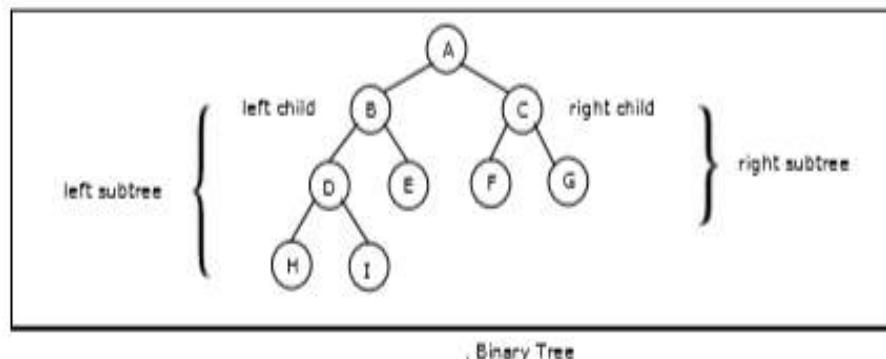
In a normal tree, every node can have any number of children. A binary tree is a special type of tree data structure in which every node can have a **maximum of 2 children**. One is known as a left child and the other is known as right child.

A tree in which every node can have a maximum of two children is called **Binary Tree**.

In a binary tree, every node can have either 0 children or 1 child or 2 children but not more than 2 children.

In general, tree nodes can have any number of children. In a binary tree, each node can have at most two children. A binary tree is either empty or consists of a node called the root together with two binary trees called the left subtree and the right subtree. A tree with no nodes is called as a null tree

Example:



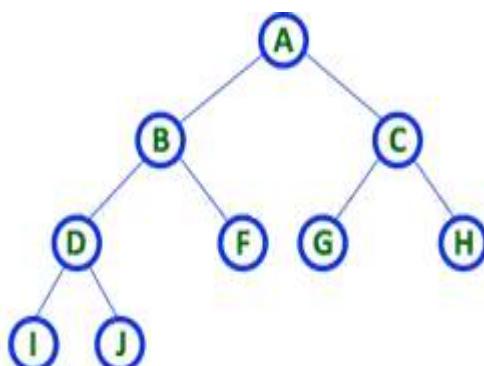
TYPES OF BINARY TREE:

1. Strictly Binary Tree:

In a binary tree, every node can have a maximum of two children. But in strictly binary tree, every node should have exactly two children or none. That means every internal node must have exactly two children. A strictly Binary Tree can be defined as follows...

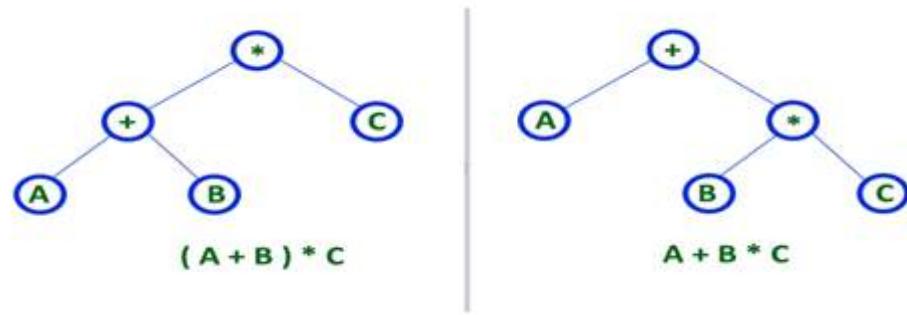
A binary tree in which every node has either two or zero number of children is called Strictly Binary Tree

Strictly binary tree is also called as **Full Binary Tree** or **Proper Binary Tree** or **2-Tree**.



Strictly binary tree data structure is used to represent mathematical expressions.

Example



2. Complete Binary Tree:

In a binary tree, every node can have a maximum of two children. But in strictly binary tree, every node should have exactly two children or none and in complete binary tree all the nodes must have exactly two children and at every level of complete binary tree there must be 2^{level} number of nodes. For example at level 2 there must be $2^2 = 4$ nodes and at level 3 there must be $2^3 = 8$ nodes.

A binary tree in which every internal node has exactly two children and all leaf nodes are at same level is called Complete Binary Tree.

Complete binary tree is also called as **Perfect Binary Tree**.

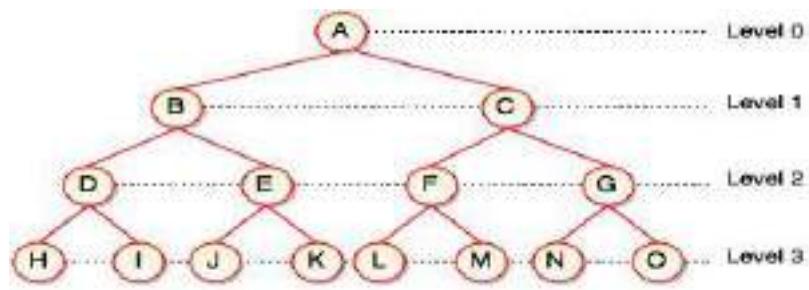
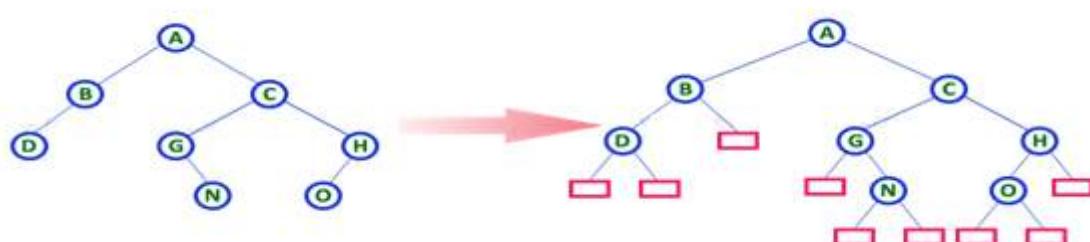


Fig. Complete Binary Tree

3. Extended Binary Tree:

A binary tree can be converted into Full Binary tree by adding dummy nodes to existing nodes wherever required.

The full binary tree obtained by adding dummy nodes to a binary tree is called as Extended Binary Tree.



In above figure, a normal binary tree is converted into full binary tree by adding dummy nodes.

4. Skewed Binary Tree:

If a tree which is dominated by left child node or right child node, is said to be a **Skewed Binary Tree**.

In a **skewed binary tree**, all nodes except one have only one child node. The remaining node has no child.

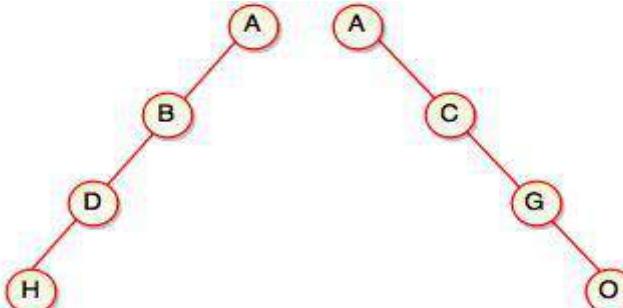


Fig. Left Skewed
Binary Tree

Fig. Right Skewed
Binary Tree

In a left skewed tree, most of the nodes have the left child without corresponding right child.

In a right skewed tree, most of the nodes have the right child without corresponding left child.

Properties of binary trees:

Some of the important properties of a binary tree are as follows:

1. If h = height of a binary tree, then

- Maximum number of leaves = 2^h
- Maximum number of nodes = $2h + 1 - 1$

2. If a binary tree contains m nodes at level l , it contains at most $2m$ nodes at level $l + 1$.

3. Since a binary tree can contain at most one node at level 0 (the root), it can contain at most 2^l node at level l .

4. The total number of edges in a full binary tree with n node is $n - 1$

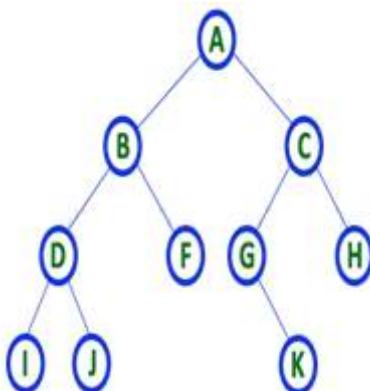
BINARY TREE REPRESENTATIONS:

A binary tree data structure is represented using two methods. Those methods are as follows...

1. Array Representation

2. Linked List Representation

Consider the following binary tree...



1. Array Representation of Binary Tree

In array representation of a binary tree, we use one-dimensional array (1-D Array) to represent a binary tree.

Consider the above example of a binary tree and it is represented as follows...



To represent a binary tree of depth ' n ' using array representation, we need one dimensional array with a maximum size of 2^{n+1} .

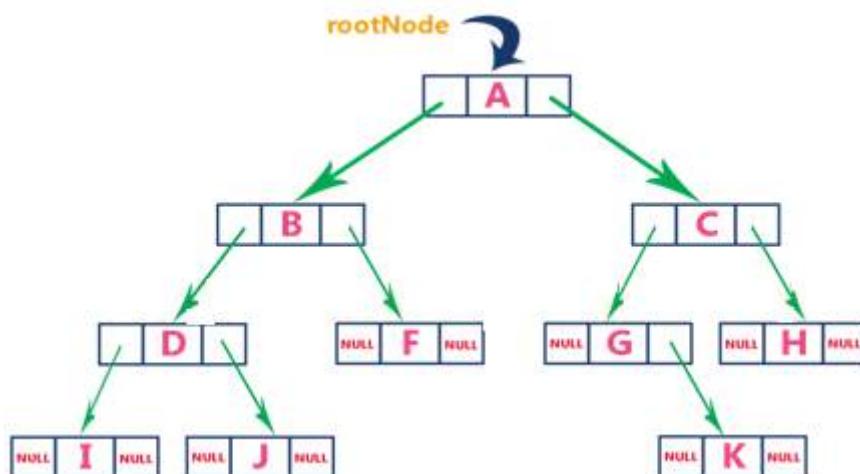
2. Linked List Representation of Binary Tree

We use a double linked list to represent a binary tree. In a double linked list, every node consists of three fields. First field for storing left child address, second for storing actual data and third for the right child address.

In this linked list representation, a node has the following structure...



The above example of the binary tree represented using Linked list representation is shown as follows...



BINARY TREE TRAVERSALS:

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, binary trees can be traversed in different ways. Following are the generally used ways for traversing binary trees.

When we wanted to display a binary tree, we need to follow some order in which all the nodes of that binary tree must be displayed. In any binary tree, displaying order of nodes depends on the traversal method.

Displaying (or) visiting order of nodes in a binary tree is called as Binary Tree Traversal.

There are three types of binary tree traversals.

1. In - Order Traversal
2. Pre - Order Traversal
3. Post - Order Traversal

1. In - Order Traversal (left Child - root - right Child):

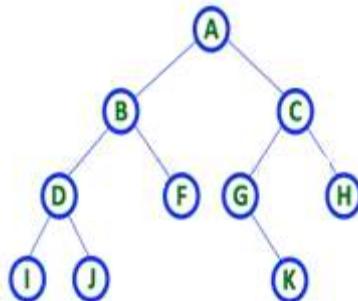
In In-Order traversal, the root node is visited between the left child and right child. In this traversal, the left child node is visited first, then the root node is visited and later we go for visiting the right child node. This in-order traversal is applicable for every root node of all sub trees in the tree. This is performed recursively for all nodes in the tree.

Algorithm:

Step-1: Visit the left subtree, using inorder.

Step-2: Visit the root.

Step-3: Visit the right subtree, using inorder.



In the above example of a binary tree, first we try to visit left child of root node 'A', but A's left child 'B' is a root node for left subtree. so we try to visit its (B's) left child 'D' and again D is a root for subtree with nodes D, I and J. So we try to visit its left child 'I' and it is the leftmost child. So first we visit 'I' then go for its root node 'D' and later we visit D's right child 'J'. With this we have completed the left part of node B. Then visit 'B' and next B's right child 'F' is visited. With this we have completed left part of node A. Then visit root node 'A'. With this we have completed left and root parts of node A. Then we go for the right part of the node A. In right of A again there is a subtree with root C. So go for left child of C and again it is a subtree with root G. But G does not have left part so we visit 'G' and then visit G's right child K. With this we have completed the left part of node C. Then visit root node 'C' and next visit C's right child 'H' which is the rightmost child in the tree. So we stop the process.

That means here we have visited in the order of I - D - J - B - F - A - G - K - C - H using In-Order Traversal.

2. Pre - Order Traversal (root - leftChild - rightChild):

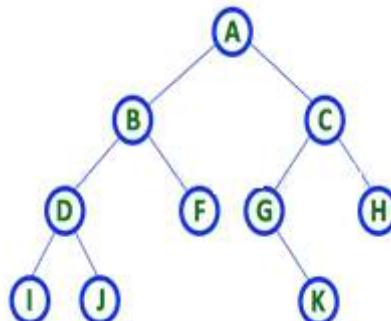
In Pre-Order traversal, the root node is visited before the left child and right child nodes. In this traversal, the root node is visited first, then its left child and later its right child. This pre-order traversal is applicable for every root node of all subtrees in the tree. Preorder search is also called backtracking.

Algorithm:

Step-1: Visit the root.

Step-2: Visit the left subtree, using preorder.

Step-3: Visit the right subtree, using preorder.



In the above example of binary tree, first we visit root node 'A' then visit its left child 'B' which is a root for D and F. So we visit B's left child 'D' and again D is a root for I and J. So we visit D's left child 'I' which is the leftmost child. So next we go for visiting D's right child 'J'. With this we have completed root, left and right parts of node D and root, left parts of node B. Next visit B's right child 'F'. With this we have completed root and left parts of node A. So we go for A's right child 'C' which is a root node for G and H. After visiting C, we go for its left child 'G' which is a root for node K. So next we visit left of G, but it does not have left child so we go for G's right child 'K'. With this, we have completed node C's root and left parts. Next visit C's right child 'H' which is the rightmost child in the tree. So we stop the process.

That means here we have visited in the order of **A-B-D-I-J-F-C-G-K-H** using Pre-Order Traversal.

3. Post - Order Traversal (leftChild - rightChild - root):

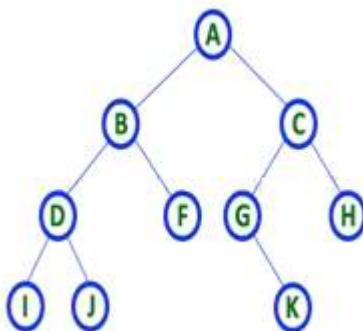
In Post-Order traversal, the root node is visited after left child and right child. In this traversal, left child node is visited first, then its right child and then its root node. This is recursively performed until the right most nodes are visited.

Algorithm:

Step-1: Visit the left subtree, using postorder.

Step-2: Visit the right subtree, using postorder

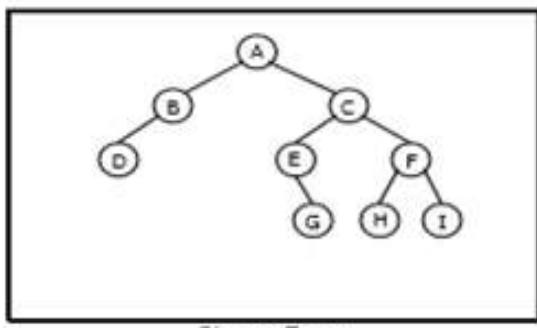
Step-3: Visit the root.



Here we have visited in the order of **I - J - D - F - B - K - G - H - C - A** using Post-Order Traversal.

Example 1:

Traverse the following binary tree in pre, post, inorder and level order.

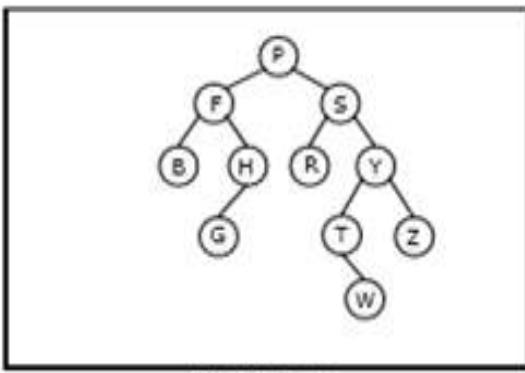


- Preorder traversal yields:
A, B, D, C, E, G, F, H, I
- Postorder traversal yields:
D, B, G, E, H, I, F, C, A
- Inorder traversal yields:
D, B, A, E, G, C, H, F, I

Pre, Post, Inorder and level order Traversing

Example 2:

Traverse the following binary tree in pre, post, inorder and level order.



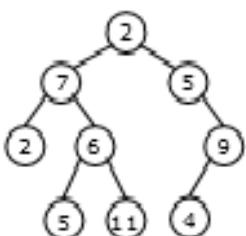
Binary Tree

- Preorder traversal yields:
P, F, B, H, G, S, R, Y, T, W, Z
- Postorder traversal yields:
B, G, H, F, R, W, T, Z, Y, S, P
- Inorder traversal yields:
B, F, G, H, P, R, S, T, W, Y, Z

Pre, Post, Inorder and level order Traversing

Example 3:

Traverse the following binary tree in pre, post, inorder and level order.



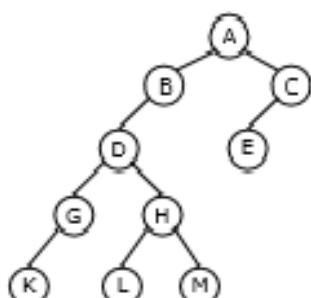
Binary Tree

- Preorder traversal yields:
2, 7, 2, 6, 5, 11, 5, 9, 4
- Postorder traversal yields:
2, 5, 11, 6, 7, 4, 9, 5, 2
- Inorder traversal yields:
2, 7, 5, 6, 11, 2, 5, 4, 9

Pre, Post, Inorder and level order Traversing

Example 4:

Traverse the following binary tree in pre, post, inorder and level order.



Binary Tree

- Preorder traversal yields:
A, B, D, G, K, H, L, M, C, E
- Postorder traversal yields:
K, G, L, M, H, D, B, E, C, A
- Inorder traversal yields:
K, G, D, L, H, M, B, A, E, C

Pre, Post, Inorder and level order Traversing

PROGRAMS ON DATA STRUCTURES

1. Write a C program to implement stack using arrays.
 2. Write a C program to implement queue using arrays.
 3. Write a C program implement the following Stack applications
 - a) infix into postfix
 - b) Evaluation of the postfix expression
 4. Write a C program to implement the following types of queues
 - a) Priority queue
 - b) Circular queue
 5. Write a C program to implement the Singly Linked List
 6. Write a C program to implement the doubly Linked List
 7. Write a C program to implement the following search algorithms.
 - i) Linear search
 - ii) Binary search
 - iii) Fibonacci search
 8. Write a C program to implement the sorting algorithms.
 9. Write a C program to implement binary tree using arrays and to perform binary traversals.
 - i) Inorder
 - ii) preorder
 - iii) post order
 10. Write a C program to balance a given tree.

1: STACK USING ARRAYS

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
int ch,max,item,top=-1,s[20]; void menu(void);
void push(int); int pop(void); void display(void); void main()
{
clrscr();
printf("ENTER STACK SIZE:"); scanf("%d",&max);
menu();
getch();
}
void menu()
{
printf("1.PUSH\n2.POP\n3.EXIT\n"); printf("ENTER YOUR CHOICE:");
fflush(stdin);
scanf("%d",&ch);
switch(ch)
{
case 1:printf("ENTER THE ELEMENT\n"); scanf("%d",&item);
push(item);
menu();
break;
case 2:item=pop(); menu();
break;
case 3:exit(0);
}
}
void push(int item)
{
if(top==max-1)
printf("STACK IS OVER FLOW\n"); else
{
top++;
s[top]=item;
}
display();
}
int pop()
{
if(top==-1)
{
printf("STACK IS UNDER FLOW\n"); return 0;
}
else
{
item=s[top]; top--;
}
display(); return item;
```

```
}

void display()
{
int i;
printf(" top -->");
for(i=top;i>=0;i--)
printf("%d\n\t",s[i]);
}
```

OUTPUT:

Enter stack size: 3

1. Push
2. Pop
3. Exit

Enter your choice:1

Enter the element: 3

Top: 3

1. Push
2. Pop
3. Exit

Enter your choice:1

Enter the element: 5

Top: 5
3

1. Push
2. Pop
3. Exit

Enter your choice:1

Enter the element: 9

Top: 9 5 3

1. Push
2. Pop
3. Exit

Enter your choice: 1

Enter the element: 15

Stack is overflow

Top: 9 5 3

1. Push
2. Pop
3. Exit

Enter your choice: 3

Popped element is: 9

Top: 5 3

1. Push
2. Pop
3. Exit

Enter your choice: 2

Popped element is: 5

Top: 3

1. Push

- 2. Pop
 - 3. Exit
- Enter your choice: 2 Stack is underflow
- 1. Push
 - 2. Pop
 - 3. Exit
- Enter your choice: 3

2. QUEUE USING ARRAYS

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h> void insertion(void); void deletion(void); void display(void);
int q[10],n,i,f,r;
int f=0,r=0; void main()
{
int op;
clrscr();
printf("ENTER THE SIZE OF QUEUE:"); scanf("%d",&n);
while(1)
{
printf("\n1.INSERTION\n2.DELETION\n3.DISPLAY\n4.EXIT\n");
printf("ENTER YOUR OPTION:");
scanf("%d",&op);
switch(op)
{
case 1:insertion(); break;
case 2:deletion(); break;
case 3:display(); break; default:exit(0);
}
}
void insertion()
{
if(r>=n)
printf("QUEUE IS OVER FLOW"); else
{
r=r+1;
printf("\nEnter AN ELEMENT TO INSERT:"); scanf("%d",&q[r]);
if(f==0)
f=1;
}
}
void deletion()
{
if(f==0)
printf("THE QUEUE IS EMPTY"); else
{
printf("THE DELETING ELEMENT IS:%5d",q[f]); f=f+1;
if(f>r)
f=0,r=0;
}
}
```

```

void display()
{
if(f==0)
printf("QUEUE IS EMPTY"); else
for(i=f;i<=r;i++)
printf("%5d",q[i]);
}

```

OUTPUT:

Enter the size of queue: 2

1. Insertion
2. Deletion
3. Display
4. Exit

Enter your option: 1

Enter an element to insert: q [1]:34

1. Insertion
2. Deletion
3. Display
4. Exit

Enter your option: 3 34

1. Insertion
2. Deletion
3. Display
4. Exit

Enter your option: 4

3: STACK APPLICATIONS

- a) INFIX INTO POSTFIX
- b) EVALUATION OF THE POSTFIX EXPRESSION

Program:(a)

```

#include<stdio.h>
#include<conio.h>
#define MAX 50
char stack[MAX];
int top=-1
void push(char); char pop();
int priority(char); void main()
{
char a[MAX],ch; int i;
clrscr();
printf("Enter an infix expression:\t"); gets(a);
printf("\nthe postfix expression for the given expression is:\t"); for(i=0;a[i]!='\0';i++)
{
ch=a[i];
if((ch>='a') && (ch<='z')) printf("%c",ch);
else if(ch=='(') push(ch); else if(ch==')')
{

```

```

while((ch=pop())!='('
printf("%c",ch);
}
else
{
while(priority(stack[top])>priority(ch))
printf("%c",pop());
push(ch);
}
}
while(top>-1) printf("%c",pop());
printf("\n");
getch();
}
void push(char ch)
{
if(top==MAX-1)
{
printf("STACK OVERFLOW"); return;
}
else
{
top++;
stack[top]=ch; }
char pop()
{
int x; if(top==-1)
{
printf("STACK EMPTY");
}
else
{ x=stack[top]; top--; }
return x;
}
int priority(char ch)
{
switch(ch)
{
case '^': return 4; case '*':
case '/': return 3; case '+':
case '-': return 2; default : return 0;
} }

```

OUTPUT:

Enter an infix expression:

$((a + b ((b ^ c - d))) * (e - (a / c)))$

The postfix expression for the given expression is:

a b b c ^ d - + e a c / - *

Program:(b)

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<ctype.h>
void push(char);
char pop(void);
char ex[50],s[50],op1,op2; int i,top=-1;
void main()
{
    clrscr();
    printf("Enter the expression:"); gets(ex); for(i=0;ex[i]!='\0';i++)
    {
        if(isdigit(ex[i])) push(ex[i]-48); else
        {
            op2=pop();
            op1=pop();
            switch(ex[i])
            {
                case '+':push(op1+op2);
                            break;
                case '-':push(op1-op2);
                            break;
                case '*':push(op1*op2);
                            break;
                case '/':push(op1/op2);
                            break;
                case '%':push(op1%op2);
                            break;
                case '^':push(pow(op1,op2));
                            break;
            }      }      }
    printf("result is :%d",s[top]); getch();
}
void push(char a)
{ s[++top]=a; }
char pop()
{ return(s[top--]); }
```

OUTPUT:

Enter the expression: 384 * 2 / +83----
Result is: 14

4: TYPES OF QUEUES

- a).Priority queue
- b).Circular queue

Program: (a)

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
typedef struct node
{
    int priority;
    int info;
    struct node *link;
}n;
n *getnode()
{
    return ( (n *)malloc(sizeof(n)));
}
n*front=NULL,*temp=NULL,*ptr=NULL,*q=NULL;
void insertion();
void deletion();
void display();
void main()
{
    int ch;
    clrscr();
    printf("\tMenu\n1.Insertion\n2.Deletion\n3.Display\n4.exit");
    while(1)
    {
        printf("Enter your choice");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:insertion();
                      break;
            case 2:deletion();
                      break;
            case 3:display();
                      break;
            case 4:exit();
            default : printf("\nInvalid choice ");
                      break;
        }
    }
    void insertion()
    { int item,item_prt;
        temp=getnode();
        printf("Enter item to insert ");
        scanf("%d",&item);
        printf("Enter item prority ");
    }
```

```

scanf("%d",&item_prt);
temp->priority=item_prt;
temp->info=item;
if(front==NULL| |item_prt>front->priority)
{
    temp->link=front;
    front=temp;
}
else
{
    q=front;
    while (q->link!=NULL &&q->link-> priority >=item_prt)
        q=q->link;
    temp->link=q->link;
    q->link=temp;
}
}
void deletion()
{ if(front==NULL)
    printf("Queue is underflow");
else
{
    temp=front;
    printf("Deleted item is %d\n",
          temp->info);
    front=front->link;
    free(temp);
}
}
void display()
{
ptr=front;
if(front==NULL)
printf("Queue is underflow");
else
{
    printf("Queue is :\n");
    printf("priority item :\n");
    while(ptr!=NULL)
    {
        printf("%5d %5d\n",ptr->priority,ptr->info);
        ptr=ptr->link;
    }
}
}
}

```

OUTPUT:

1 - Insert an element into queue
2 - Delete an element from queue
3 - Display queue elements
4 - Exit
Enter your choice: 1

Enter value to be inserted: 20
Enter your choice: 1
Enter value to be inserted: 45
Enter your choice: 1
Enter value to be inserted: 89
Enter your choice: 3
89 45 20
Enter your choice: 1
Enter value to be inserted: 56
Enter your choice: 3
89 56 45 20
Enter your choice: 2
Enter value to delete: 45
Enter your choice: 3
89 56 20
Enter your choice: 4

Program: (b)

```
#include<stdio.h>
#include<conio.h>
#define max 3
int q[max],rear=-1,front=-1;
void main()
{ int ch;
clrscr();
do
{ printf("\nqueue implementation\n");
printf("1.insert 2.delete 3.display 4.exit\n");
printf("enter your choice\n");
scanf("%d",&ch);
switch(ch)
{ case 1:insert(); break;
 case 2:delete(); break;
 case 3:display(); break;
 case 4:exit(1);
 default:printf("wrong choice\n"); break;
}
}while(ch<=4);
getch();
}
insert()
{ int item;
```

```

if(rear==max-1)
{ printf("queue overflow\n"); }
else
{ if(front==-1)
  front=0;
  printf("insert the element in queue:");
  scanf("%d",&item);
  rear++;
  q[rear]=item;
}
}
delete()
{ if(front==-1)
{ printf("queue underflow\n");
}
else
{ printf("element deleted from queue is:%d\n",q[front]);
  front++;
  if(front==max)
    front=rear=-1;
}
}
display()
{ int i;
if(front==-1)
printf("queue is empty\n");
else
{ printf("queue is :\n");
  for(i=front;;i++)
{ printf("%2d",q[i]);
  if(i==rear)
    return;
} } }

```

OUTPUT:

1. Insert
2. Delete
3. Display
4. Exit

Enter your choice:1

Enter element to cqueue: 10

1. Insert
2. Delete
3. Display
4. Exit

Enter your choice: 1

Enter element to circular queue: 20

1. Insert
2. Delete

```

3. Display
4. Exit
Enter your choice: 2
Deleted element from circular queue is: 10
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Elements from circular queue is: 20
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4

```

5: SINGLY LINKED LIST

Program:

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<alloc.h> struct node
{
int data;
struct node* link; };
typedef struct node* pnode; pnode head=NULL;
void menu(void); void insbeg(int); void delbeg(void); void insend(int); void delend(void);
void insafter(int,int); void delmid(int); void display(void); void main()
{
int ch,x,pos; clrscr(); while(1)
{
menu();
printf("enter ur choice\n"); scanf("%d",&ch);
switch(ch)
{
case 1: printf("enter element to insert\n");
scanf("%d",&x);
insbeg(x);
break;
case 2:delbeg();
break;

case 3: printf("enter element to insert\n");
scanf("%d",&x);
insend(x);
break;
case 4:delend();
break;
case 5: printf("enter element,pos to insert\n");
}

```

```

scanf("%d%d",&x,&pos);
insafter(x,pos);
break;
case 6:printf("enter position of element to delete\n"); scanf("%d",&pos);
delmid(pos);
break;
case 7:display();
break;
case 8:exit(0);
} } }
void menu()
{
printf("1.insbeg\n2.delbeg\n3.insend\n4.delend\n");
printf("5.insafter\n6.delmid\n7.display\n8.exit\n");
}
void insbeg(int x)
{
pnode ptr;
ptr=(pnode)malloc(sizeof(struct node));
ptr->data=x;
ptr->link=head; head=ptr;
}
void delbeg()
{
pnode tmp; int x;
if(head==NULL)
{
printf("list is empty\n"); return;
}
tmp=head;
head=tmp->link;
printf("deleted element is %d\n",tmp->data);
free(tmp);
}
void insend(int x)
{
pnode tmp,ptr;
ptr=(pnode)malloc(sizeof(struct node));
ptr->data=x;
ptr->link=NULL; if(head==NULL)
{
head=ptr;
}
else
{
tmp=head;
while(tmp->link!=NULL)
tmp=tmp->link;
tmp->link=ptr;
}
}

```

```

    }
}

void delend()
{
pnodeprev=NULL,ptr=head;
int x; if(head==NULL)
{
    printf("list is empty\n"); return;
}
while(ptr->link!=NULL)
{
prev=ptr; ptr=ptr->link;
}
if(prev==NULL)
    head=NULL;
else
    prev->link=NULL;
printf("deleted node:%d\n",ptr->data);
free(ptr);
}
void insafter(intx,intpos)
{
pnodetmp=head,ptr;
int i;
for(i=1;i<pos;i++)
{
    tmp=tmp->link;
    if(tmp==NULL)
    {
        printf("position out of range\n"); return;
    }
}
ptr=(pnode)malloc(sizeof(struct node));
ptr->data=x;
ptr->link=tmp->link;
tmp->link=ptr;
}
void delmid(intpos)
{
pnodeprev=NULL,tmp=head;
int i;

if(head==NULL)
{
    printf("list is empty\n"); return;
}
for(i=0;i<pos;i++)
{
prev=tmp; tmp=tmp->link;
if(tmp==NULL)
{ printf("position out of range\n"); return;
}

```

```

    }
}

if(prev!=NULL)
    prev->link=tmp->link;
else
    head=tmp->link;
printf("deleted element: %d\n",tmp->data);
free(tmp);
}
void display()
{
pnodeptr=head;
while(ptr!=NULL)
{
printf("%d-->",ptr->data);
ptr=ptr->link;
}
printf("\n");
getch();
}

```

OUTPUT:

1. Ins beg
2. el beg
3. Ins end
4. Del end
5. Ins after
6. Del mid
7. Display
8. Exit

Enter your choice: 1

Enter element to insert: 94

1. Ins beg
2. Del beg
3. Ins end
4. Del end
5. Ins after
6. Del mid
7. Display
8. Exit

Enter your choice: 1

Enter element to insert: 90

1. Ins beg
2. Del beg
3. Ins end
4. Del ed
5. Ins after
6. Del mid
7. Display
8. Exit

Enter your choice 5
Enter element, Pos to insert
55 2

1. Ins beg
2. Del beg
3. Ins end
4. Del end
5. Ins after
6. Del mid
7. Display
8. Exit

Enter your choice 7 90->94->55->

1. Ins beg
2. Del beg
3. Ins end
4. Del end
5. Ins after
6. Del mid
7. Display
8. Exit

Enter your choice 8

6: DOUBLY LINKED LIST

Program:

```
#include<stdio.h>
#include<conio.h> struct node
{
    struct node *prev; int data;
    struct node *nxt;
}
*head=NULL,*curr=NULL,*curr1=NULL,*p;

void insert(int pos)
{
    int count=1,i; p=head;
    while(p->nxt!=NULL)
    {
        count++; p=p->nxt;
    }
    p=head;
    if(pos<=count+1)
    {
        curr=(struct node*)malloc(sizeof(struct node));
        printf("Enter the node:");
        scanf("%d",&curr->data);
        curr->nxt=NULL;
        curr->prev=NULL;
        if(head==NULL)
```

```

{
head=curr; }
else if(pos==1)
{
head->prev=curr;
curr->nxt=head;
head=curr; }
else
{ for(i=1;i<(pos-1);i++) p=p->nxt;
curr->prev=p; curr->nxt=p->nxt;
p->nxt->prev=curr;
p->nxt=curr;
}
printf("\n%d inserted at pos:%d!\n",curr->data,pos);
}
else
printf("\nEnter a valid position!");

}

void deletenode(int data)
{
int found=0; curr=head; if(head->data == data)
{
(head->nxt)->prev=NULL; head=head->nxt;
printf("\n%d deleted!\n",curr->data);
free(curr);
}
else
{
curr=curr->nxt;
while(curr->nxt!=NULL)
{
if(curr->data==data)
{
found=1;
break;
}
else
curr=curr->nxt;
}
if(found==1 || curr->data==data)
{
curr1=curr->prev; curr1->nxt=curr->nxt; (curr->nxt)->prev=curr1; printf("\n%d
deleted!\n",curr->data); free(curr);
}
else
printf("\n%d is not present in the list!\n",data);
} }
void display()

```

```

{
curr=head;
if(head==NULL)
    printf("\nList is empty!\n"); else {
printf("\nList:\n"); while(curr->nxt!=NULL) {
printf("%d\t",curr->data); curr=curr->nxt;
}
printf("%d\n",curr->data);
}
}

void main()
{
int op,data;
clrscr();
printf("Creation of Doubly Linked List\n");
curr=(struct node*)malloc(sizeof(struct node));
curr->nxt=NULL;
curr->prev=NULL; printf("Enter the first node:"); scanf("%d",&curr->data); head=curr;
head->nxt=NULL; head->prev=NULL; do
{ printf("\nDOUBLY LINKED LIST OPERATIONS:\n1.Insert a node ");
printf("\n2.Delete a node\n3.Display\n4.Exit\n");
printf("\nEnter an operation:"); scanf("%d",&op);
switch(op)
{
case 1:
{ printf("Enter the position where you want to insert the node:"); scanf("%d",&data);
insert(data); break;
}
case 2:{ printf("Enter the node to be deleted:"); scanf("%d",&data);
deletenode(data); break; }

case 3:      {
display();
break; }
case 4:
break;
default:
printf("\nEnter a valid option!");
}
}while(op!=4);
getch();
}

```

OUTPUT:

Doubly linked list operations

1. Insert node
2. Delete node
3. Display

4. Exit
Select an operation: 1
Enter the position to insert node: 1 Enter the node: 59
59 inserted at pos: 1
1. Insert node
2. Delete node
3. Display
4. Exit
Select an operation: 4

7: SEARCHING ALGORITHMS

- i) Linear search ii) Binary search
- iii) Fibonacci search

Program: (i)and (ii)

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void main()
{
    Int a[10],n,flag=0,i,lb,ub,key,mid,ch;
    clrscr();
    printf("enter the size of the elements\n");
    scanf("%d",&n);
    printf("enter the elements\n");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    printf("enter any key element to search\n");
    scanf("%d",&key);
    printf("menu\n");
    printf("\n1.linear search\n2.binary search \n");
    printf("enter your choice:\n"); scanf("%d",&ch);
    switch(ch)
    { case 1:for(i=0;i<n;i++) if(a[i]==key)
    {
        flag=1;
        break;}
    case 2:for(lb=0,ub=n-1;lb<=ub;)
    {
        mid=(lb+ub)/2;
        if(key==a[mid])
        {
            flag=1;
            break;
        }
        else if(key<a[mid]) ub=mid-1;
        else lb=mid+1;
    }
}
```

```

break;
default:exit(0);
}
if(flag==1)
printf("seach is successful");
else
printf("search is not successful \n");
getch();

}

```

OUTPUT:

Enter the size of the elements 5

Enter the elements 3 2 6 3 9 5

Enter any element to search 3

Menu

1. Linear search
 2. Binary search Enter your choice: 2
- Search is successful

Program:(iii)

```

#include<stdio.h>
void main()
{
    int n,a[50],i,key,loc,p,q,r,m,fk;
    clrscr();
    printf("\nEnter number elements to be entered");
    scanf("%d",&n);
    printf("enter elements");
    for(i=1;i<=n;i++)
        scanf("%d",&a[i]);
    printf("enter the key element");
    scanf("%d",&key);
    fk=fib(n+1);
    p=fib(fk);
    q=fib(p);
    r=fib(q);
    m=(n+1)-(p+q);
    if(key>a[p])
        p=p+m;
    loc=rfibsearch(a,n,p,q,r,key);
    if(loc==0)
        printf("key is not found");
    else
        printf("%d is found at location %d",key,loc);
    getch();
}
int fib(int m)
{
    int a,b,c;

```

```

a=0;
b=1;
c=a+b;
while(c<m)
{
    a=b;
    b=c;
    c=a+b;
}
return b;
}
int rfibsearch(int a[],int n,int p,int q,int r,int key)
{
    int t;
    if(p<1 || p>n)
        return 0;
    else if(key==a[p])
        return p;
    else if(key<a[p])
    {
        if(r==0)
            return 0;
        else
        {
            p=p-r;
            t=q;
            q=r;
            r=t-r;
            return rfibsearch(a,n,p,q,r,key);
        }
    }
    else
    {
        if(q==1)
            return 0;
        else
        {
            p=p+r;
            q=q-r;
            r=r-q;
            return rfibsearch(a,n,p,q,r,key);
        }
    }
}

```

OUTPUT:

Enter the number elements to be entered 8

Enter the elements 1 3 2 5 4 6 7 9

Enter the key element 9

8 is found at location 8

8: SORTING ALGORITHMS

Program: Bubble Sort

```
#include<stdio.h>
#include<conio.h>
#define TRUE 1
#define FALSE 0
void bubblesort(int x[],int n); void main()
{
intnum[10],i,n;
clrscr();
printf("Enter the no of elements\n"); scanf("%d",&n);
printf("Enter the elements\n"); for(i=0;i<n;i++) scanf("%d",&num[i]); bubblesort(num,n);
printf("sorted elements are\n"); for(i=0;i<n;i++) printf("%d\t",num[i]);
getch();}
void bubblesort(int x[],int n)
{
inthold,j,pass,K=TRUE;
for(pass=0;pass<n-1&&K==TRUE;pass++)
{
K=FALSE;
for(j=0;j<n-pass-1;j++)
{
if(x[j]>x[j+1])
{
K=TRUE;
hold=x[j];
x[j]=x[j+1];
x[j+1]=hold;}}}
```

OUTPUT:

```
Enter the no of elements 5
Enter the elements 36 23 59 68 2
Sorted elements are 2 23 36 59 68
```

Program: selection sort

```
#include<stdio.h>
#include<conio.h> void main()
{
intn,i,j,a[10],min,t;
clrscr();
printf("enter how many elements\n"); scanf("%d",&n);
printf("enter the elements\n"); for(i=0;i<n;i++) scanf("%d",&a[i]); for(i=0;i<n-1;i++)
{
min=i;
for(j=i+1;j<n;j++)
{
if(a[min]>a[j])
min=j;
}
t=a[i];
a[i]=a[min];
a[min]=t;
```

```

}
printf("the sorted elements are \n"); for(i=0;i<n;i++)
printf("%5d",a[i]);
getch();
}

```

OUTPUT:

Enter how many elements 5
 Enter the elements 56 48 46 23 35
 The sorted elements are 23 35 46 56 98

Program: insertion sort

```

#include<stdio.h>
#include<conio.h> void main()
{
intn,i,a[10],t,j;
clrscr();
printf("enter how many elements\n");
scanf("%d",&n);
printf("enter the elements\n");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
for(i=1;i<n;i++)
{
for(j=i;j>0;j--)
{
if(a[j]<a[j-1])
{
t=a[j]; a[j]=a[j-1]; a[j-1]=t;
} } }
printf("the sorted elements are\n"); for(i=0;i<n;i++)
printf("%5d",a[i]);
getch();
}

```

OUTPUT:

Enter how many elements 5
 Enter the elements 26 36 98 12 5
 The sorted elements are 5 12 26 36 98

Program:Quick sort

```

#include<stdio.h>
#include<conio.h>
void quick(int a[10],intlb,int n);
void main()
{
intn,i,a[10];
clrscr();
printf("enter how many elements \n"); scanf("%d",&n);
printf("enter the elements \n"); for(i=0;i<n;i++) scanf("%d",&a[i]); quick(a,0,n-1);
printf("the sorted elements are \n"); for(i=0;i<n;i++)

```

```

printf("%d \n",a[i]); getch();
}
void quick(int a[],int lb,int ub)
{
int i,j,t,key;
if(lb>ub) return; i=lb;
j=ub;
key=lb;
while(i<j)
{
while(a[key]>a[i])
i++;
while(a[key]<a[j]) j--;
if(i<j)
{
t=a[i];
a[i]=a[j];
a[j]=t;
}
}
t=a[j];
a[j]=a[key];
a[key]=t;
quick(a,0,j-1);
quick(a,j+1,ub);
}

```

OUTPUT:

```

Enter how many elements 5
Enter the elements 65 23 89 68 71
The sorted elements are 23 65 68 71 89

```

program:heap sort

```

#include<conio.h>
void maxheap(int [],int,int);
void buildmaxheap(int a[],int n)
{
int i; for(i=n/2;i>=1;i--)
{
maxheap(a,i,n);
}
}
void maxheap(int a[],int i,int n)
{
int R,L,largest,t;
L=2*i;
R=2*i+1;
if((L<=n) && (a[L]>a[i])) largest=L;
else largest=i;
if((R<=n) && (a[i]>a[largest])) largest=R;

```

```

if(largest!=i)
{
t=a[i];
a[i]=a[largest];
a[largest]=t;
maxheap(a,largest,n);
}
}

void heapsort(int a[],int n)
{
int i,temp;
buildmaxheap(a,n);
for(i=n;i>=2;i--)
{
temp=a[1];
a[1]=a[i];
a[i]=temp; maxheap(a,1,i-1);
}
}
}

void main()
{
int a[50],i,n; clrscr();
printf("Enter the size of array : "); scanf("%d",&n);
printf("Enter the elements of array \n"); for(i=1;i<=n;i++)
{
scanf("%d",&a[i]);
}
heapsort(a,n);
printf("sorted array is \n"); for(i=1;i<=n;i++)
{
printf("%d\t",a[i]);
}
getch();
}

```

OUTPUT:

```

Enter the size of array: 4
Enter the elements of array: 35 21 95 17
Sorted array is: 17 21 35 95

```

Program: merge sort

```

#include<stdio.h>
#include<conio.h>
void merge(int [],int ,int ,int );
void part(int [],int ,int );
int main()
{
intarr[30];
int i,size;
printf("\n\t----- Merge sorting method ----- \n\n");
printf("Enter total no. of elements : "); scanf("%d",&size);

```

```

for(i=0; i<size; i++)
{printf("Enter %d element : ",i+1);
 scanf("%d",&arr[i]);
}
part(arr,0,size-1);
printf("\n\t----- Merge sorted elements ----- \n\n"); for(i=0; i<size; i++)

printf("%d ",arr[i]); getch();
return 0;
}
void part(intarr[],intmin,int max)
{
int mid; if(min<max)
{
mid=(min+max)/2;
part(arr,min,mid);

part(arr,mid+1,max);
merge(arr,min,mid,max);
}
}
void merge(intarr[],intmin,intmid,int max)
{
inttmp[30];
inti,j,k,m;
j=min;
m=mid+1;
for(i=min; j<=mid && m<=max ; i++)
{
if(arr[j]<=arr[m])
{
tmp[i]=arr[j];
j++;
}
else
{
tmp[i]=arr[m];
m++;
}
}
if(j>mid)
{
for(k=m; k<=max; k++)
{
tmp[i]=arr[k];
i++;
}
}
else

```

```

{
for(k=j; k<=mid; k++)
{
tmp[i]=arr[k];
i++;
}
}
for(k=min; k<=max; k++) arr[k]=tmp[k];
}

```

OUTPUT:

Merge sorting method

Enter total no of elements: 4

Enter 4 elements:

35

95

17

21

Merge sorted elements: 17 21 35 95

9.BINARY TREE TRAVERSALS

i) Preorder ii) Inorder iii) Postorder

Program:

```

#include<stdio.h>
#include<conio.h>
#include<malloc.h>
#include<stdlib.h> struct node
{
int data;
struct node *left; struct node *right; };
typedef struct node *pnode; pnode root=NULL;
void insert(int val)
{
pnode p,q,t; t=(pnode)malloc(sizeof(struct node)); t->left=t->right=NULL;
t->data=val; if(root==NULL)
{
root=t;
return;
}
p=root;q=NULL;
while(p)
{
if(p->data==val)
{
return;
}
q=p;
if(val<p->data) p=p->left;
else if(val>p->data)
p=p->right;
}

```

```

}

if(val<q->data) q->left=t; if(val>q->data) q->right=t;
}

int search(int key)
{
pnode p=root;
while(p)
{
if(p->data==key) return 1;
else if(key<p->data) p=p->left;
else if(key>p->data) p=p->right;
}
return 0;
}

void inorder(pnode p)
{
if(p==NULL)
return; inorder(p->left); printf("%3d",p->data); inorder(p->right);
}

void preorder(pnode p)
{
if(p==NULL)
return;
printf("%3d",p->data);
preorder(p->left);
preorder(p->right);
}

void postorder(pnode p)
{
if(p==NULL)
return;
postorder(p->left);
postorder(p->right);
printf("%3d",p->data);
}

int main()
{
intch,x;
clrscr();
while(1)
{
printf("\n1.insertion");
printf("\n2.inorder");
printf("\n3.preorder");
printf("\n4.postorder");
printf("\n5.search");
printf("\n6.exit");
printf("\nEnterur choice\n");
scanf("%d",&ch);
}

```

```

switch(ch)
{
case 1:printf("enter an elements\n"); scanf("%d",&x);
insert(x);
break;
case 2:inorder(root); break;
case 3:preorder(root); break;
case 4:postorder(root); break;
case 5:printf("enter key elements\n");
scanf("%d",&x);
if(search(x))
printf("found"); else
printf("not found"); break;
case 6:exit(0);
}
}

```

OUTPUT:

Tree traversal

Enter the number of terms to add 7 Enter the item 15

Enter the item 7 Enter the item 9 Enter the item 18 Enter the item 6 Enter the item 21 Enter the item 2

In order traversal 2 6 7 9 15 18 21

Pre order traversal 15 7 6 2 9 18 21

Post order traversal 2 6 9 7 21 18 15

10.BALANCE A TREE

Program:

```

#include <stdio.h>
#include <stdlib.h>
struct btnode
{
    int value;
    struct btnode *l;
    struct btnode *r;
};

typedef struct btnode N;
N* bst(int arr[], int first, int last);
N* new(int val);
void display(N *temp);
int main()
{
    int arr[] = {10, 20, 30, 40, 60, 80, 90};
    N *root = (N*)malloc(sizeof(N));
    int n = sizeof(arr) / sizeof(arr[0]), i;

    printf("Given sorted array is\n");
    for (i = 0; i < n; i++)

```

```

    printf("%d\t", arr[i]);
    root = bst(arr, 0, n - 1);
    printf("\n The preorder traversal of binary search tree is as follows\n");
    display(root);
    printf("\n");
    return 0;
}
N* new(int val)
{
    N* node = (N*)malloc(sizeof(N));

    node->value = val;
    node->l = NULL;
    node->r = NULL;
    return node;
}

N* bst(int arr[], int first, int last)
{
    int mid;
    N* temp = (N*)malloc(sizeof(N));
    if (first > last)
        return NULL;
    mid = (first + last) / 2;
    temp = new(arr[mid]);
    temp->l = bst(arr, first, mid - 1);
    temp->r = bst(arr, mid + 1, last);
    return temp;
}
void display(N *temp)
{
    printf("%d->", temp->value);
    if (temp->l != NULL)
        display(temp->l);
    if (temp->r != NULL)
        display(temp->r);
}

```

OUTPUT:

Given sorted array is

10 20 30 40 60 80 90

The preorder traversal of binary search tree is as follows

40->20->10->30->80->60->90

Linux And Shell Scripting

DECAP448

**Edited by
Rishi Chopra**



LOVELY
PROFESSIONAL
UNIVERSITY



L O V E L Y
P R O F E S S I O N A L
U N I V E R S I T Y

Linux And Shell Scripting

Edited By:
Rishi Chopra

CONTENT

Unit 1:	Getting Started with Linux	1
	<i>Divya, Lovely Professional University</i>	
Unit 2:	Installation Guide	11
	<i>Divya, Lovely Professional University</i>	
Unit 3:	Connecting to Internet	43
	<i>Divya, Lovely Professional University</i>	
Unit 4:	Installing Software	57
	<i>Divya, Lovely Professional University</i>	
Unit 5:	Utilities	73
	<i>Divya, Lovely Professional University</i>	
Unit 6:	File Systems	99
	<i>Divya, Lovely Professional University</i>	
Unit 7:	The Shell and popular Editors	126
	<i>Divya, Lovely Professional University</i>	
Unit 8:	The Bourne Again Shell and TC Shell	166
	<i>Divya, Lovely Professional University</i>	
Unit 9:	Programming the Bourne Again Shell	212
	<i>Divya, Lovely Professional University</i>	
Unit 10:	Linux System Administration	242
	<i>Divya, Lovely Professional University</i>	
Unit 11:	Web Server Configuration	291
	<i>Divya, Lovely Professional University</i>	
Unit 12:	File Server Configuration	308
	<i>Divya, Lovely Professional University</i>	
Unit 13:	Samba Servers	323
	<i>Divya, Lovely Professional University</i>	
Unit 14:	Network File Systems	336
	<i>Divya, Lovely Professional University</i>	

Unit 01: Getting Started with Linux

CONTENTS

Objectives

Introduction

1.1 The History of UNIX and GNU-Linux

1.2 What is so good about Linux

1.3 Why Linux Is Popular with Hardware Companies and Developers

1.4 Overview of Linux

Summary

Keywords

Self Assessment

Answers for Self Assessment

Review Questions:

Further Readings

Objectives

After studying this unit, you will be able to

- understand the operating system
- know the history of Linux
- understand the features of Linux
- understand the basic commands of Linux
- understand the shell of Linux

Introduction

An operating system is the low-level software that schedules tasks, allocates storage, and handles the interfaces to peripheral hardware, such as printers, disk drives, the screen, keyboard, and mouse. An operating system has two main parts: the kernel and the system programs.

- The kernel allocates machine resources – including memory, disk space, and CPU cycles – to all other programs that run on the computer.
- The system programs include device drivers, libraries, utility programs, shells (command interpreters), configuration scripts and files, application programs, servers, and documentation.

The Linux kernel was developed by Linus Torvalds. He released Linux version 0.01 in September 1991. The name ‘Linux’ is a combination of Linus and UNIX. The Linux OS, is a product of the Internet and is a free OS. Linux is free software. “Free software” is a matter of liberty, not price. To understand the concept, you should think of “free” as in “free speech,” not as in “free beer.” The UNIX OS is the common ancestor of Linux. A Linux distribution comprises the Linux kernel, utilities, and application programs. Many distributions are available, including Ubuntu, Fedora, Red Hat, Mint, OpenSUSE, Mandriva, CentOS, and Debian.

1.1 The History of UNIX and GNU-Linux

The Heritage of Linux: UNIX

The UNIX system was developed by researchers who needed a set of modern computing tools to help them with their projects. When the UNIX OS became widely available in 1975, Bell Labs offered it to educational institutions at nominal cost. The schools, colleges, universities and industries accepted the OS and worked on it.

Linux-BSD

One version is called the Berkeley Software Distribution (BSD) of the UNIX system (or just Berkeley UNIX).

Fade to 1983

Richard Stallman announced the GNU Project for creating an operating system, both kernel and system programs. GNU, which stands for Gnu's Not UNIX, is the name for the complete UNIX-compatible software system.

Next Scene, 1991

The GNU Project has moved well along toward its goal. Much of the GNU operating system, except for the kernel, is complete.

The Code is Free

The tradition of free software dates back to the days when UNIX was released to universities at nominal cost, which contributed to its portability and success. This tradition eventually died as UNIX was commercialized. Another problem with the commercial versions of UNIX related to their complexity.

1.2 What is so good about Linux

- Standards
- Applications
- Peripherals
- Software
- Platforms
- Emulators
- Virtual Machines
- Xen
- VMWare
- KVM
- Qemu
- Virtual Box

Standards

In 1985, the POSIX standard was developed, which is based largely on the SVR4 and other earlier standardization efforts. These efforts were spurred by the U.S. government, which needed a standard computing environment to minimize its training and procurement costs.

Applications

A rich selection of applications is available for Linux – both free and commercial – as well as a wide variety of tools: graphical, word processing, networking, security, administration, Web server, and many others.

Peripherals

Linux often supports a peripheral or interface card before any company does. Unfortunately some types of peripherals – particularly proprietary graphics cards – lag in their support.

Software's

Also important to users is the amount of software that is available – not just source code but also prebuilt binaries that are easy to install and ready to run. These programs include more than free software.

Platforms

Linux is not just for Intel-based platforms: It has been ported to and runs on the PowerPC. Nor is Linux just for single-processor machines.

Emulators

Linux supports programs, called emulators, that run code intended for other operating systems.

Virtual Machines

A virtual machine appears to the user and to the software running on it as a complete physical machine. The software that provides the virtualization is called a virtual machine monitors (VMM) or hypervisor. Each VM can run a different OS from the other VMs.

Xen

Xen, which was created at the University of Cambridge and is now being developed in the open-source community, is an open-source VMM. Xen introduces minimal performance overhead when compared with running each of the operating systems natively.

VMWare

VMware, Inc. offers VMware Server, a free, downloadable, proprietary product you can install and run as an application under Linux. VMware Server enables you to install several VMs, each running a different OS, including Windows and Linux.

KVM

The Kernel-based Virtual Machine (KVM) is an open-source VM and runs as part of the Linux kernel.

Qemu

Qemu, is an open-source VMM that runs as a user application with no CPU requirements. It can run code written for a different CPU than that of the host machine.

Virtual Box

Virtual Box is an open-source VM developed by Sun Microsystems.

1.3 Why Linux Is Popular with Hardware Companies and Developers

Two trends in the computer industry set the stage for the growing popularity of UNIX and Linux. These are proprietary and generic operating systems.

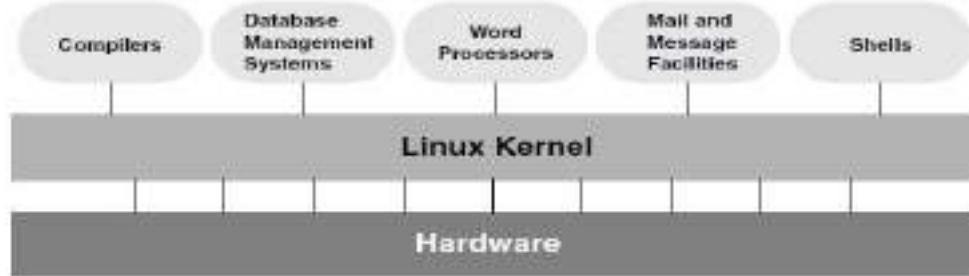
Proprietary operating systems: A proprietary OS is one that is written and owned by the manufacturer of the hardware (for example, DEC/Compaq owns VMS). Today's manufacturers need a generic OS that they can easily adapt to their machines.

Generic operating systems: Generic OS is written outside of the company manufacturing the hardware and is sold (UNIX, OS X, Windows) or given (Linux) to the manufacturer. Linux is a generic OS.

Linux emerged to serve both needs: It is a generic OS that takes advantage of available hardware. A portable OS is one that can run on many different machines. More than 95% of the Linux OS is written in the C programming language. Because Linux is portable, it can be adapted (ported) to different machines and can meet special requirements. The ancestor of Linux is UNIX. The UNIX OS was written in assembly language. For this reason, the original UNIX OS was not portable. To make UNIX portable, Thompson developed the B programming language, a machine-independent language. Dennis Ritchie developed the C programming language by modifying B and, with Thompson, rewrote UNIX in C in 1973.

1.4 Overview of Linux

Like UNIX, it is also a well-thought-out family of utility programs and a set of tools that allow users to connect and use these utilities to build systems and applications.



Kernel Programming Interface

The Linux kernel—the heart of the Linux OS—is responsible for allocating the computer's resources and scheduling user jobs so each one gets its fair share of system resources. Programs interact with the kernel through system calls. A programmer can use a single system call to interact with many kinds of devices. For example, there is one write() system call, rather than many device-specific ones. It also makes it possible to move programs to new versions of the OS without rewriting them (provided the new version recognizes the same system calls).

Supports Many Users

Depending on the hardware and the types of tasks the computer performs, a Linux system can support from 1 to more than 1,000 users, each concurrently running a different set of programs.

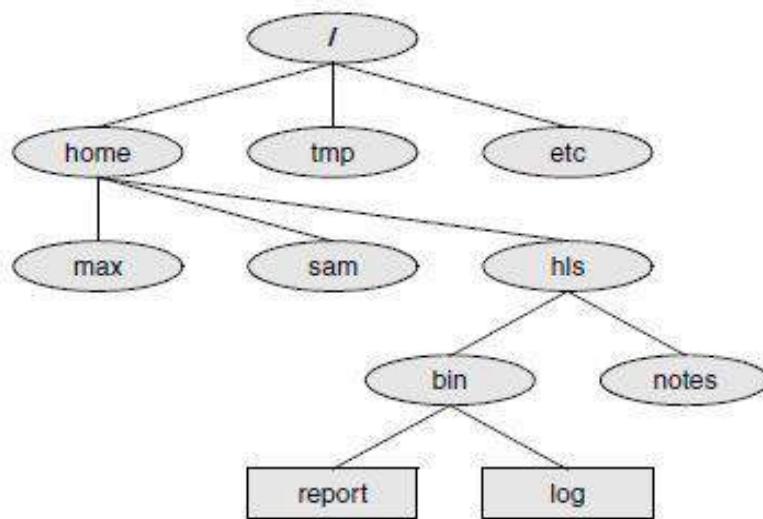
Runs Many Tasks

Linux is a fully protected multitasking OS, allowing each user to run more than one job at a time. Processes can communicate with one another but remain fully protected from one another, just as the kernel remains protected from all processes.

Secure Hierarchical File system

The Linux file system provides a structure whereby files are arranged under directories, which are like folders or boxes. Each directory has a name and can hold other files and directories. Directories, in turn, are arranged under other directories, and so forth, in a treelike organization.

Linux File System Structure



Standards

The Linux File system Standard (FSSTND) was developed, which has since evolved into the Linux File system Hierarchy Standard (FHS).

Links

A link allows a given file to be accessed by means of two or more names. The alternative names can be located in the same directory as the original file or in another directory.

Security

Like most multiuser OSs, Linux allows users to protect their data from access by other users. It also allows users to share selected data and programs with certain other users by means of a simple but effective protection scheme.

The Shell

The shell can work as a command interpreter as well as a programming language.

The shell as a command interpreter: In a textual environment, the shell—the command interpreter—acts as an interface between you and the OS. A number of shells are available for Linux. The four most popular shells are:

- Bourne Again Shell
- Debian Almquist Shell
- TC Shell
- Z Shell

The shell as a Programming Language: The shell is a high-level programming language. Shell commands can be arranged in a file for later execution which is known as shell scripts. This flexibility allows users to perform complex operations with relative ease.

Filename Generation

When you type commands to be processed by the shell, you can construct patterns using characters that have special meanings to the shell. These characters are called wildcard characters. The patterns, which are called ambiguous file references, are a kind of shorthand.

Completion

In conjunction with the Readline library, the shell performs command, filename, pathname, and variable completion.

Device-Independent Input and Output

- Redirection: When you give a command to the Linux OS, you can instruct it to send the output to any one of several devices or files. This diversion is called output redirection.
- Device independence: In a similar manner, a program's input, which normally comes from a keyboard, can be redirected so that it comes from a disk file instead.

Shell Functions

Many shells, including the BASH, support shell functions that the shell holds in memory so it does not have to read them from the disk each time you execute them.

Job Control

Job control is a shell feature that allows users to work on several jobs at once, switching back and forth between them as desired.

A Large Collection of Useful Utilities

Linux includes a family of several hundred utility programs, often referred to as commands. These utilities perform functions that are universally required by users. For example: Sort utility.

Inter process Communication

Linux enables users to establish both pipes and filters on the command line. A pipe sends the output of one program to another program as input. A filter is a special kind of pipe that processes a stream of input data to yield a stream of output data.

System Administration

On a Linux system the system administrator is frequently the owner and only user of the system. This person has many responsibilities. The first responsibility may be to set up the system, install the software, and possibly edit configuration files.

Additional Features of Linux

The developers of Linux included features from BSD, System V, and Sun Microsystems' Solaris, as well as new features, in their OS.

GUIs: Graphical User Interfaces

X11:

Given a terminal or workstation screen that supports X, a user can interact with the computer through multiple windows on the screen, display graphical information, or use special-purpose applications to draw pictures, monitor processes, or preview formatted output. Usually two layers run on top of X: a desktop manager and a window manager.

Desktop manager: A desktop manager is a picture-oriented user interface that enables you to interact with system programs by manipulating icons instead of typing the corresponding commands to a shell.

Window manager: A window manager is a program that runs under the desktop manager and allows you to open and close windows, run programs, and set up a mouse so it has different effects depending on how and where you click.

(Inter)Networking Utilities

Linux network support includes many utilities that enable you to access remote systems over a variety of networks. In addition to sending email to users on other systems, you can access files on disks mounted on other computers.

Software Development

One of Linux's most impressive strengths is its rich software development environment. Linux supports compilers and interpreters for many computer languages.

Utilities

These utilities facilitate you for a large task. There are various utilities like ls, cd, cat, mv, cp, echo and date.

Summary

- An operating system has two main parts: the kernel and the system programs.
- A Linux distribution comprises the Linux kernel, utilities, and application programs.
- Linux supports programs, called emulators, that run code intended for other operating systems.
- The software that provides the virtualization is called a virtual machine monitor.
- The Kernel-based Virtual Machine (KVM) is an open-source VM and runs as part of the Linux kernel.
- A portable OS is one that can run on many different machines.
- The four most popular shells are: Bourne Again Shell, Debian Almquist Shell, TC Shell and Z Shell.
- Shell commands can be arranged in a file for later execution which are known as shell scripts.

Keywords

- **ls utility:** A utility used for listing the contents of a directory.
- **mv utility:** A utility used for moving the files from one directory to another.
- **cd utility:** A utility used for changing the directory.
- **Operating System:** An operating system is the low-level software that schedules tasks, allocates storage, and handles the interfaces to peripheral hardware, such as printers, disk drives, the screen, keyboard, and mouse.
- **Emulators:** Linux supports programs, called emulators, that run code intended for other operating systems.
- **Proprietary OS:** It is the one that is written and owned by the manufacturer of the hardware.
- **Generic OS:** It is written outside of the company manufacturing the hardware and is sold or given to the manufacturer.

Self Assessment

1. An operating system is responsible for
 - A. Scheduling the tasks.
 - B. Allocation the storage
 - C. Handling the interfaces for peripherals
 - D. All of the above
2. An operating system has
 - A. Kernel
 - B. System programs
 - C. Both of the above
 - D. None of the above
3. Which of these are distributions of Linux?
 - A. Fedora
 - B. RedHat
 - C. CentOS
 - D. All of the above
4. A Linux distribution comprises of
 - A. Application programs
 - B. Utilities
 - C. Kernel
 - D. All of the above
5. Which of these shells are available in Linux?
 - A. TC Shell
 - B. Z Shell
 - C. Debian Almquist Shell
 - D. All of the above
6. Which is the core of operating system?
 - A. Kernel
 - B. Commands
 - C. Shell
 - D. Script

7. Which of these utilities is used to show which files and folders are available in the system?
 - A. ls
 - B. cat
 - C. rm
 - D. All of the above
8. Linux OS supports
 - A. Multi User
 - B. Multi Process
 - C. Multi-tasking
 - D. All of the above
9. Which of these utilities is used for concatenation of files?
 - A. ls
 - B. cat
 - C. rm
 - D. echo
10. Which of these utilities is used for displaying the contents of a file?
 - A. ls
 - B. cat
 - C. rm
 - D. echo
11. Linux is an example of
 - A. Web browser
 - B. Word processing software
 - C. Operating system
 - D. Photo editor
12. Who founded the Linux Kernel?
 - A. Richard Stallman
 - B. Ben Thomas
 - C. Linus Torvalds
 - D. None of the above
13. Which of the following OS is not based upon Linux?
 - A. BSD
 - B. Redhat
 - C. Ubuntu
 - D. CentOS
14. What is available to users in Linux OS?
 - A. Source code
 - B. Prebuilt binaries
 - C. Both source code and prebuilt binaries
 - D. None of the above

15. Which of these represents system programs?
 - A. Libraries
 - B. Device drivers
 - C. Servers
 - D. All of the above

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. D | 2. C | 3. D | 4. D | 5. D |
| 6. A | 7. A | 8. D | 9. B | 10. B |
| 11. C | 12. C | 13. A | 14. C | 15. D |

Review Questions:

1. What is an operating system? Explain its main parts.
2. What are the features of Linux? Explain.
3. What are proprietary and generic operating systems? Explain why Linux is popular with hardware companies and developers?
4. What is so good about Linux? Explain about its applications, peripherals, platforms and standards.
5. What is kernel programming interface? Explain.
6. What are the basic utilities in Linux? Explain.



Further Readings

Mark G Sobell, A Practical Guide to Linux Commands, Editors, and Shell Programming, Second Edition.



Web Links

<https://www.redhat.com/en/topics/linux/what-is-linux>

Unit 02: Installation Guide

CONTENTS

Objectives

Introduction:

- 2.1 Booting sequence:
- 2.2 Download Linux
- 2.3 Installation of Linux
- 2.4 Moving around the Desktop
- 2.5 Components of Desktop

Summary:

Keywords:

Self Assessment

Answers for Self Assessment

Review Questions:

Further Readings

Objectives

After studying this unit, you will be able to

- Understand the booting process
- Understand the installation process of Linux
- Understand the partitioning of hard drives
- Understand the file system types
- Understand how to log in the system

Introduction: An operating system (OS) is a system software that manages computer hardware, software resources, and provides common services for computer programs. Booting is a bootstrapping process that starts operating systems when the user turns on a computer system. A boot sequence is the set of operations the computer performs when it is switched on that load an operating system.

2.1 Booting sequence:

The booting sequence follows several steps. These are:

- Turn on the system
- CPU jump to address of BIOS
- BIOS runs POST (Power-On Self-Test)
- Find bootable devices
- Loads and execute boot sector from MBR
- Load OS

Linux and Shell Scripting

The first and foremost task is to turn on the system. So that the other processes can start. Rest of the process includes jumping of CPU to the address of CPU, BIOS runs POST, finding of bootable devices, loads and execute boot sector from MBR and loading of operating system.



BIOS (Basic Input/ Output System)

BIOS refer to the software code run by a computer when first powered on. It identifies your computer's hardware, configures it, tests it, and connects it to the operating system for further instruction. This is called the boot process. The primary function of BIOS is code program embedded on a chip that recognizes and controls various devices that make up the computer.

MBR(Master Boot Record)

OS is booted from a hard disk, where the Master Boot Record (MBR) contains the primary boot loader. The MBR is a 512-byte sector, located in the first sector on the disk (sector 1 of cylinder 0, head 0). After the MBR is loaded into RAM, the BIOS yields control to it.

Boot loader

Boot loader could be more adeptly called the kernel loader. The task at this stage is to load the Linux kernel. GRUB and LILO are the most popular Linux boot loader. Examples of boot loaders are:

 Example: GRUB, LILO, GRUB2WIN, BOOTCAMP, BOOTKEY, NTLDR and Syslinux

GRUB:

GRUB stands for GRand Unified Bootloader. It is an operating system independent boot loader. It is a multiboot software packet from GNU. It has a flexible command line interface. It has file system access. It supports multiple executable formats. It supports diskless system.

LILO: LInux Loader

This boot loader does not depend on a specific file system. It can boot from hard-disk and floppy

Task of kernel

The kernel helps in process management, memory management. The device management is also one of the tasks of kernel. The system calls are also handled by kernel.

Init process

The first thing the kernel does is to execute init program. Init is the root/parent of all processes executing on Linux. The first processes that init starts is a script /etc/rc.d/rc.sysinit. Based on the appropriate run-level, scripts are executed to start various processes to run the system and make it functional. The init process is identified by process id "1". Init is responsible for starting system processes as defined in the /etc/inittab file.Upon shutdown, init controls the sequence and processes for shutdown.

Runlevels

A run-level is a software configuration of the system which allows only a selected group of processes to exist. Init can be in one of seven run-levels: 0-6.

Runlevel	Scripts Directory (RedHat/Fedora Core)	State
0	/etc/rc.d/rc0.d/	shutdown/halt system
1	/etc/rc.d/rc1.d/	Single user mode
2	/etc/rc.d/rc2.d/	Multiuser with no network services exported
3	/etc/rc.d/rc3.d/	Default text/console only start. Full multiuser
4	/etc/rc.d/rc4.d/	Reserved for local use. Also X-windows (Slackware/BSD)
5	/etc/rc.d/rc5.d/	XDM X-windows GUI mode (Redhat/System V)
6	/etc/rc.d/rc6.d/	Reboot
s or S		Single user/Maintenance mode (Slackware)
M		Multiuser mode (Slackware)

2.2 Download Linux

- To install Red Hat, you will need to download the ISO images (CD Images) of the installation CD-ROMs from <http://fedora.redhat.com>
- Download the i386 images for 32 Intel Processors, PPC images for Apple Macintosh and x86_64 for 64-bit AMD Processors.

Linux and Shell Scripting

Here the installation of Linux distribution is shown under VMWare. It can be installed as an individual operating system as well. Generally, when we buy a new system, the seller gives us the Windows operating system by default in that.

2.3 Installation of Linux

New Virtual Machine Wizard:

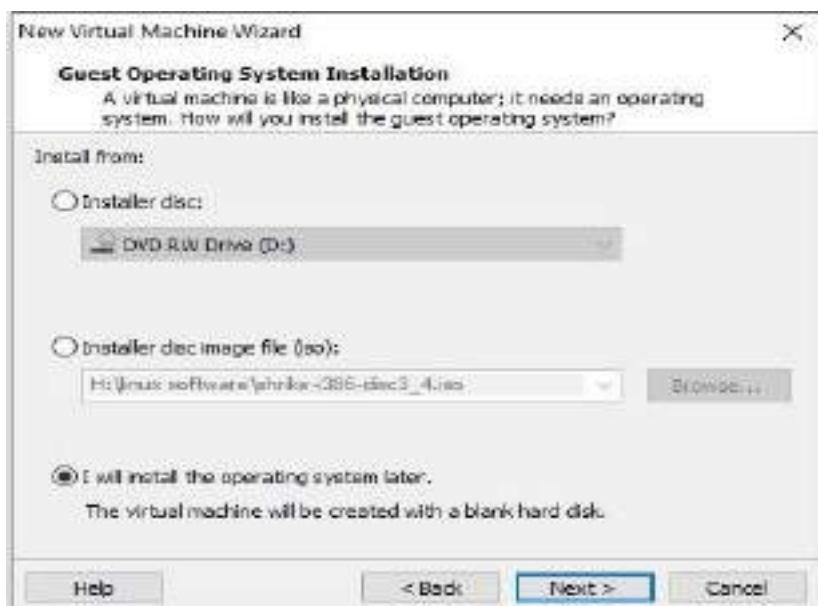
After the installation of VMWare in the system, the New Virtual Machine Wizard will open. It asks for the type of configuration you want to go with: typical or custom. The typical configuration is the recommended one and custom is the configuration with the advanced options. Choose the appropriate option and go to next step.



Installation of operating system

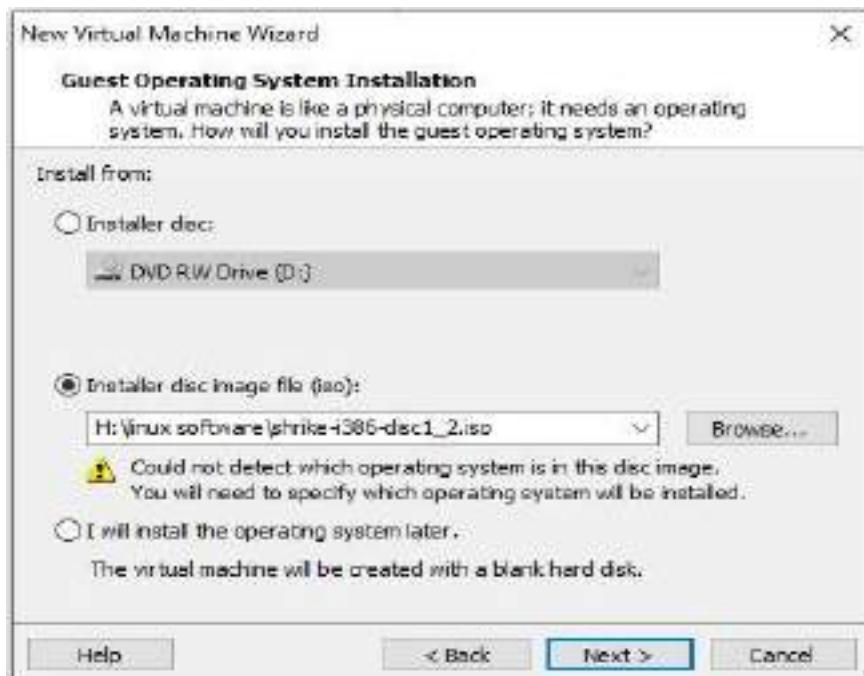
In this step, the guest operating system will be installed. Here the available options are

- Installer disc
- Installer disc image file (iso)
- Later installation.

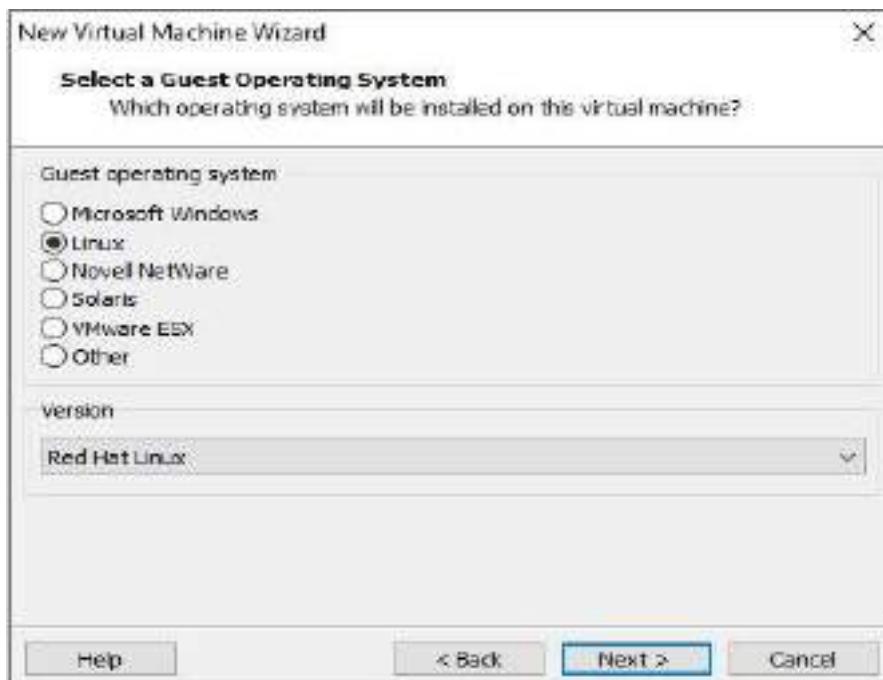


Unit 02: Installation Guide

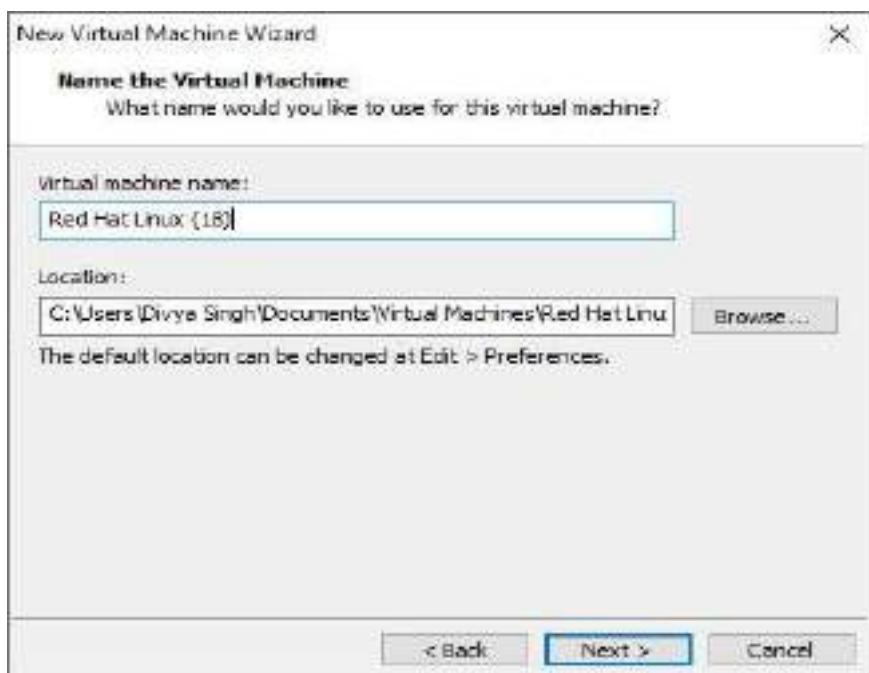
So out of these three options, choose the option depending upon the availability of the operating system. Here the installation using iso images is shown. The whole operating system is divided into three iso images.



After selecting the first iso image, choose which guest operating system you want to install. Here the Linux is chosen for installation on the virtual machine. The version of the operating system will also be selected here. Once it is done, click on Next to go further.



Write the name of the virtual machine and choose the location where you want to have all of its files. Once this is done, click on next to go further.



Specify disk capacity

When the name and location is specified, next task is to specify the disk capacity. By default, the recommended maximum disk size is 8 GB. We can increase or decrease it as per our requirements. Then it will ask whether you want to store virtual disk as a single file or into multiple files. Choose the appropriate option and go further.



After this, there are two options. One is to power on this virtual machine and other is to edit virtual machine settings. If you have done any mistake while setting the machine, then edit those settings. Otherwise turn on the power machine.



When you power on the virtual machine, the next screen will be:

- To install or upgrade Red Hat Linux in graphical mode, press the <ENTER> key.
- To install or upgrade Red Hat Linux in text mode, type: linux text <ENTER>.
- Use the function keys listed below for more information.

[F1-Main] [F2-Options] [F3-General] [F4-Kernel] [F5-Rescue]
boot: _

When you press the ENTER key, the next screen will be:

```
hid-core.c: USB HID support drivers
mice: PS/2 mouse device common for all mice
sd: sd driver 0.98.0 MAX_MIDI=256, MD_SBB=256
sd: Autodetecting RAID arrays.
sd: autofs ...
sd: ... autofs DONE.
NET4: Linux TCP/IP 1.8 for NET4.0
IP Protocols: ICMP, UDP, TCP
IP: routing cache hash table of 2896 buckets, 16Kbytes
TCP: Hash tables configured (established 16384 bind 32768)
NET5: Unix domain sockets 1.8/SMP for Linux NET4.0,
RHDISK: Compressed Image found at block 0
Pressing initrd memory: 2648k freed
EXT2-fs warning: checking forced, running e2fsck is recommended.
UFS: Mounted root (ext2 filesystem).
Greetings.
Red Hat install init version 9.0 starting
mounting /proc filesystem... done
mounting /dev/sda1 (ext3 64M) Filesystem... done
checking for NFS root filesystem... no
trying to remount root filesystem read/write... done
checking for writeable /tmp... yes
running install...
running /sbin/loader
```

Testing the CD Media

If you are using the CD media for the first time, then it is advised to test it before the installation process. If the same media is used for installation earlier, then it can be skipped.



Running Anaconda

The Anaconda starts running. It is best to install Anaconda for the local user, which does not require administrator permissions and is the most robust type of installation. However, if you need to, you can install Anaconda system wide, which does require administrator permissions.



The process starts with the screen "Welcome to Red Hat Linux". Click on Next for further process.



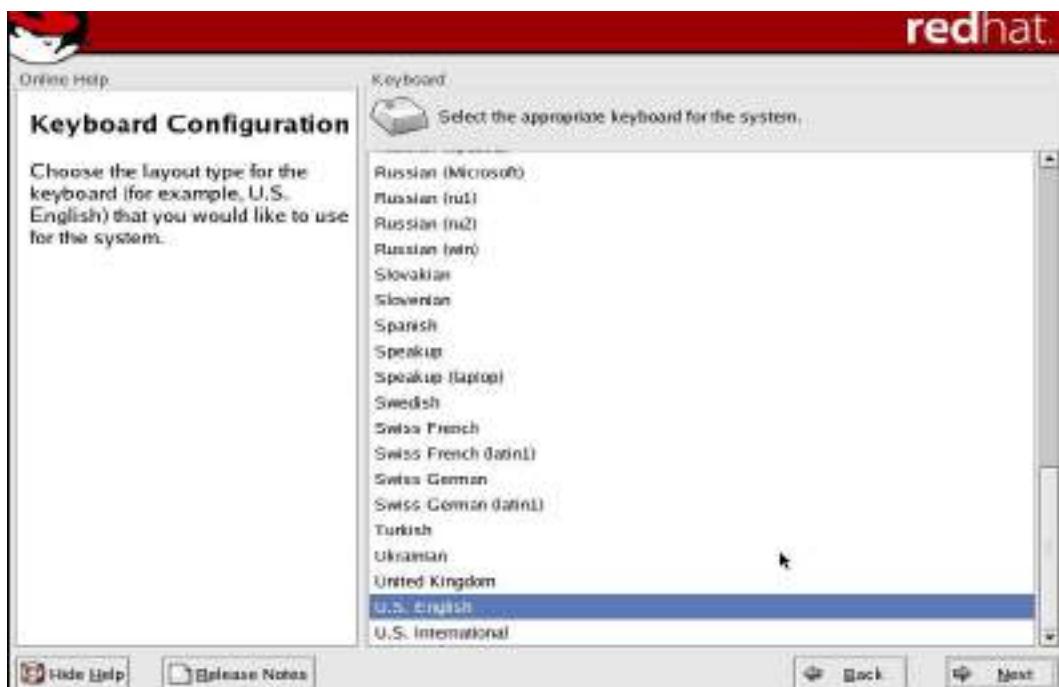
Language Selection

The next screen is for language selection. Choose the language that you want to use during installation process. After selection of language, click on next.



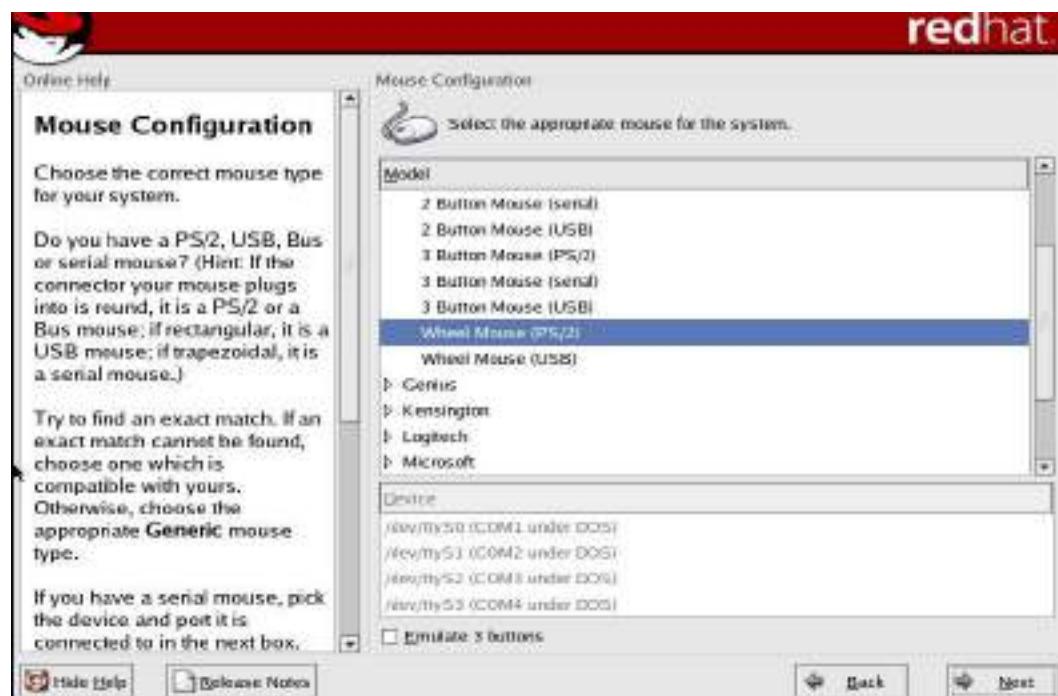
Keyboard Configuration

The screen is for choosing of layout type of keyboard that we want to use for the system. After chosen, click on next.



Mouse Configuration

The screen is for mouse configuration. By default, it will choose the appropriate option. So, it is advised to go with the recommended settings. Click on next.



Installation Type

Next is to choose the installation type which best meet your needs. The installation type available are: Personal Desktop, workstation, server and custom. Choose the appropriate option and click on next.



Disk Partitioning Setup

Partitioning is a means to divide a single hard drive into many logical drives. A partition is a contiguous set of blocks on a drive that are treated as an independent disk. A partition table is an index that relates sections of the hard drive to partitions.

The task is now to partition the drive. It can be done in two ways: automatically partition and manually partition with disk druid. The automatically partition is dependent upon the selected installation type. The manual partition uses the tool Disk Druid for partitioning. The ppartition fields are:

- **Device:** This field displays the partition's device name.
- **Start:** This field shows the sector on your hard drive where the partition begins.
- **End:** This field shows the sector on your hard drive where the partition ends.
- **Size:** This field shows the partition's size (in MB).
- **Type:** This field shows the partition's type (for example, ext2, ext3, or vfat).
- **Mount Point:** A mount point is the location within the directory hierarchy at which a volume exists; the volume is "mounted" at this location. This field indicates where the partition will be mounted.

The file system types are:

- **ext2** — An ext2 filesystem supports standard Unix file types (regular files, directories, symbolic links, etc). It provides the ability to assign long file names, up to 255 characters. Versions prior to Red Hat Linux 7.2 used ext2 file systems by default.
- **ext3** — The ext3 filesystem is based on the ext2 filesystem and has one main advantage — journaling. Using a journaling filesystem reduces time spent recovering a filesystem after a crash as there is no need to fsck the filesystem.
- **swap** — Swap partitions are used to support virtual memory. In other words, data is written to a swap partition when there is not enough RAM to store the data your system is processing.
- **vfat** — The VFAT filesystem is a Linux filesystem that is compatible with Windows 95/NT long filenames on the FAT filesystem.

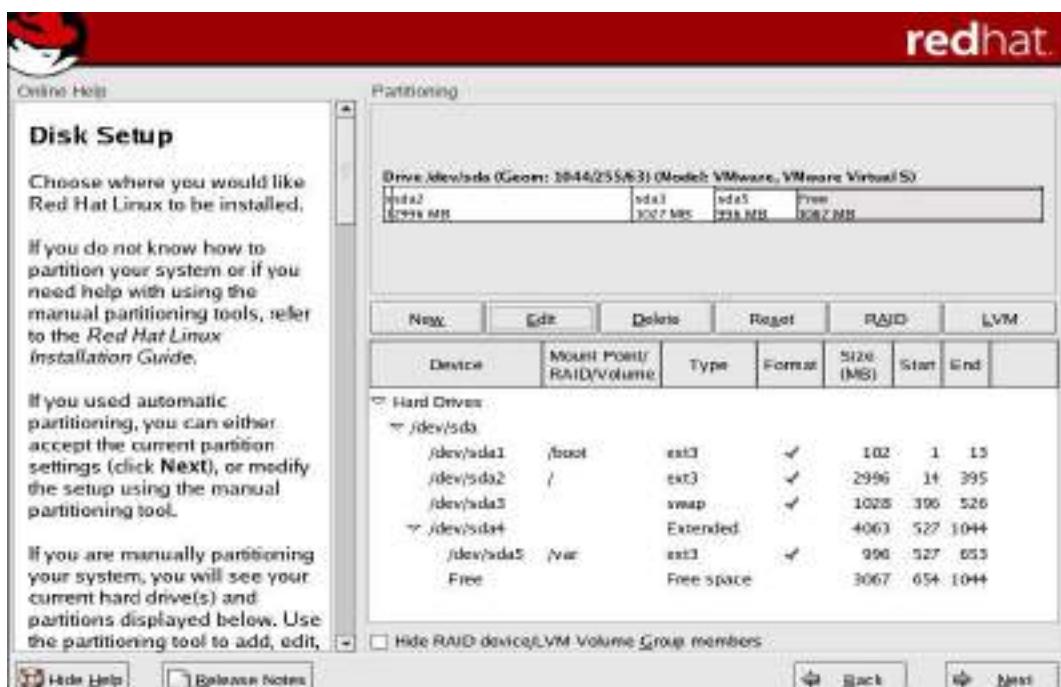


Recommended Partitioning Scheme

- Unless you have a reason for doing otherwise, it is recommended that you create the following partitions:
- /boot partition – contains kernel images and grub configuration and commands
- / partition
- /var partition
- Any other partition based on application (e.g /usr/local for squid)
- swap partition – swap partitions are used to support virtual memory. In other words, data is written to a swap partition when there is not enough RAM to store the data your system is processing. The size of your swap partition should be equal to twice your computer's RAM.

Here we are going with manual partition in which disk druid tool helps.

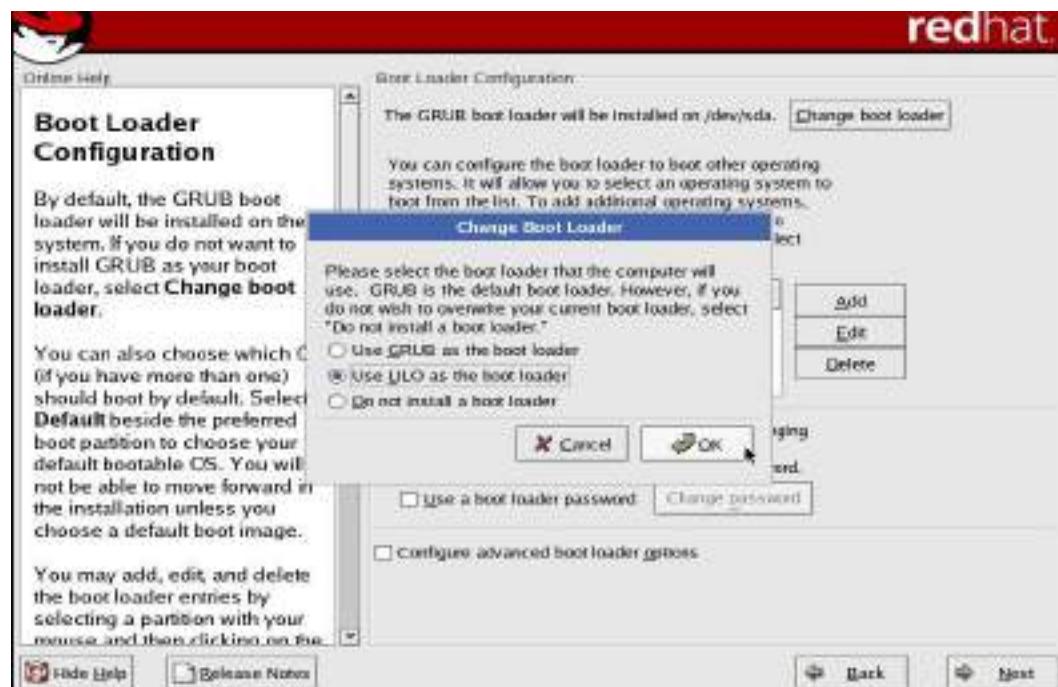
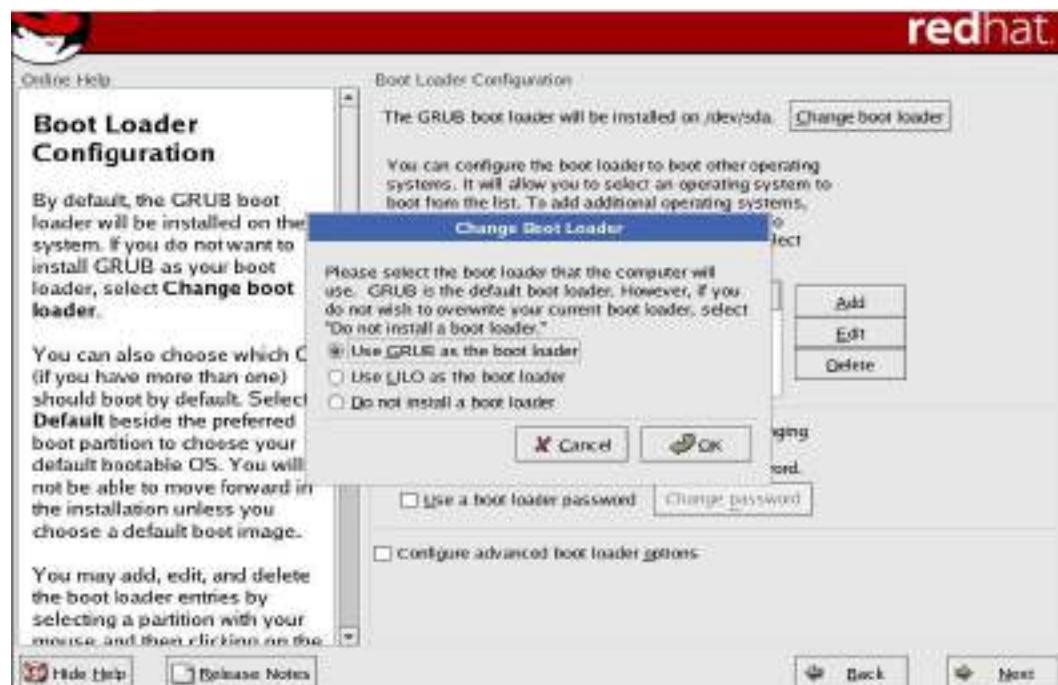




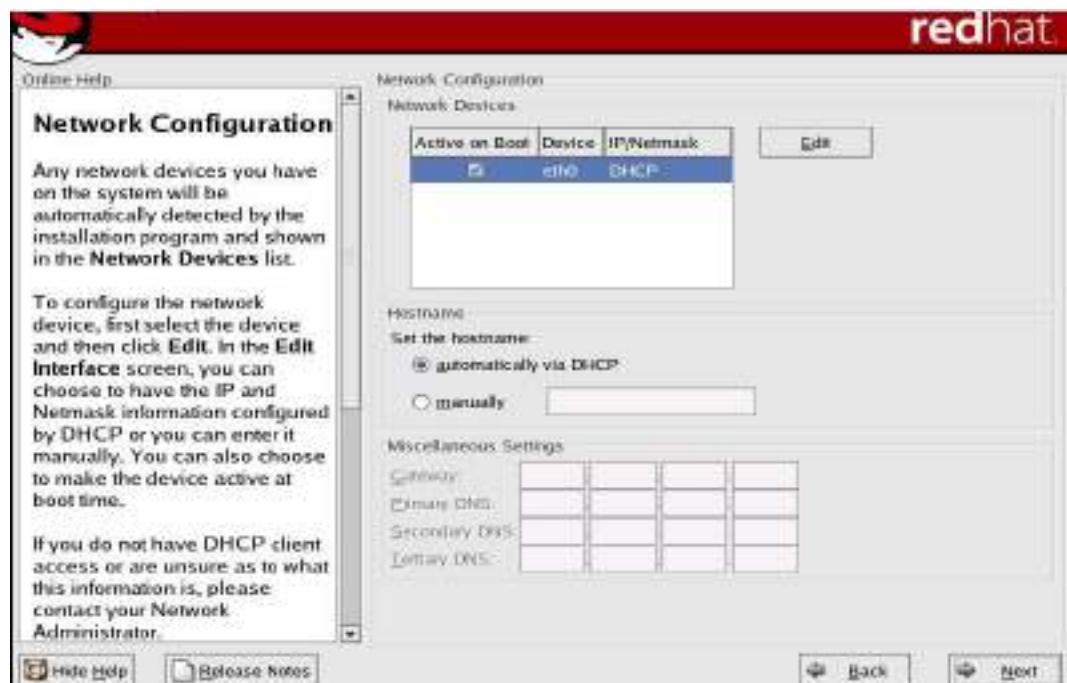
Boot Loader Configuration



Change Boot Loader



Network Configuration



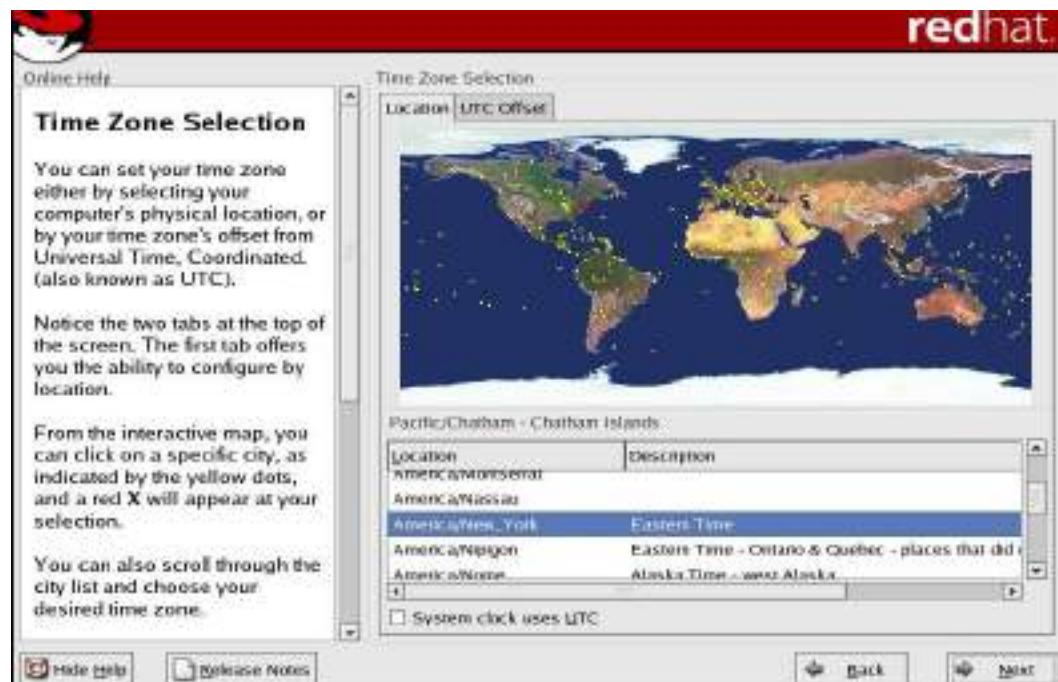
Firewall Configuration



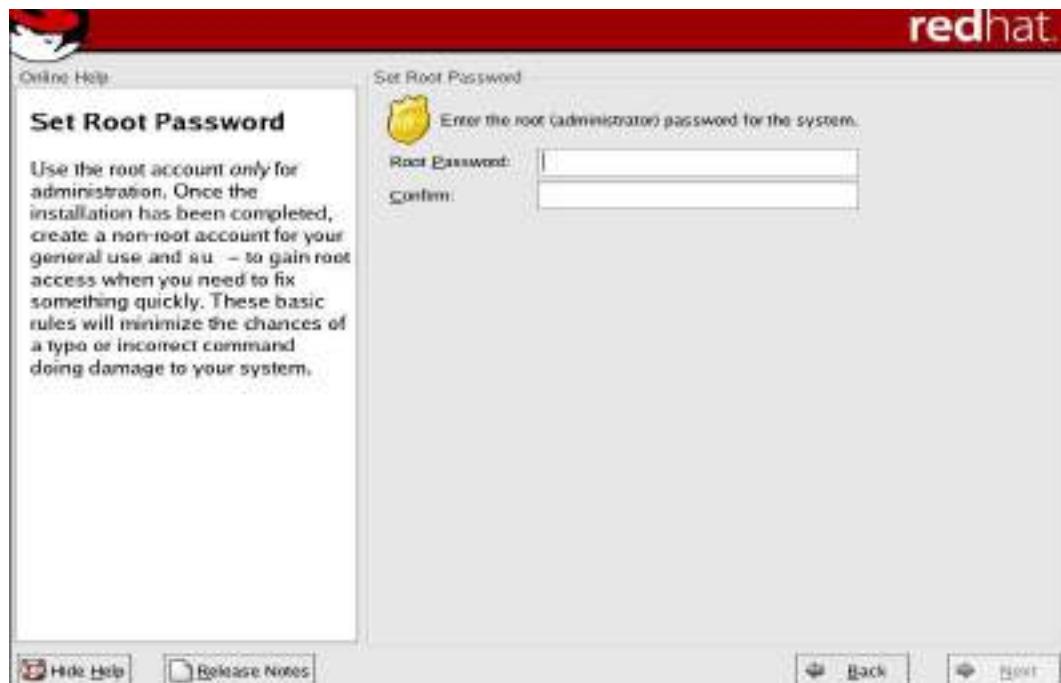
Additional Language Support



Time Zone Selection



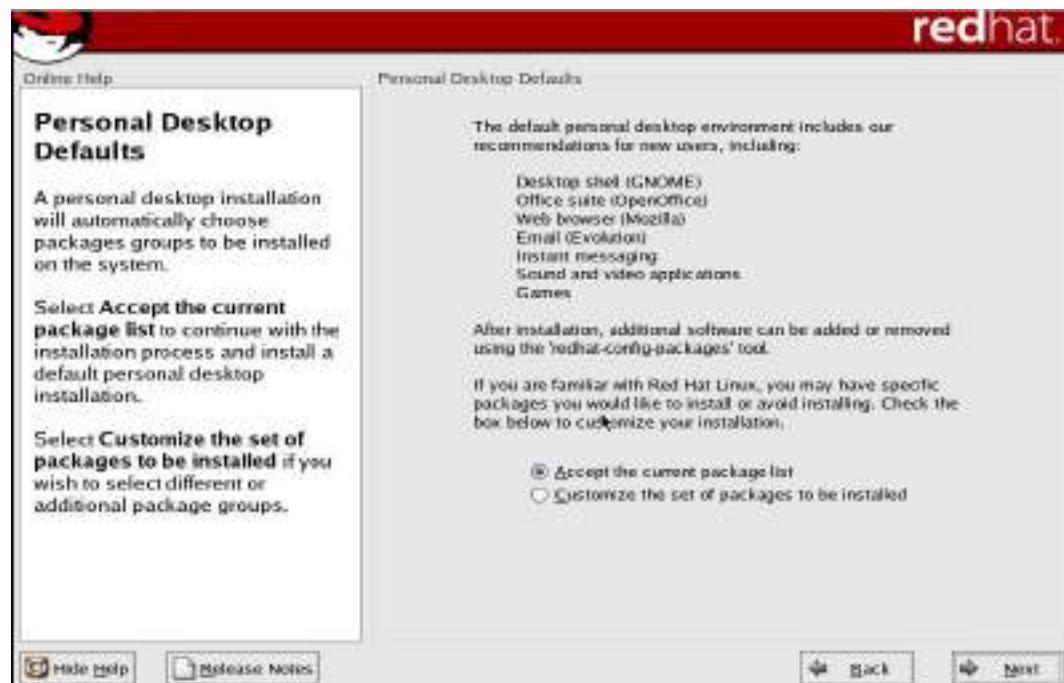
Set Root Password



Provide Root Password



Personal Desktop Defaults (Packages)



About to Install

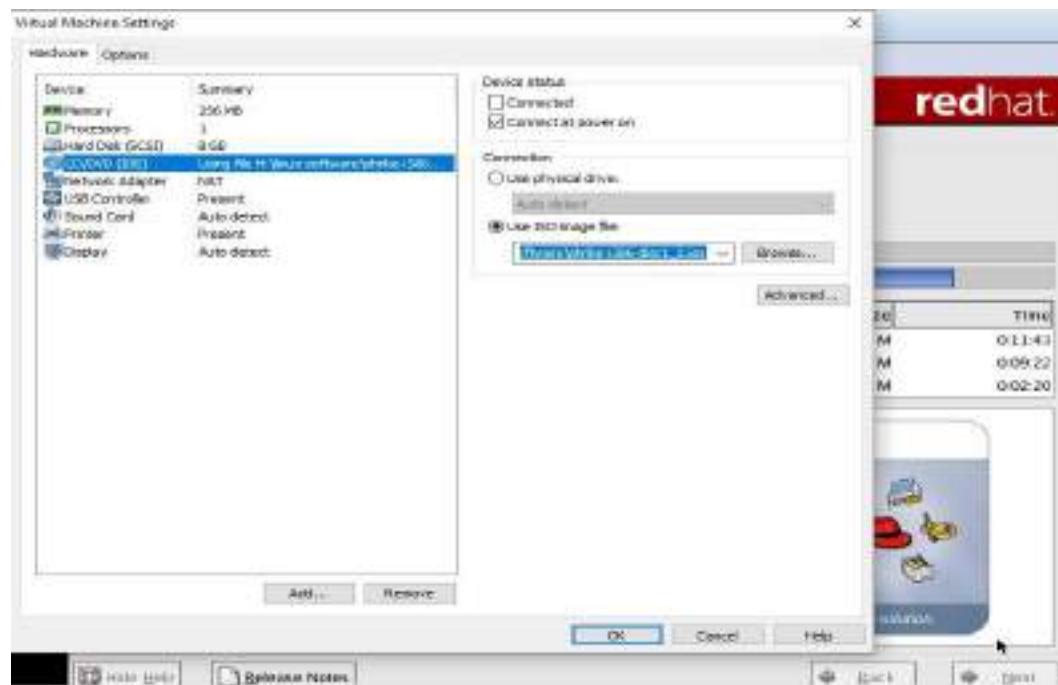
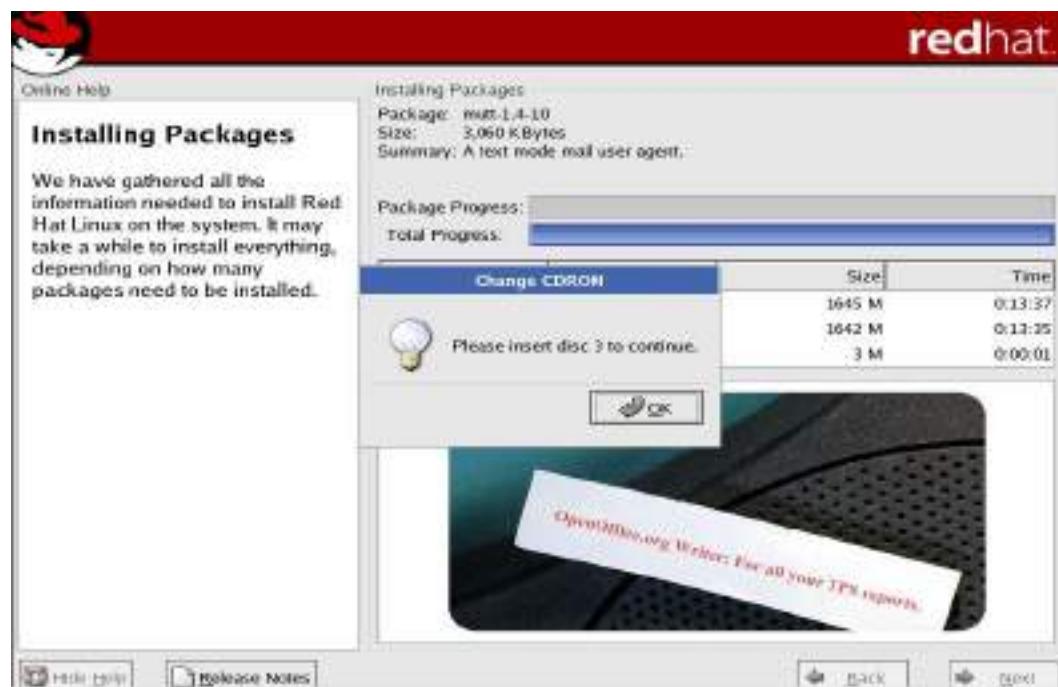


Installing Packages

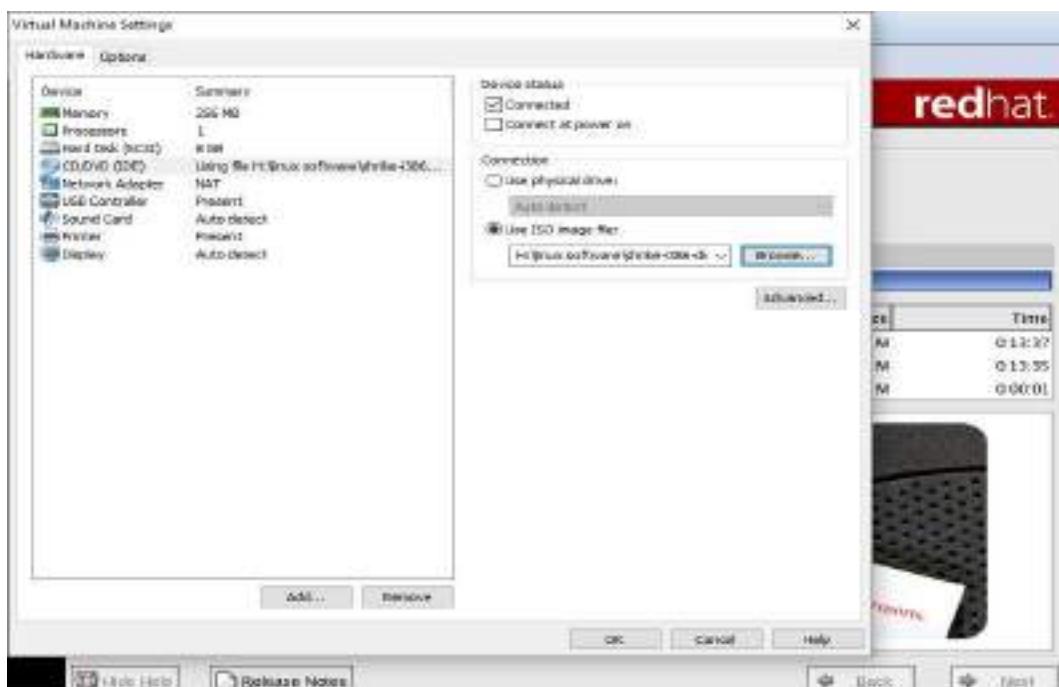


Insert Disc 2



Linux and Shell Scripting**Insert Disc 3**

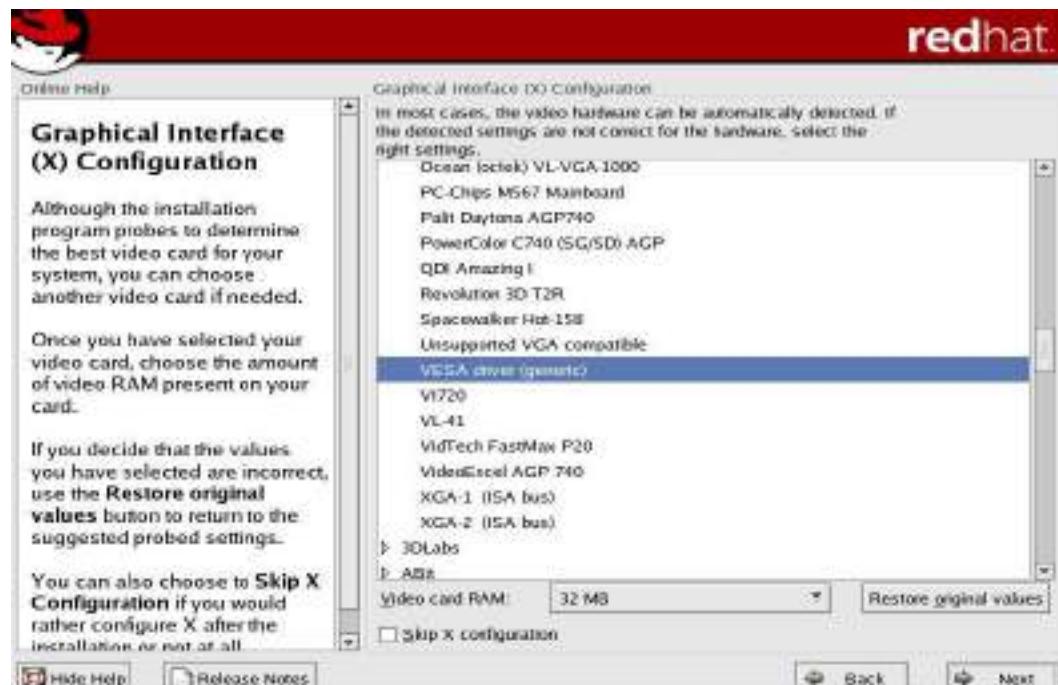
Proper Settings



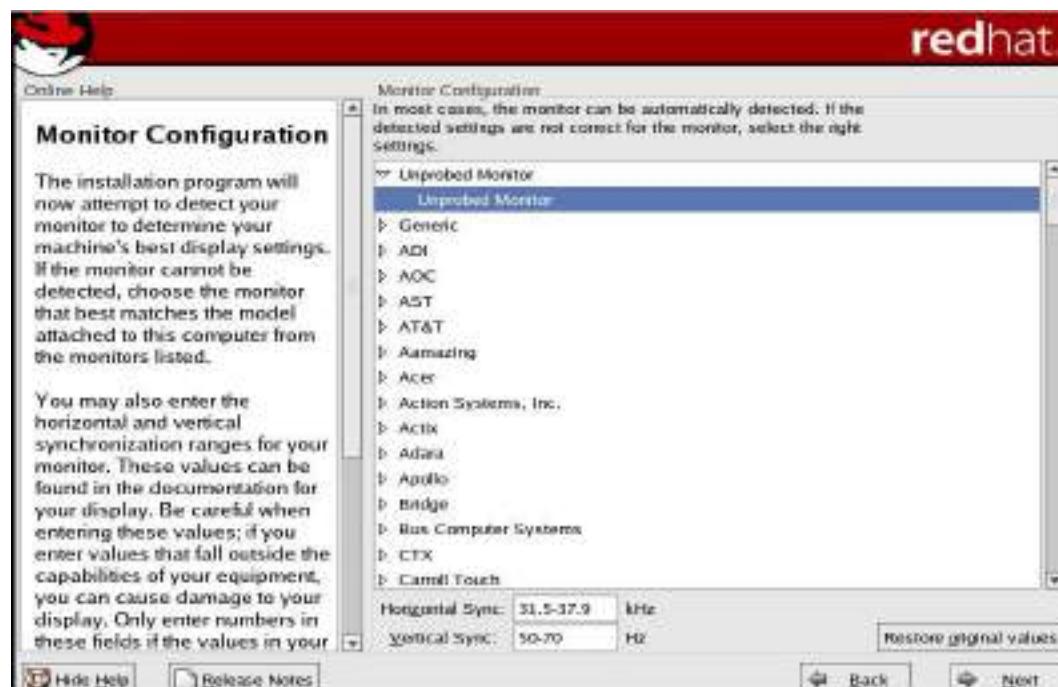
Boot Diskette Creation



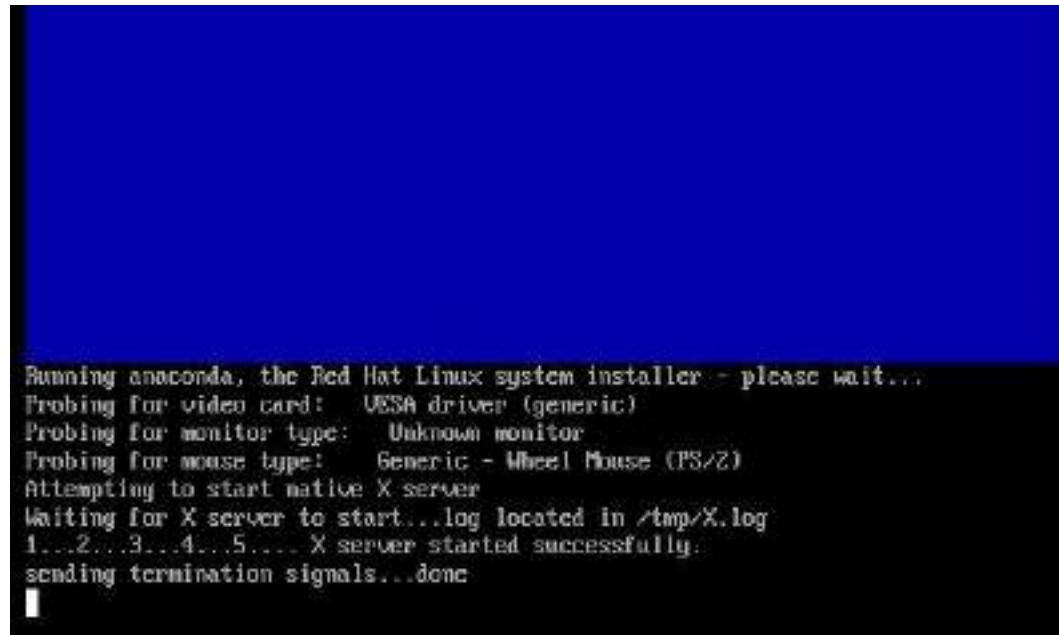
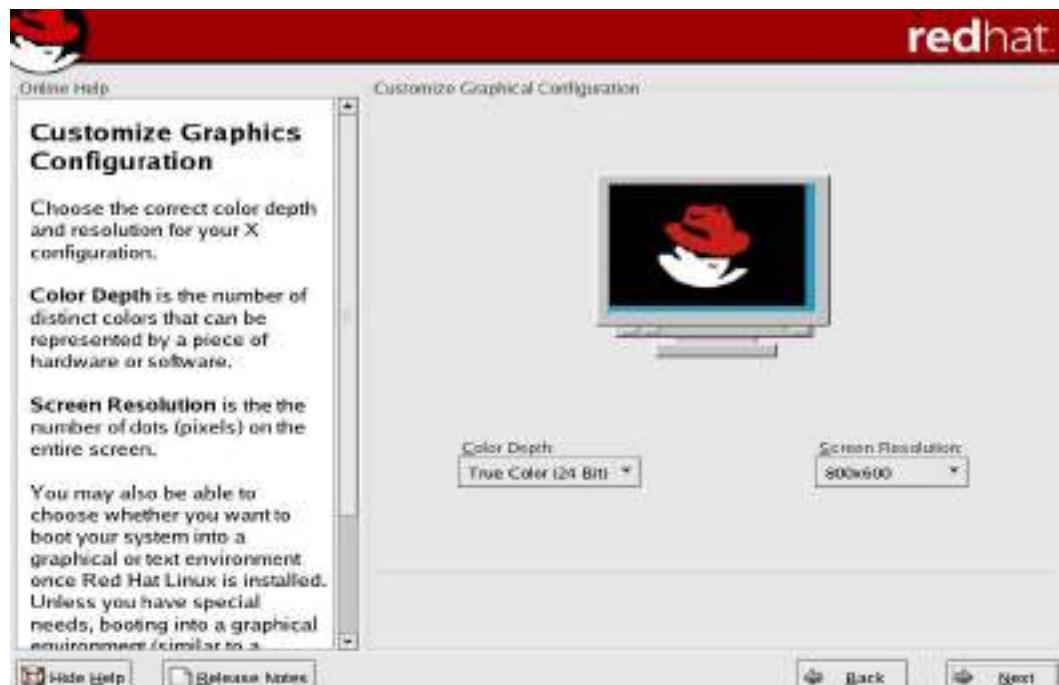
Graphical Interface (X) Configuration



Monitor Configuration



Customize Graphics Configuration

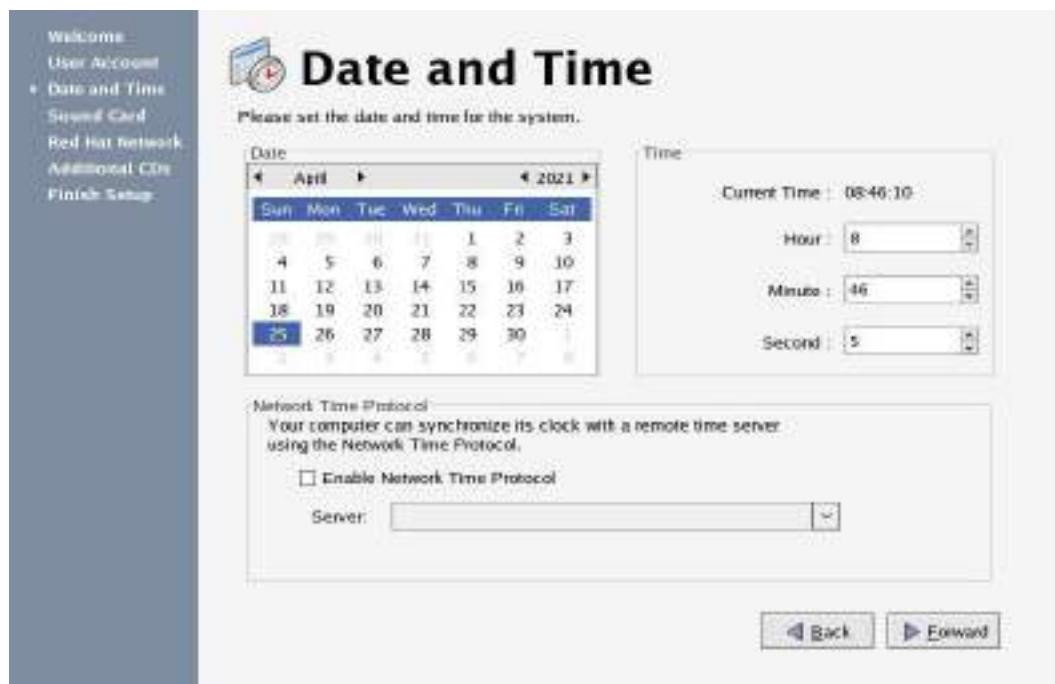


Welcome Screen**Creation of User Account**

Enter Credentials



Enter Date and Time



Sound Card**Registration with Red Hat Network**

Installation of packages from additional CDs**Finish Setup**

Logging in



Home Screen



2.4 Moving around the Desktop

- **GNOME:** The default desktop interface of Red Hat Linux 9. GNOME represents a presentation layer that provides a graphical user interface as well as the focused working environment, which enables you to access all your work from one place.
- **KDE:** KDE desktop is included in Red Hat Linux 9 distribution but not installed by default. KDE is a desktop environment for an integrated set of cross-platform applications designed to run on Linux, FreeBSD, Microsoft Windows, Solaris and Mac OS, designed by the KDE Community.

2.5 Components of Desktop

- **Panel:** The panel stretches across the bottom of the desktop. By default, it contains the main menu icon and quick-launch icons for logging out, opening a terminal window, and other common applications and utilities. The panel is highly configurable. You can add and remove buttons that launch applications easily.
- **Workspace:** Workspaces refer to the grouping of windows on your desktop. You can create multiple workspaces, which act like virtual desktops. Workspaces are meant to reduce clutter and make the desktop easier to navigate. Workspaces can be used to organize your work. For example, you could have all your communication windows, such as e-mail and your chat program, on one workspace, and the work you are doing on a different workspace. Your music manager could be on a third workspace.

Summary:

- A boot sequence is the set of operations the computer performs when it is switched on that load an operating system.
- The primary function of BIOS is code program embedded on a chip that recognizes and controls various devices that make up the computer.
- GRUB and LILO are the most popular Linux boot loader.
- The first thing the kernel does is to execute init program.
- There are four types of installation available in Linux: personal desktop, workstation, server and custom.
- There are two ways the disk can be partitioned: manually and automatically.

Keywords:

- **Booting:** It is a bootstrapping process that starts operating systems when the user turns on a computer system.
- **BIOS:** It refers to the software code run by a computer when first powered on.
- **Partitioning:** It is a means to divide a single hard drive into many logical drives. A partition is a contiguous set of blocks on a drive that are treated as an independent disk.
- **Swap:** Swap partitions are used to support virtual memory. In other words, data is written to a swap partition when there is not enough RAM to store the data your system is processing.
- **GNOME:** The default desktop interface of Red Hat Linux is GNOME.
- **Run-level:** A run-level is a software configuration of the system which allows only a selected group of processes to exist

Self Assessment

1. Which of these executes kernel?
 - A. MBR
 - B. BIOS
 - C. GRUB
 - D. Init

2. The first thing a Kernel does is _____.
 - A. Execute GRUB
 - B. Execute LILO
 - C. Execute init program
 - D. None of the above

3. Which of these tasks are handled by Kernel?
 - A. System Call
 - B. Process Management
 - C. Device Management
 - D. All: System call, process and device management

4. While installation of Red Hat Linux in the system, it asks for _____.
 - A. Language Selection
 - B. Keyboard Configuration
 - C. Mouse Configuration
 - D. All: Language selection, keyboard and mouse configuration

5. In partition field, i.e., SIZE, the measurement unit is _____.
 - A. TB
 - B. MB
 - C. GB
 - D. KB

6. Which of these defines the runlevels?
 - A. 0-6
 - B. 1-7
 - C. 2-8
 - D. 3-9

7. Which of these partitions is used to support virtual memory?
 - A. /
 - B. /var
 - C. /boot
 - D. swap

8. Which of these is a Red Hat Linux installer?
 - A. Anaconda
 - B. GRUB
 - C. LILO
 - D. Emulator

9. Which of these programs allows us to partition the disk?
 - A. Anaconda
 - B. GRUB
 - C. Disk Druid
 - D. Disk Help

10. MBR is executed by _____.
 - A. BIOS
 - B. GRUB
 - C. Kernel
 - D. Init

11. In which mode, it is possible to install and upgrade Red Hat Linux?
 - A. Graphical
 - B. Text
 - C. Both graphical and text
 - D. None of the above

12. What does BIOS mean?
 - A. Basic Input/ Output Service
 - B. Basic Input/ Output System
 - C. Buffer Input/ Output System
 - D. Buffer Input/ Output Service

13. MBR is executed by _____.
 - A. BIOS
 - B. GRUB
 - C. Kernel
 - D. Init

14. In which mode, it is possible to install and upgrade Red Hat Linux?
 - A. Graphical
 - B. Text
 - C. Both graphical and text
 - D. None of the above

15. What type can be installed in Red Hat Linux?

- A. Personal Desktop
- B. Server
- C. Workstation
- D. All: personal desktop, server and workstation

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. C | 2. C | 3. D | 4. D | 5. B |
| 6. A | 7. D | 8. A | 9. C | 10. A |
| 11. C | 12. B | 13. A | 14. C | 15. D |

Review Questions:

1. What is booting? Explain the booting sequence in detail.
2. What is a kernel? Explain the tasks of a kernel in detail.
3. What is a partition? Write the partition fields. What is the recommended partition scheme?
4. What is a file system? Explain its types in detail.
5. What is a run-level? Explain about the run-level of in it.



Further Readings

Mark G. Sobell, A Practical Guide to Fedora and Red Hat Enterprise Linux, Fifth Edition, Pearson Education, Inc.



Web Links

<https://www.educba.com/install-linux/>

<https://phoenixnap.com/kb/linux-create-partition>

Unit 03: Connecting to Internet

CONDUCT

Objectives

Introduction

3.1 Internet Configuration Wizard

3.2 Connecting to LAN

3.3 Domain Name System

Keywords

Summary

Self Assessment

Answers for Self Assessment

Review Questions:

Further Readings

Objectives

After studying this unit, you will be able to understand

- Understand the Network Interfacing Tool
- Connect to LAN using static and dynamic addresses
- Understand DNS
- Know useful commands for configuration of system for internet
- Understand the internet connectivity in Linux

Introduction

When the installation of operating system is completed on the computer system, the next task we do is to connect it to the internet. The surfing of websites, playing online games, sending, and receiving of emails etc. is possible only through internet. For this, first we need to configure our computer system so that the connection can take place.

3.1 Internet Configuration Wizard

The internet configuration wizard can be opened by following this path: Main Menu | System Tools | Internet Configuration Wizard. Network interfacing tool is also known as the network administration tool or network configuration tool.



3.2 Connecting to LAN

A local area network (LAN) is a collection of devices connected in one physical location, such as a building, office, or home. A LAN can be small or large, ranging from a home network with one user to an enterprise network with thousands of users and devices in an office or school. A network device connected to a TCP/IP network has an IP address associated with it, such as 192.168.100.20. Using the IP address of a machine, other machines on the network can address it uniquely. A LAN comprises cables, access points, switches, routers, and other components that enable devices to connect to internal servers, web servers, and other LANs via wide area networks.

Once the network interfacing wizard is opened, then we need to select the device type. The available options are VPN connection, ethernet connection, ISDN connection, modem connection, token ring connection, wireless connection and xDSL connection. As we are going to connect to LAN, so for that we need to click on 'Ethernet connection', it will create a new ethernet connection. Once selected, click on forward.



Next it will ask for the selection of Ethernet card. The options are AMD PCnet32 and other ethernet cards. Click on AMD PCnet32 (eth0) and then click forward.



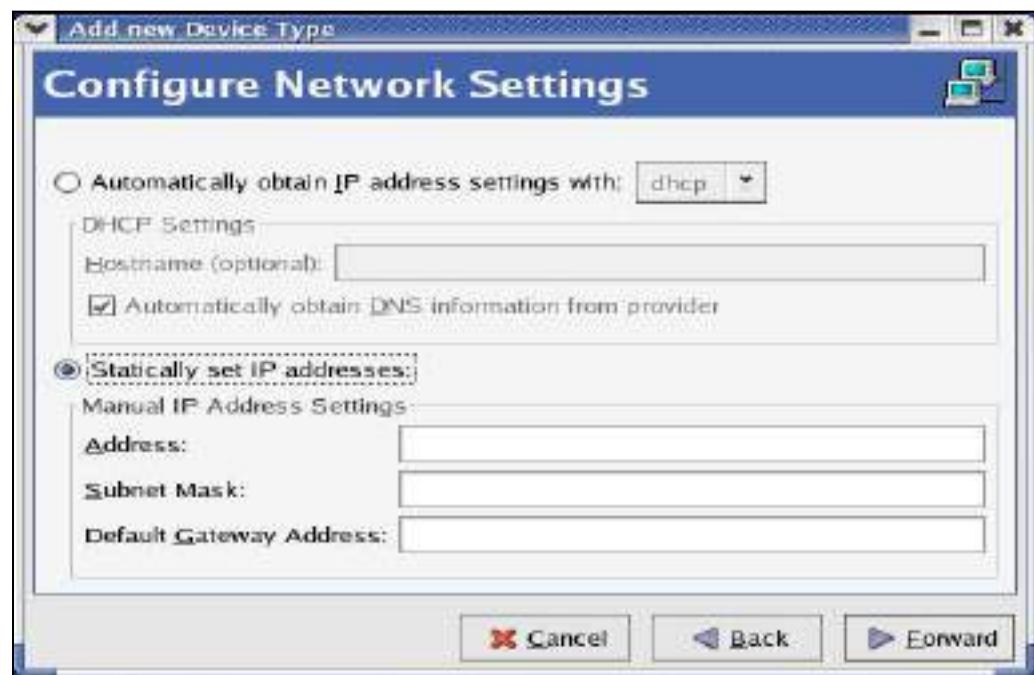
Next it will ask for configuration of network settings, and these networks are provided by the means of IP addresses. An IP address is a unique address that identifies a device on the internet or a local network. IP stands for "Internet Protocol," which is the set of rules governing the format of data sent via the internet or local network. In essence, IP addresses are the identifier that allows information to be sent between devices on a network: they contain location information and make devices

accessible for communication. The internet needs a way to differentiate between different computers, routers, and websites. IP addresses provide a way of doing so and form an essential part of how the internet works. These are of two types: static IP address and dynamic IP address.

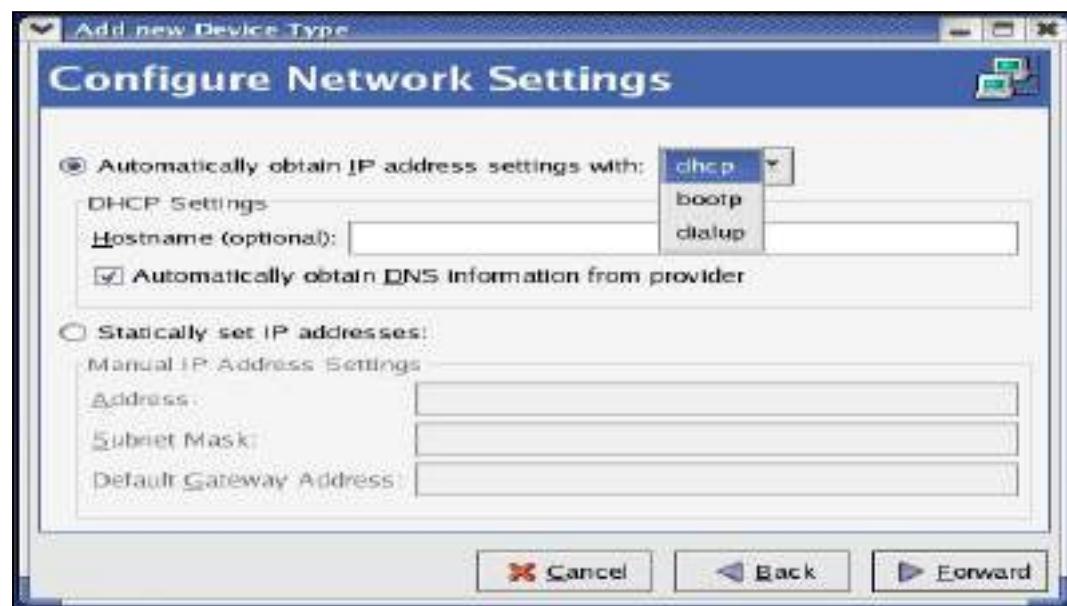
Static IP address: A static IP address is simply an address that doesn't change. Once your device is assigned a static IP address, that number typically stays the same until the device is decommissioned or your network architecture changes. Static IP addresses generally are used by servers or other important equipment. Static IP addresses are assigned by Internet Service Providers (ISPs). Your ISP may or may not allocate you a static IP address depending on the nature of your service agreement. We describe your options a little later, but for now assume that a static IP address adds to the cost of your ISP contract. A static IP address may be IPv4 or IPv6; in this case the important quality is static. Some day, every bit of networked gear we have might have a unique static IPv6 address. We're not there yet. For now, we usually use static IPv4 addresses for permanent addresses.

Dynamic IP address: As the name suggests, dynamic IP addresses are subject to change, sometimes at a moment's notice. Dynamic addresses are assigned, as needed, by Dynamic Host Configuration Protocol (DHCP) servers. We use dynamic addresses because IPv4 doesn't provide enough static IP addresses to go around. So, for example, a hotel probably has a static IP address, but each individual device within its rooms would have a dynamic IP address. On the internet, your home or office may be assigned a dynamic IP address by your ISP's DHCP server. Within your home or business network, the dynamic IP address for your devices -- whether they are personal computers, Smartphone, streaming media devices, tablet, what have you -- are probably assigned by your network router. Dynamic IP is the standard used by and for consumer equipment.

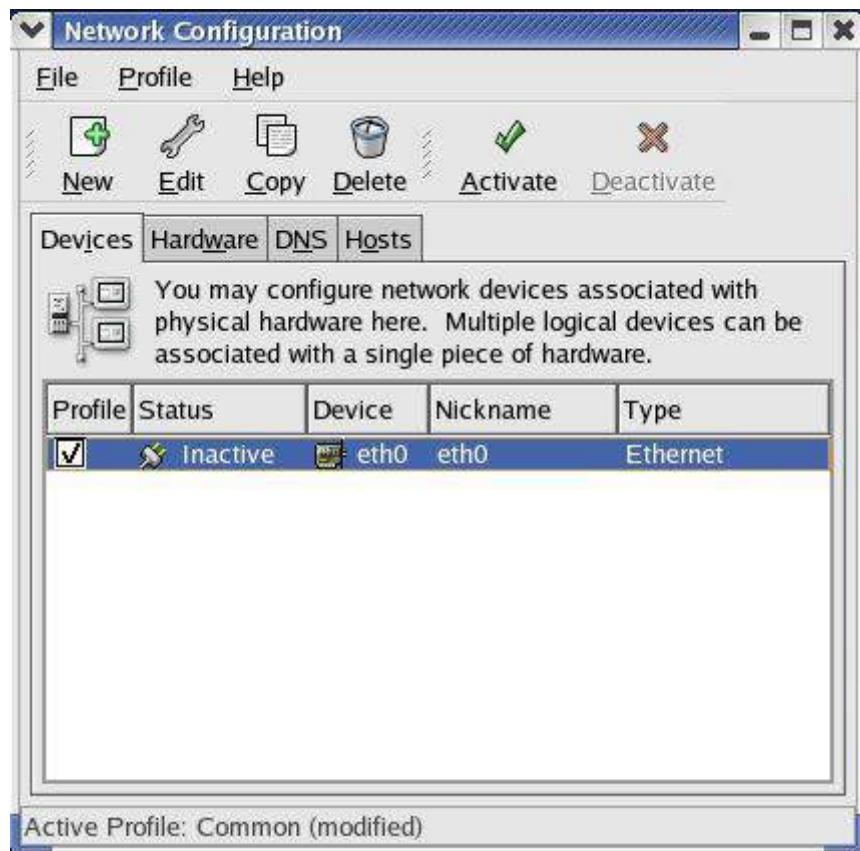
The next step is to select static or dynamic IP address. So when you go further, the next screen will show this.



Here we need to select the dynamic IP address. Dynamic IP addresses are distributed and managed via dynamic address allocation protocols. In the case of a machine connected to a LAN using a dynamic IP address, the address is allocated either using the DHCP or BOOTP. For ISPs that use PPPoE, the address is allotted by the PPPoE protocol, in which case we need to choose dialup.



When everything is done, it will create an Ethernet device.



The network interfacing tool will be opened. It has four tabs:

- Devices Tab: Lists the device connections that we have available on our machine.
- Hardware Tab: Allows us to manage the various network devices on the system, such as Ethernet cards, internal modems, and wireless cards.
- DNS Tab: Allows us to specify DNS server information.

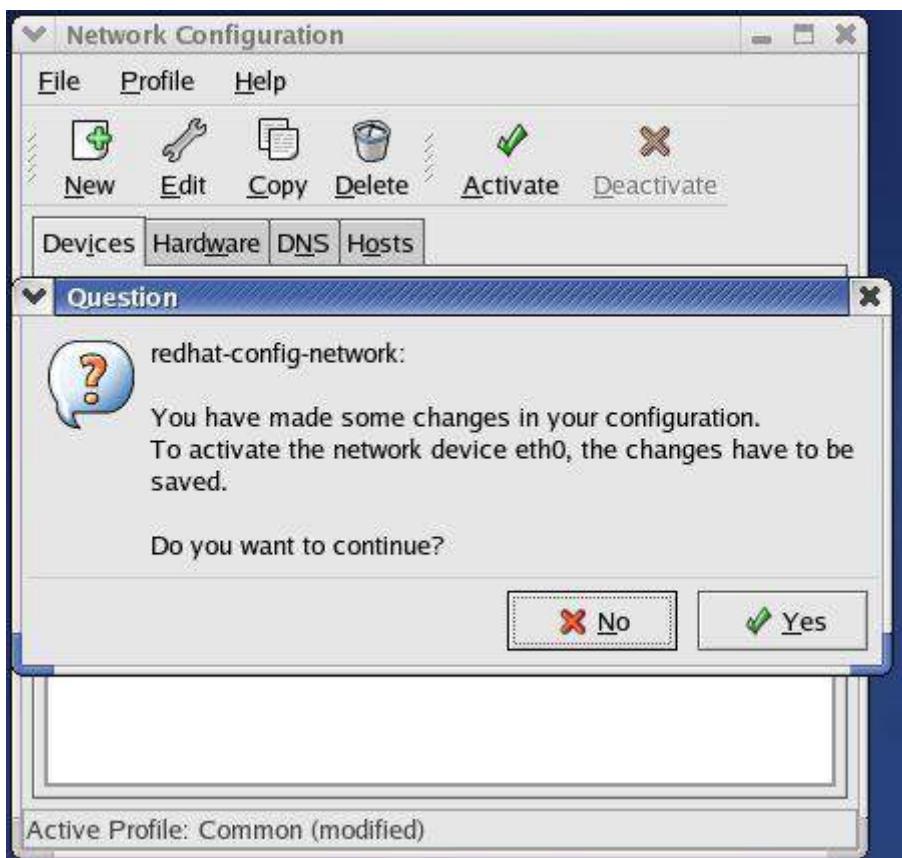
Unit 03: Connecting to Internet

- Hosts Tab: Allows us to modify the hostname of the machine and add aliases to the same host.

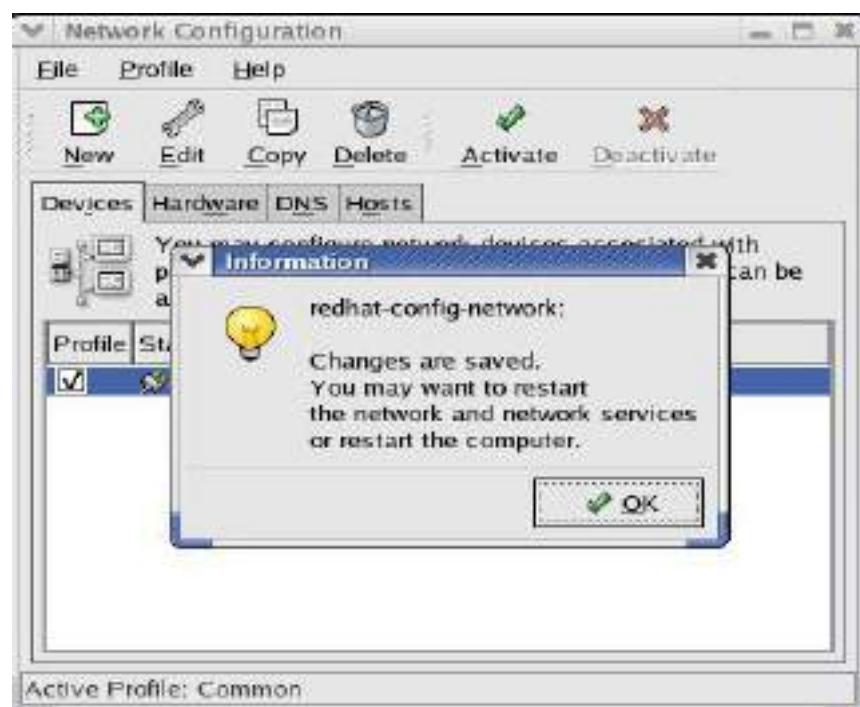
When you double click on the selected device, the next screen will look like this.



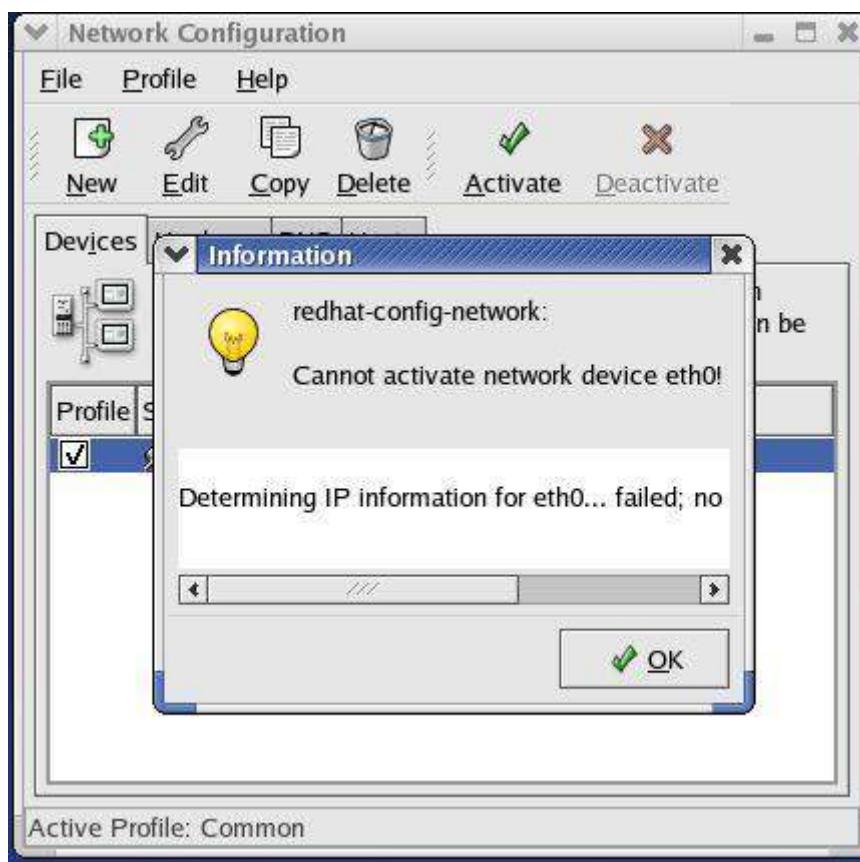
Click on allow all users to enable and disable the device. The next screen is:



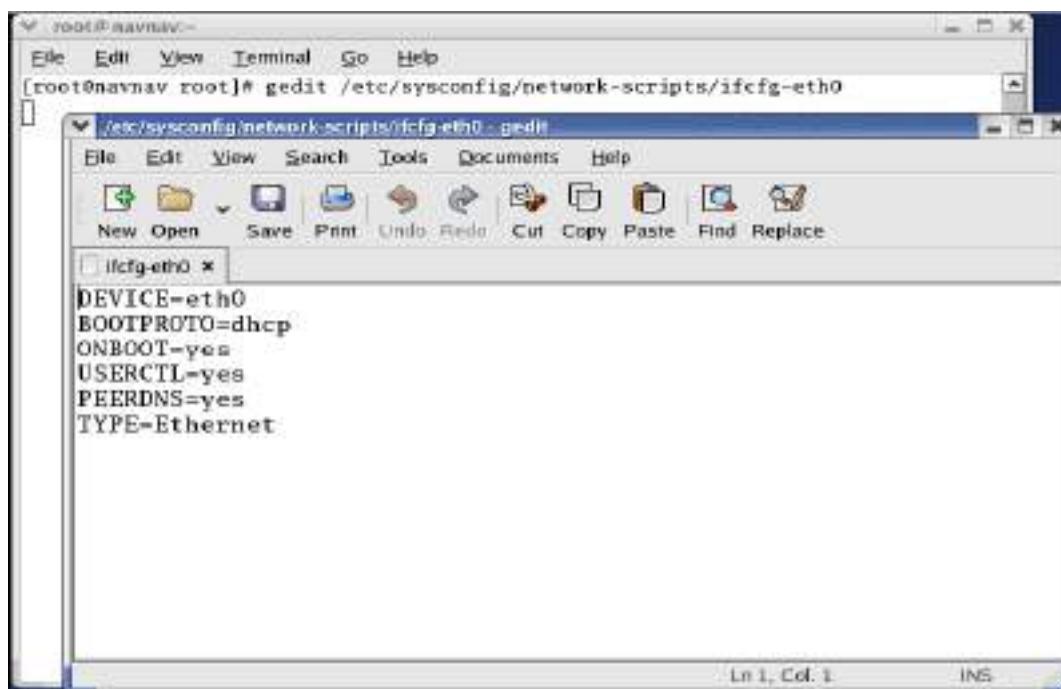
Click on OK.



To activate the internet connection, click on ok.



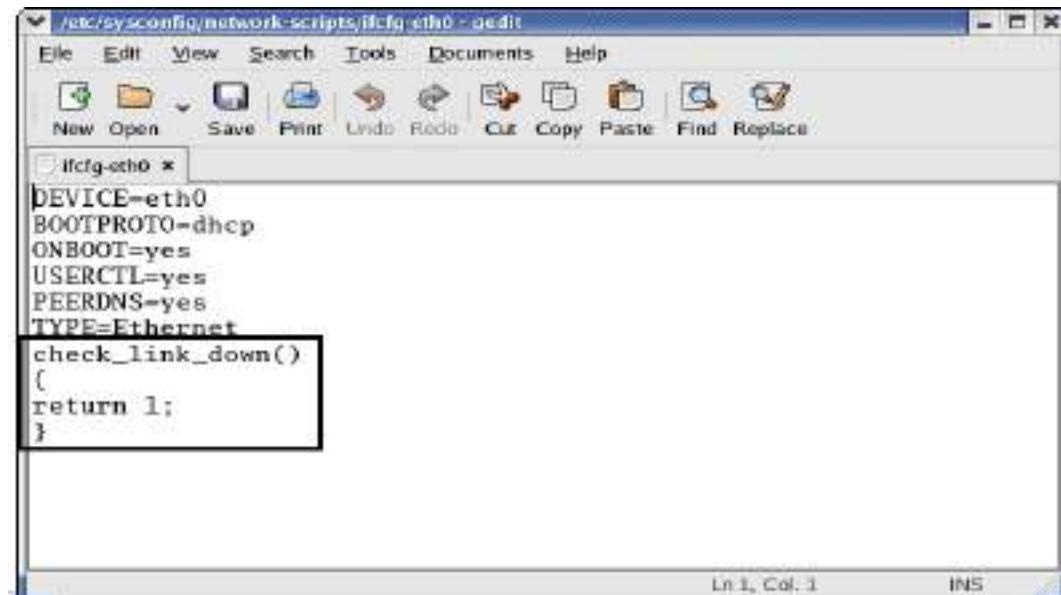
When this is done, our next task is to modify three files so that the computer system can be connected to the internet. These files are opened using the editors. For the opening of first file, we need to write #gedit /etc/sysconfig/network-scripts/ifcfg-eth0 in terminal and then press ENTER. When this file is opened, it will show



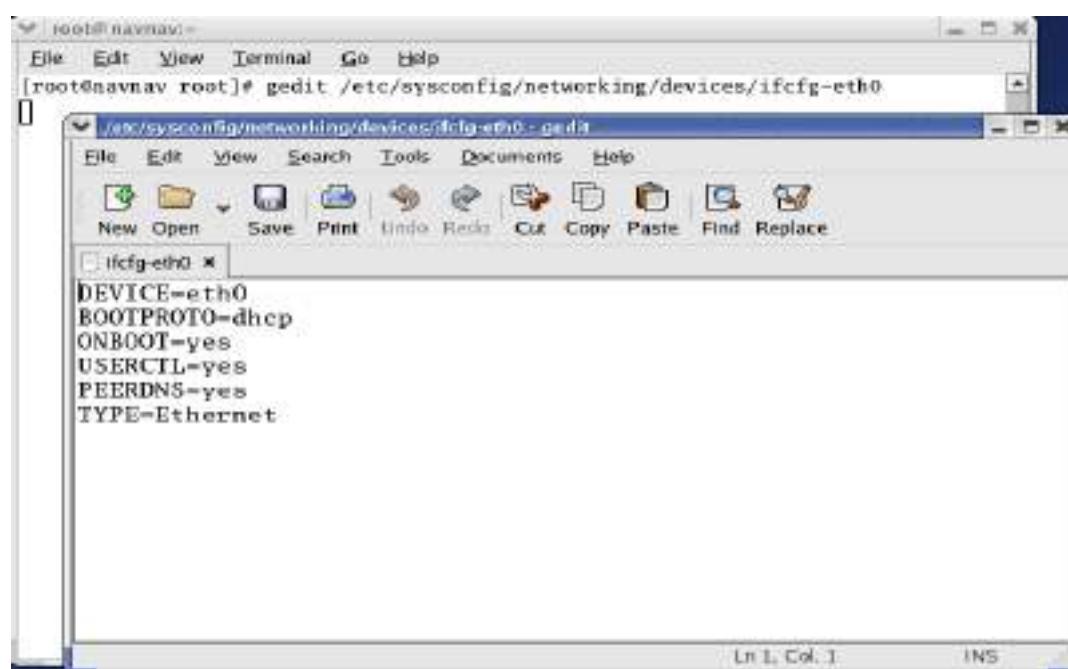
Here at the bottom of it, we need to add some content, i.e.,

```
Check_link_down()
{
return 1;
}
```

So, after modification, it will look like



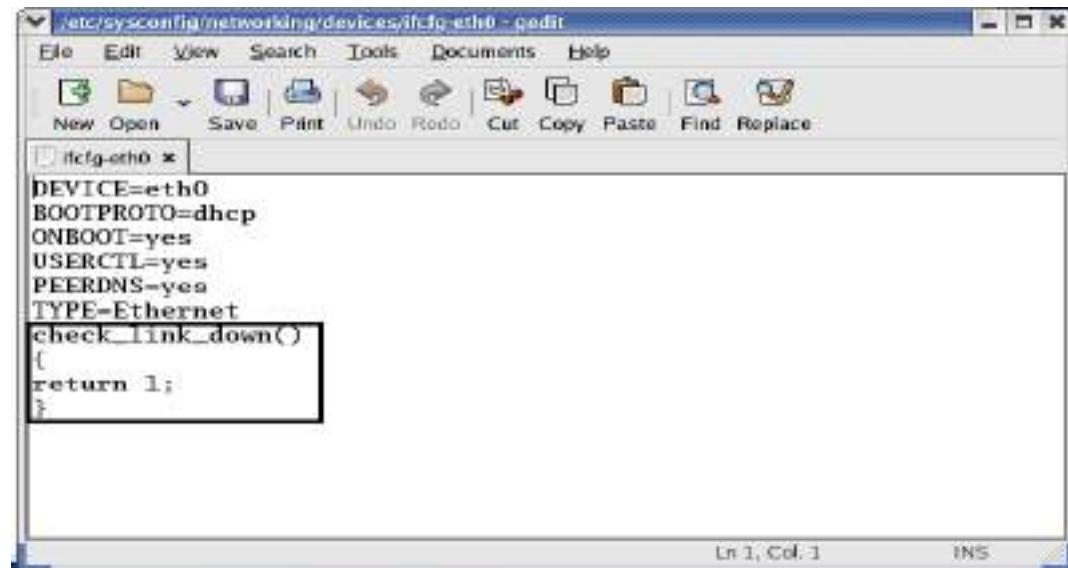
Next we need to open and modify second file: #gedit /etc/sysconfig/networking/devices/ifcfg-eth0.

Linux and Shell Scripting

Here at the bottom of it, we need to add some content, i.e.,

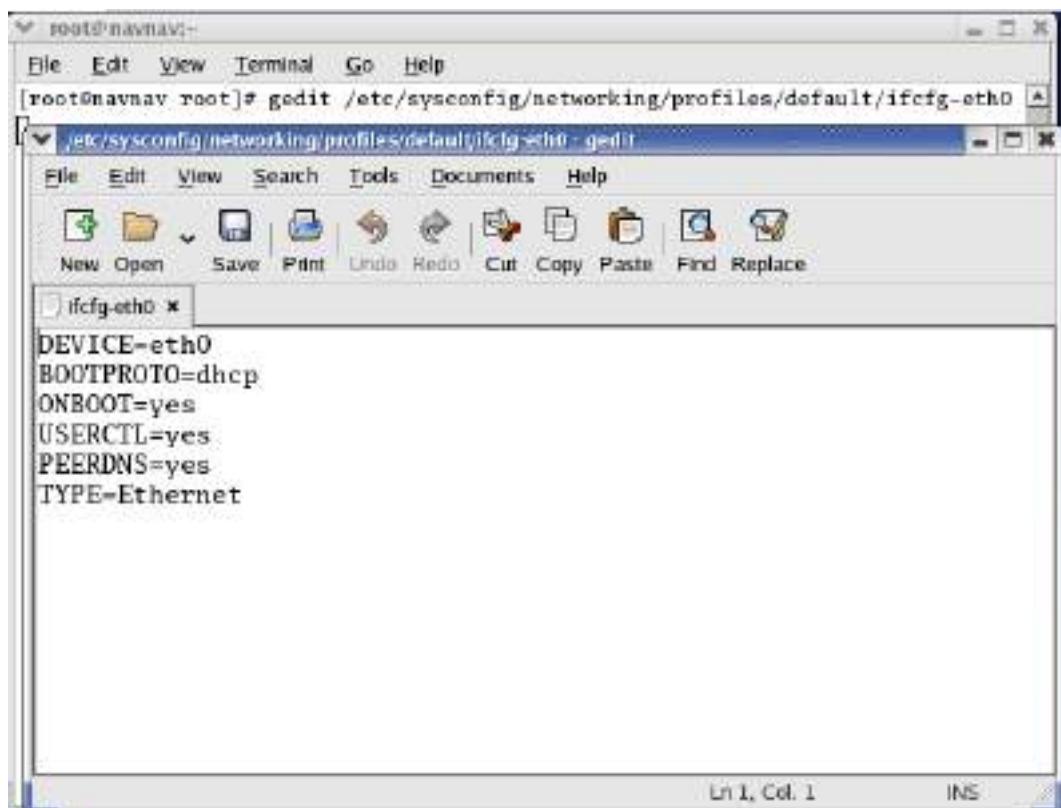
```
Check_link_down()
{
return 1;
}
```

So, after modification, it will look like



Next, we will open the third file: #gedit/etc/sysconfig/networking/profiles/default/ifcfg-eth0

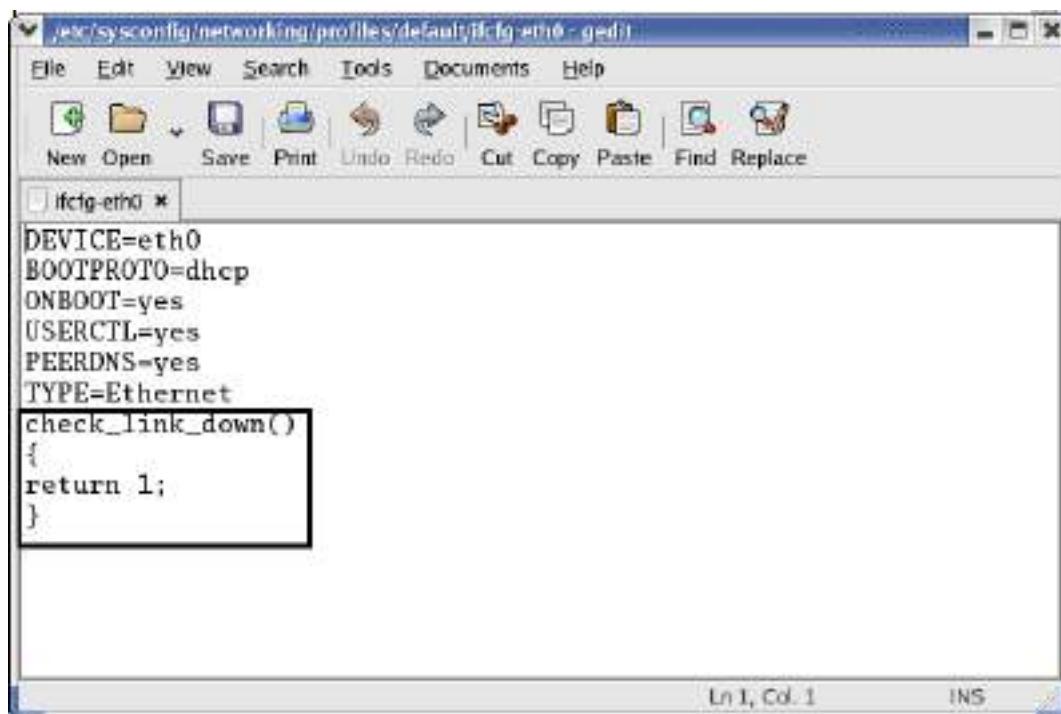
Unit 03: Connecting to Internet



```
[root@navnav root]# gedit /etc/sysconfig/networking/profiles/default/ifcfg-eth0
```

The screenshot shows a terminal window with the command `gedit /etc/sysconfig/networking/profiles/default/ifcfg-eth0` being run. Below the terminal, a file editor window titled "ifcfg-eth0" is open, displaying the configuration file for the eth0 interface. The file contains the following settings:

```
DEVICE=eth0
BOOTPROTO=dhcp
ONBOOT=yes
USERCTL=yes
PEERDNS=yes
TYPE=Ethernet
```

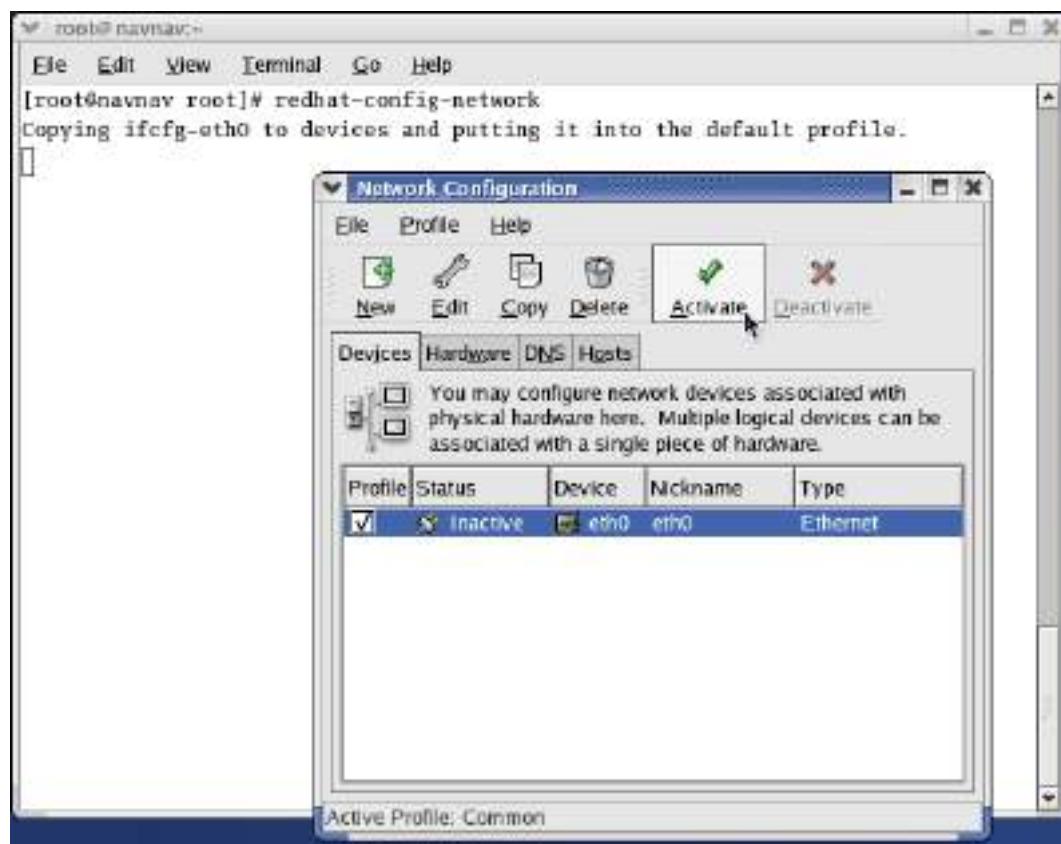


The screenshot shows a file editor window titled "ifcfg-eth0" with the following content:

```
DEVICE=eth0
BOOTPROTO=dhcp
ONBOOT=yes
USERCTL=yes
PEERDNS=yes
TYPE=Ethernet
check_link_down()
{
return 1;
}
```

A new section of code has been added to the bottom of the file, enclosed in braces {}, which defines a function named `check_link_down()` that returns 1.

After modification of these three files, we ned to activate the internet.



Once it is activated, we can browse the internet through web browser.

3.3 Domain Name System

- Rather than remember the IP address of the Wrox web site, it is easier for us to remember www.wrox.com.
- Domain Name System (DNS) servers provide the mapping between human-readable addresses (such as www.wrox.com) and the IP addresses of the machines acting as the web servers for the corresponding web service.
- Applications such as web browsers and e-mail clients require the IP address to connect to a web site or a mail server respectively.
- In order to get this from the human-readable input that we provide them with, they query a DNS server for the corresponding IP address information.
- Obviously, this also means that the browser and other clients on the machine need to know the IP address of the DNS server.
- For machines that use DHCP, the information about the DNS server is automatically available when the machine is configured.

Some useful Commands for connecting to internet:

- `#redhat-config-network` // to open network configuration manager
- `ifconfig` // to get information about active network
- `ping` // used to test the reachability of a host

Keywords

- Network Interfacing Tool is also known as the Network Administration Tool or Network Configuration Tool.
- A network device connected to a TCP/IP network has an IP address associated with it, such as 192.168.100.20. Using the IP address of a machine, other machines on the network can address it uniquely.
- Dynamic IP addresses are distributed and managed via dynamic address allocation protocols.
- Applications such as web browsers and e-mail clients require the IP address to connect to a web site or a mail server respectively.
- For machines that use DHCP, the information about the DNS server is automatically available when the machine is configured.

Summary

- **Static address:** These are allotted to machines indefinitely and do not change. Typically, static addresses are allocated to servers.
- **Dynamic address:** These are allotted to machines for a specific period with no guarantee that the same address will be available next time the machine connects to the network.
- **Devices Tab:** This tab lists the device connections that we have available on our machine.
- **Hardware Tab:** It allows us to manage the various network devices on the system, such as Ethernet cards, internal modems, and wireless cards.
- **DNS Tab:** It allows us to specify DNS server information.
- **Hosts Tab:** It allows us to modify the hostname of the machine and add aliases to the same host.
- **Domain Name System (DNS):** It provides the mapping between human-readable addresses (such as www.wrox.com) and the IP addresses of the machines acting as the web servers for the corresponding web service.

Self Assessment

1. If we want to make a connection to the LAN, then what kind of device will be chosen?
 - A. CIPC Connection
 - B. Ethernet Connection
 - C. ISDN Connection
 - D. None of the above

2. What is an internet?
 - A. A tool to write the text data
 - B. A network that connects the computer all over the world
 - C. A tool to convert the word doc to pdf
 - D. None of the above

3. The IP address can be assigned
 - A. Statically
 - B. Dynamically
 - C. Both statically and dynamically

- D. None of the above
4. What is another name of Network Interfacing Tool?
- A. Network Administration Tool
 - B. Network Configuration Tool
 - C. Both network administration and configuration tool
 - D. None of the above
5. For a machine connected to a LAN or ISP using a static IP address, we need to obtain the network details like
- A. IP address
 - B. Subnet mask
 - C. Default gateway address
 - D. All IP address, subnet mask and default gateway address
6. Which of these tabs are available in Network Interfacing Tool?
- A. Device tab
 - B. Hardware tab
 - C. DNS tab
 - D. All device, hardware, and DNS tabs
7. In the case of a machine connected to a LAN using a dynamic IP address, the address is allocated either using ____
- A. DHCP
 - B. BOOTP
 - C. Either DHCP or BOOTP
 - D. None of the above
8. In the process of activating the internet, how many files were modified?
- A. One
 - B. Two
 - C. Three
 - D. Four
9. Which of these is not a web browser?
- A. Windows
 - B. Google Chrome
 - C. Mozilla Firefox
 - D. Microsoft Edge
10. How to open the network configuration manager through command line?
- A. redhat-config-network
 - B. redhat-config-internet

- C. redhat-config-mozillabrowser
- D. None of the above

11. What is used by browsers to retrieve any published resource on the web?

- A. URL
- B. VRL
- C. LRU
- D. None of the above

12. What is the path to enter the Network Administration Wizard?

- A. Main Menu | System Tools | Internet Configuration Wizard
- B. Main Menu | Systems | Settings
- C. Main Menu | System Tools | Web Browsers
- D. None of the above

13. What is the format of IP address?

- A. x.x
- B. x.x.x
- C. x.x.x.x
- D. None of the above

14. What is required to access internet?

- A. Pdf converter
- B. Photoshop
- C. Web browser
- D. None of the above

15. What provides the interfacing between human-readable address and IP address of the machine?

- A. DNS
- B. ABC
- C. Wrox
- D. None of the above

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. C | 2. B | 3. C | 4. C | 5. D |
| 6. D | 7. C | 8. C | 9. A | 10. A |
| 11. A | 12. A | 13. C | 14. C | 15. A |

Review Questions:

1. Is it necessary to configure the system before the internet connection take place? If yes, how can we configure it?
2. What is an IP address? Explain the difference between static and dynamic IP address.
3. What is a network interfacing tool? Explain its tabs.
4. What is DNS? Explain.
5. Explain the process of configuring the system for internet connection.



Further Readings

Mark G. Sobell, A Practical Guide to Fedora and Red Hat Enterprise Linux, Fifth Edition, Pearson Education, Inc.



Web Links

<https://www.redhat.com/sysadmin/network-interface-linux>

Unit 04: Installing Software

CONTENTS

- Objectives
- Introduction
- 4.1 RPM Package Manager
- 4.2 Adding and Removing Packages
- 4.3 RPM Command Line Tool
- 4.4 Querying Packages
- 4.5 Package Installation in TAR Format
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions:
- Further Readings

Objectives

After studying this unit, you will be able to understand

- Understand RPM
- See the RPM Package Management tool
- Add and remove the packages
- Query the RPM package
- Install package in TAR format

Introduction

RPM stands for Redhat Package Manager. The RPM package manager is an open-source packaging system distributed under the GNU GPL. It runs on most Linux distributions and makes it easy for you to install, uninstall, and upgrade the software on your machine. RPM files can be easily recognized by their .rpm file extension and the 'package' icon that appears in your navigation window. The RPM package management is shown below.

4.1 RPM Package Manager



Benefits of using RPM: There are few reasons to use RPM.

- **Simplicity:** RPM is quite simple to use. The interface of RPM is very clear. The packages and the groups in RPM are very easy to locate. So, this is the remarkable feature of RPM.
- **Upgradability:** RPM interface is easy to upgrade. If a new package comes, it can be easily upgraded.
- **Manageability:** RPM interface is easily manageable. If we want to add or delete some packages using RPM, then it can be easily done. So, the manageability is one of the greatest feature of RPM interface.
- **Package Queries:** The packages are easily queried in RPM. By querying the packages, we can see which all packages are installed in the computer system.
- **Uninstalling:** It is very easy to uninstall a package or a group. If we don't need any package or its related extra group at some time, then it can be deleted at that time.
- **System Verification:** System verification can be easily done using RPM.
- **Security:** The RPM way of installing and installing packages is secure.

Ways to use RPM: RPM can be used in two different, yet complementary ways –

- From the desktop, using the GUI interface,
- From the command line.

The RPM package management (GUI) tool

This tool is a graphical user interface (GUI) designed for the management of package installation and removal. The GUI allows us to add and remove packages at the click of a mouse.

Starting the RPM Package Management Tool: There are two ways to start RPM.

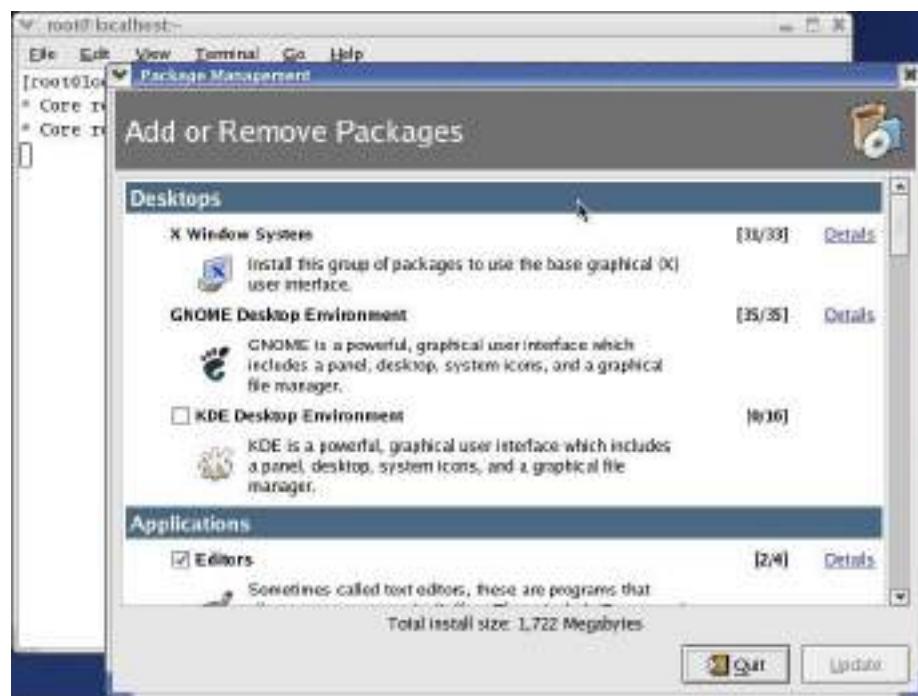
- Main Menu, select Main Menu | System Settings | Add/Remove Applications.
- \$ redhat-config-packages

When you use either of the way, the next moment your screen will look like this. It checks the system package status.

Unit 04: Installing Software

The package management window shows all the packages and its group. The interface contains:

- Package category
- Package group
- Details link
- Number of packages installed/Out of total number of packages
- Summary of disk space required to install the package
- Update button
- Quit button



Linux and Shell Scripting

Package categories and groups: There are various package categories and groups available.

Standard and extra packages

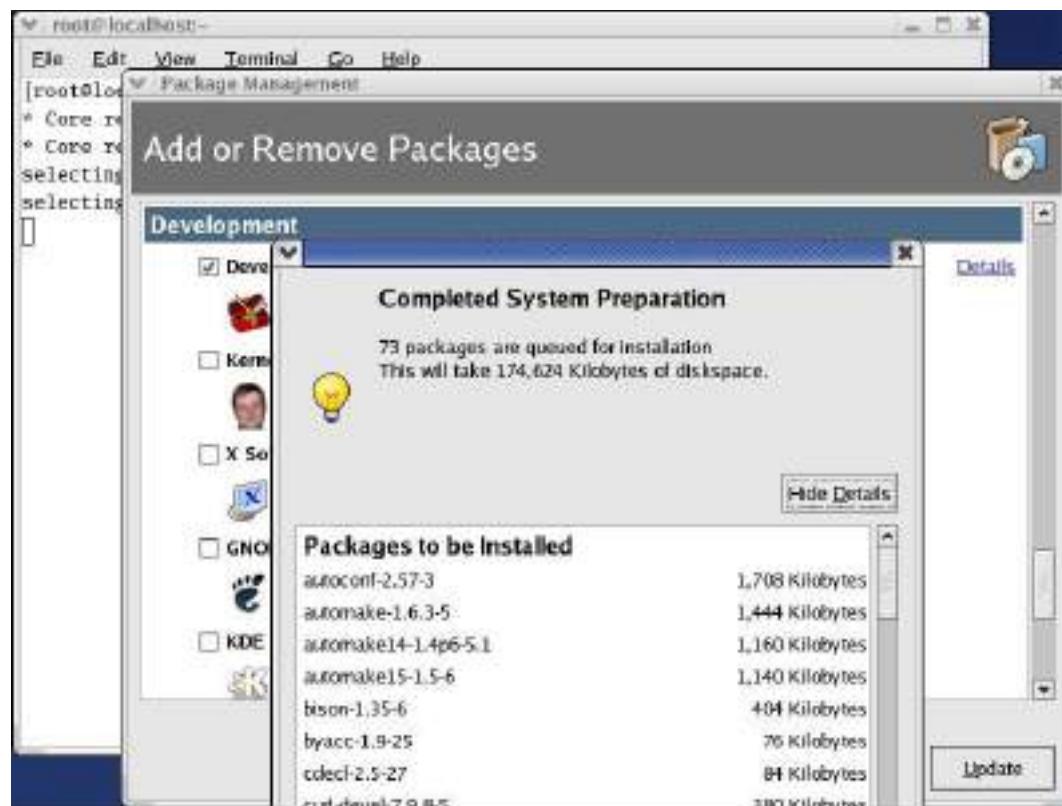
Each group may have standard packages and extra packages, or just extra packages. Standard packages are always available when a package group is installed – so you can't add or remove them explicitly unless the entire group is removed. However, the extra packages are optional so they can be individually selected for installation or removal at any time.

Package Category	Package Groups
Desktop	X Window System GNOME Desktop Environment KDE Desktop Environment
Applications	Editors Engineering and Scientific Graphical Internet Text-based Internet Office/Productivity Sound and Video Authoring and Publishing Graphics Games and Entertainment
Servers	Server Configuration Tools Web Server Mail Server Windows File Server DNS Name Server FTP Server SQL Database Server News Server Network Servers
Development	Development Tools Kernel Development X Software Development GNOME Software Development KDE Software Development
System	Administration Tools System Tools Printing Support

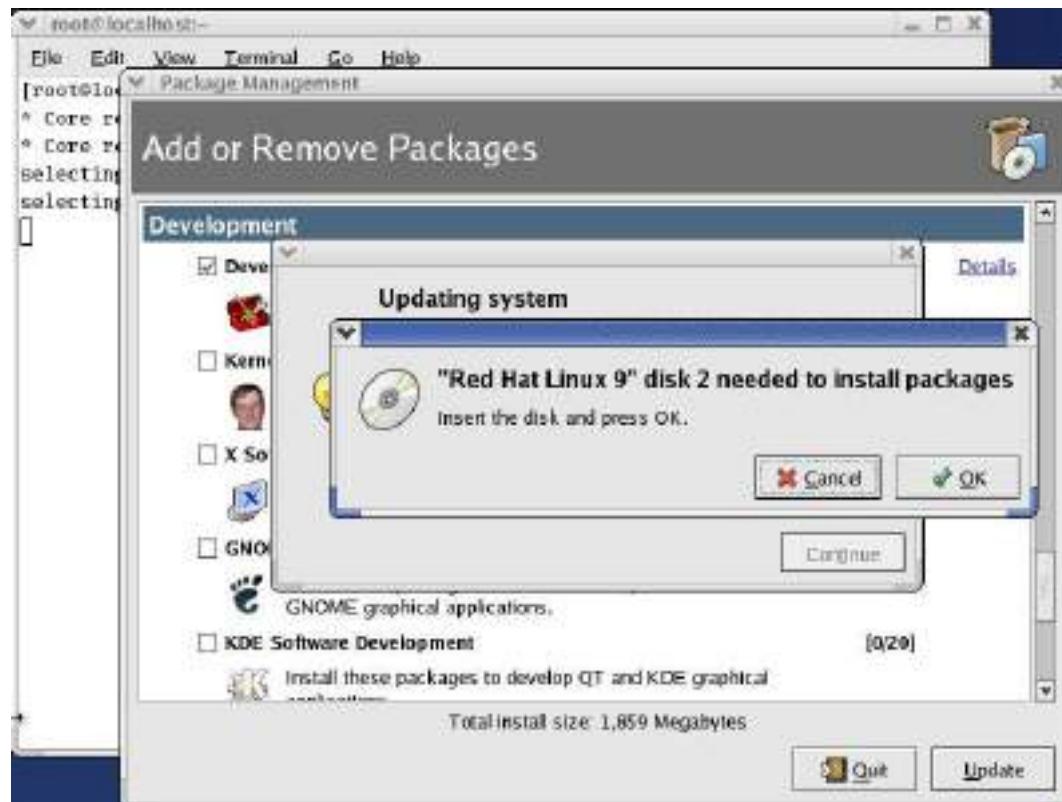
4.2 Adding and Removing Packages

Installing new software from the package management tool is very simple. When we select any group using the RPM package management tool interface, it automatically selects the standard packages (if any) that are needed for the category as well as any dependent packages that it may have. We can customize the packages to be installed by clicking on the Details button. Once you've made your selections, click on the Update button on the main window. The package management tool will then calculate the disk space required for installing packages, as well as any dependencies, before displaying the following dialog:





Use the required disk for updation of system.



It will start updating the system.

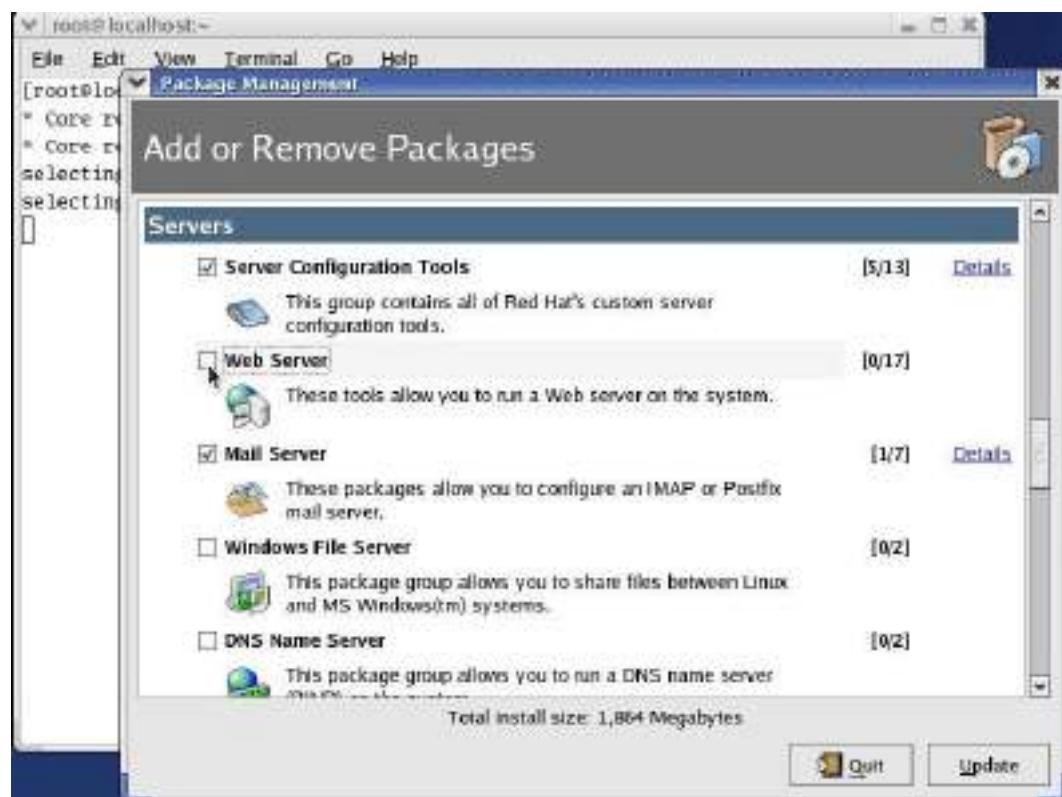
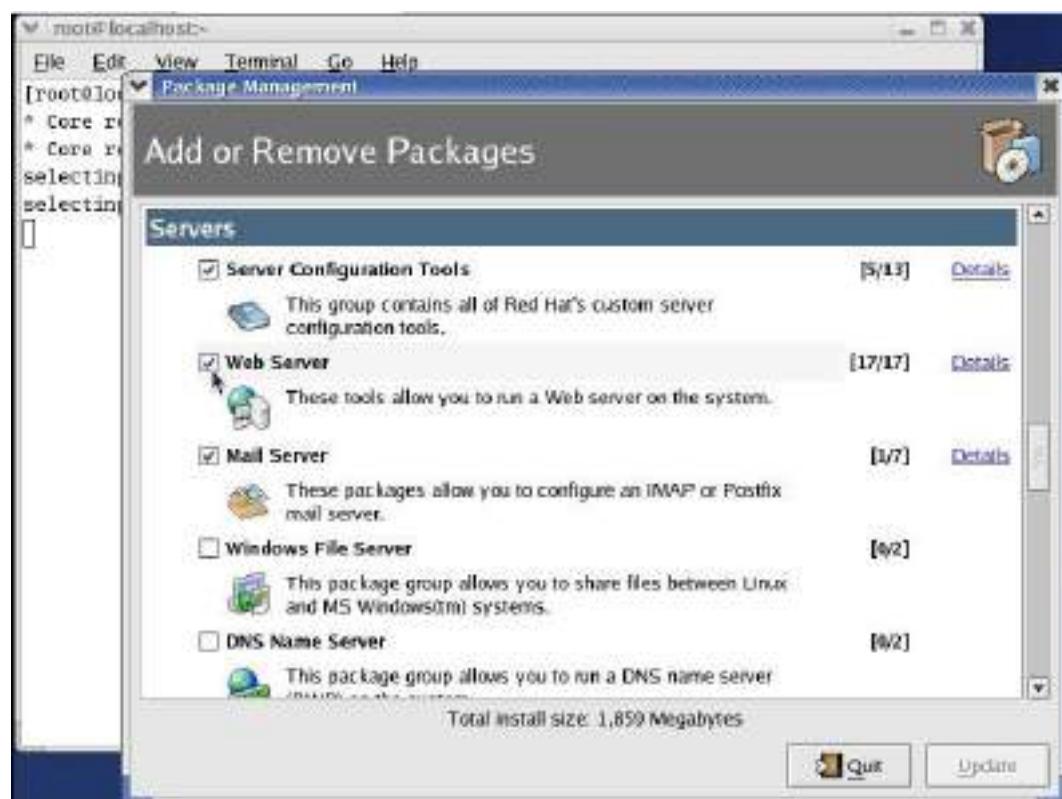
Unit 04: Installing Software



When the update is complete, it will show the updation complete message.

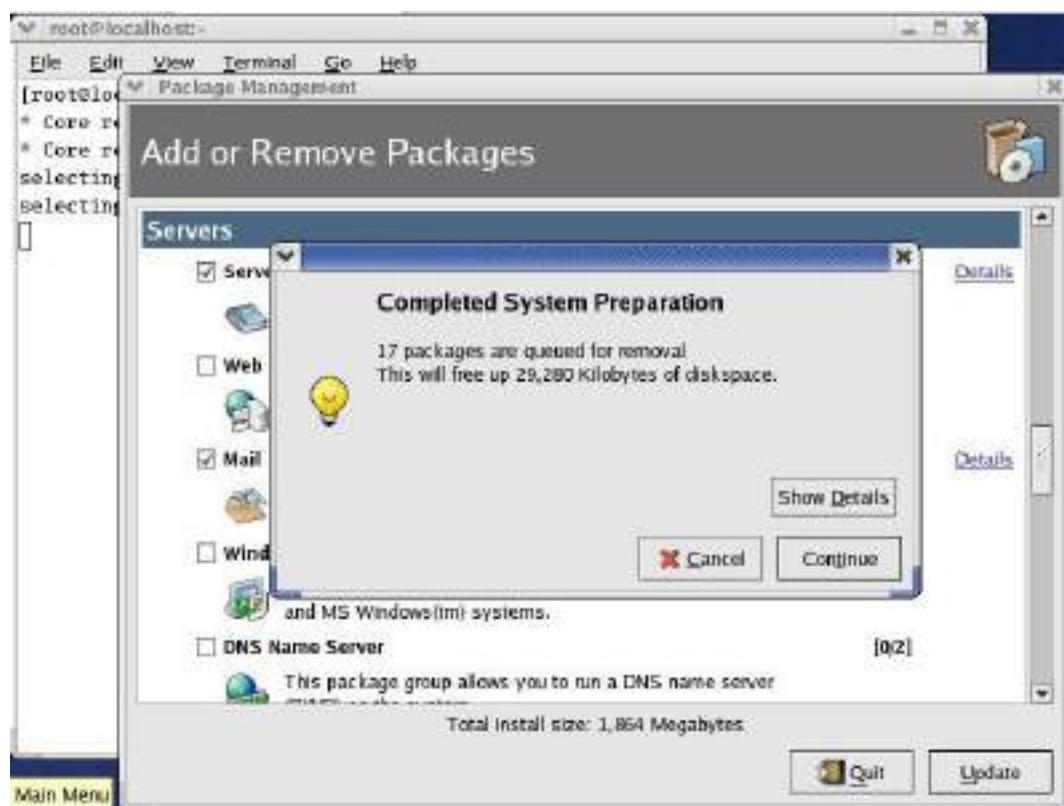


We can also delete the package if some package is no longer required. We just need to uncheck that package and click on update.

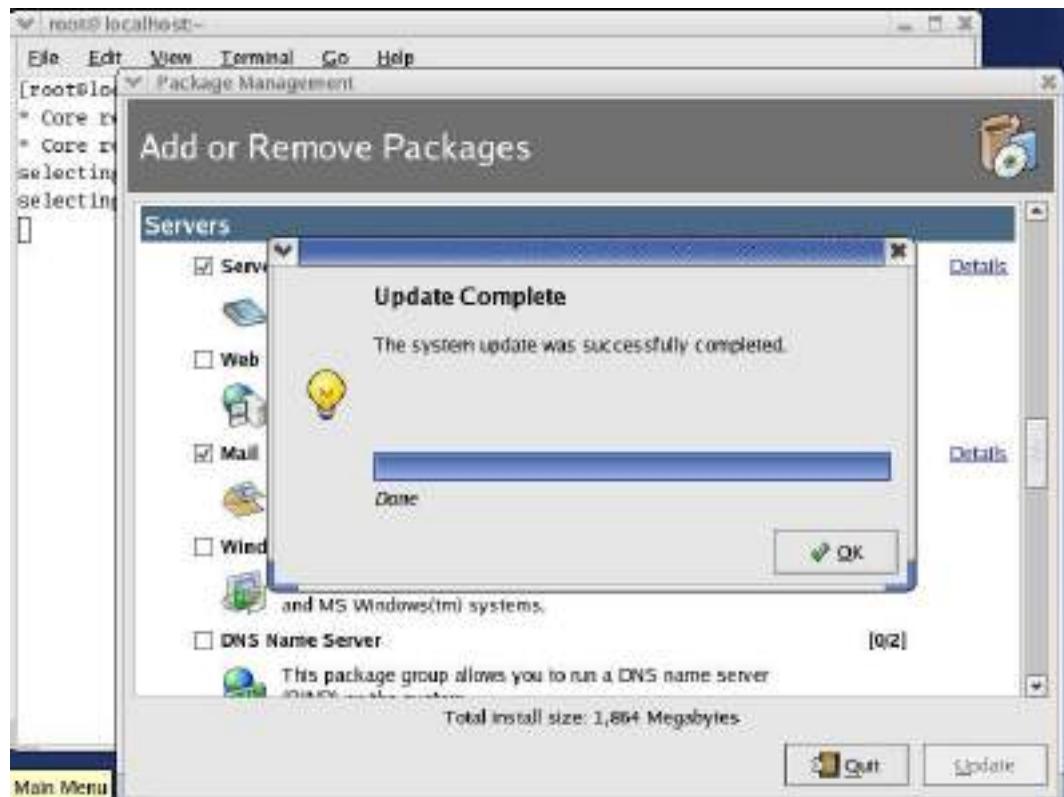


Then the packages will be placed in queue for deletion. Here the 17 packages are queued for removal. It will also show how much space will be freed after removal of packages. If the packages are correct for deletion (click on show details for confirmation), click on continue. It will remove the packages from the system.

Unit 04: Installing Software



When the update is complete. Click on ok.



4.3 RPM Command Line Tool

Up to now, we've talked about how to install and remove packages using Red Hat's graphical package management tool. While this tool is simple to use, it's lacking in functionality. For example:

- It cannot install packages using network, FTP, or HTTP connections.
- It does not show the location the files in a package are installed to.
- It lacks the capability to query for specific packages installed on the system.
- It does not provide full details of the RPM package – such as the vendor, build date, signature, description, and so on.
- It does not have a function to verify a package. That means it cannot compare information about files like size, MD5 sum, permissions, type, owner, and group installed from a package with the same information from the original package.
- It does not show all the packages available in a product. So you won't always know if you've got the whole thing.

4.4 Querying Packages

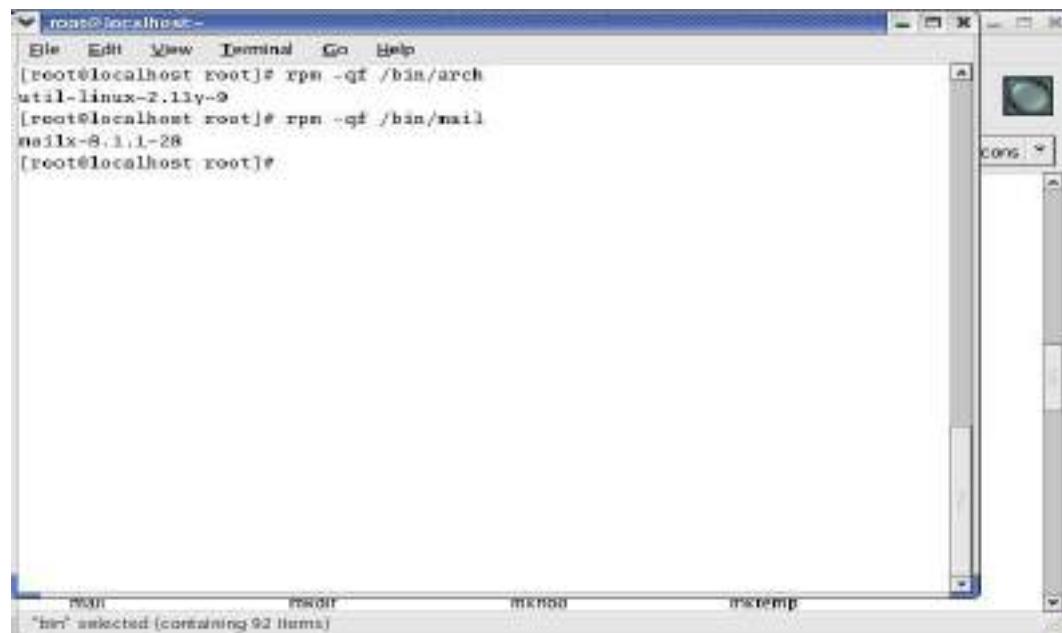
RPM keeps a record of all the packages installed on your system in a database. By querying the database, you obtain a complete list of all the packages that you've installed on your system. Should you want to, you can then go further and query each individual package for more details about itself.

The syntax for a basic query is as follows:

- rpm -q [options] <filename>

You have list of files, and you would like to find out which package belongs to these files. The syntax is:

- rpm -qf<filename>



A screenshot of a terminal window titled "root@localhost". The window shows the following command and output:

```
[root@localhost root]# rpm -qf /bin/arch
util-linux-2.13.y-9
[root@localhost root]# rpm -qf /bin/mail
mailx-8.1.1-28
[root@localhost root]#
```

The terminal window has a blue header bar with the title "root@localhost". Below the title is a menu bar with "File", "Edit", "View", "Terminal", "Go", and "Help". The main area of the window contains the command-line interface. On the right side, there is a vertical scroll bar and a small icon in the top right corner. At the bottom, there is a status bar with tabs labeled "MAIL", "MAILDIR", "MESSAGE", and "INBOX". A message at the bottom of the status bar says "Inbox selected (containing 92 items)".

You have installed an rpm package and want to know the information about the package. The syntax is:

- rpm -qi <package _name>

```
[root@localhost root]# rpm -qi vsftpd
Name        : vsftpd                                Relocations: (not relocatable)
Version     : 1.1.3                                 Vendor: Red Hat, Inc.
Release     : 8                                    Build Date: Fri 28 Feb 2003 02:21:36
PM EST
Install Date: Wed 14 Apr 2021 09:19:42 AM EDT      Build Host: daffy.perf.redhat
.com
Group       : System Environment/Daemons          Source RPM: vsftpd-1.1.3-8.src.rpm
Size        : 149635                               License: GPL
Signature   : DSA/SHA1, Thu 13 Mar 2003 04:50:02 PM EST, Key ID 219180cddb42a60e
Packager    : Red Hat, Inc. <http://bugzilla.redhat.com/bugzilla>
Summary     : vsftpd - Very Secure Ftp Daemon
Description :
vsftpd is a Very Secure FTP daemon. It was written completely from
scratch.
[root@localhost root]#
```

You have downloaded a package from the internet and want to know the information of a package before installing. The syntax is:

- `rpm -qip<package_name>`

To get the list of available documentation of an installed package. The syntax is:

- `rpm -qdf<package_name>`

```
[root@localhost root]# rpm -qdf /usr/bin/vmstat
/usr/share/doc/procps-2.0.11/BUGS
/usr/share/doc/procps-2.0.11/NEWS
/usr/share/doc/procps-2.0.11/TODO
/usr/share/man/man1/free.1.gz
/usr/share/man/man1/oldps.1.gz
/usr/share/man/man1/pgrep.1.gz
/usr/share/man/man1/pkill.1.gz
/usr/share/man/man1/ps.1.gz
/usr/share/man/man1/skill.1.gz
/usr/share/man/man1/snice.1.gz
/usr/share/man/man1/tload.1.gz
/usr/share/man/man1/top.1.gz
/usr/share/man/man1/uptime.1.gz
/usr/share/man/man1/w.1.gz
/usr/share/man/man1/watch.1.gz
/usr/share/man/man5/mysctl.conf.5.gz
/usr/share/man/man8/mysctl.8.gz
/usr/share/man/man8/vmstat.8.gz
[root@localhost root]#
```

4.5 Package Installation in TAR Format

TAR stands for tape archive. An archive is nothing, but you bundle (put) many files together into a single file on a single tape or disk. The tar program combines multiple files into a single large file. It is separate from the compression tool, so it allows you to select which compression tool to use or whether you even want compression. A tar ball is a (usually compressed) archive of files, similar to

Linux and Shell Scripting

a Zip file on Windows or a Sit on the Mac. Tar balls come in files that end in .tar, .tar.gz, .tgz, or something along these lines.

- Structure of the tar command
- ```
[root@localhost /root]# tar [commands and options] filename
```
- Option for tar
    - c Create a new archive.
    - t View the contents of an archive.
    - x Extract the contents of an archive.
    - f Specify the name of the file (or device) in which the archive is located.
    - v Be verbose during operations.
    - z Use gzip to compress or decompress the file.
    - u To upgrade the tar file

### To create a tar file

```
[root@localhostmohit]# tar -cvf filename.tar directory [path of files]
```

In this example, filename.tar represents the file you are creating and directory/file represents the directory and file you want to put in the archived file.

Where,

- c : Create a tar ball.
- v : Verbose output (show progress).
- f : Output tar ball archive file name.
- x : Extract all files from archive.tar.
- t : Display the contents (file list) of an archive.

### To View a Tar Ball

- It Contains (list file inside a tar ball)
  - Type the following command:
- ```
tar -tvf/tmp/data.tar ar Ball
```

To Extract a Tar Ball

Type the following command to extract /tmp/data.tar in a current directory, enter:

```
tar -xvf/tmp/data.tar
```

Summary

- RPM files can be easily recognized by their .rpm file extension and the 'package' icon that appears in your navigation window.
- There are various benefits of using RPM for package installation and deletion: simplicity, upgradability, manageability, easy uninstallation, system verification and high security.
- The RPM package management tool contains: the button for package category, package group, details link, number of packages installed/Out of total number of packages, summary of disk space required to install the package, update button, and quit button.
- Each group may have standard packages and extra packages, or just extra packages.
- We can customize the packages to be installed by clicking on the Details button. Once you've made your selections, click on the Update button on the main window.
- RPM keeps a record of all the packages installed on your system in a database.

Unit 04: Installing Software

- The tar program combines multiple files into a single large file. It is separate from the compression tool, so it allows you to select which compression tool to use or whether you even want compression.

Keywords

- **RPM:**The RPM package manager is an open-source packaging system distributed under the GNU GPL.
- **Standard packages:** These are always available when a package group is installed – so you can't add or remove them explicitly unless the entire group is removed.
- **Extra packages:**These are optional so they can be individually selected for installation or removal at any time.
- **Archive:**An archive is nothing, but you bundle (put) many files together into a single file on a single tape or disk.
- **Tar ball:** A tar ball is a (usually compressed) archive of files, like a Zip file on Windows or a Sit on the Mac. Tar balls come in files that end in .tar, .tar.gz, .tgz, or something along these lines.

Self Assessment

1. TAR stands for
 - A Tour Archive
 - B Tape Archive
 - C Tape Assistance
 - D Tour Assistance
2. Under development tools, what can be installed using RPM package management tool?
 - A KDE Software development
 - B GNOME Software development
 - C X Software development
 - D All of the above
3. While graphical interface of RPM package management tool can install/remove/update the packages, but it still lacks which functionality.
 - A It cannot install packages using network, FTP, or HTTP connections.
 - B It does not show the location the files in a package are installed to.
 - C Both above
 - D None of the above
4. The RPM package management tool is a _____
 - A Graphical interface
 - B Textual interface
 - C Not an interface
 - D None of the above

5. Using RPM package management tool, we can install
 - A. Web server
 - B. Mail server
 - C. DNS name server
 - D. All of the above

6. We can check which packages are installed/ not installed by clicking on
 - A. Update button
 - B. Quit button
 - C. Details link
 - D. None of the above

7. Which of these packages are always available when a package group is installed?
 - A. Standard packages
 - B. Extra packages
 - C. Grouped packages
 - D. None of the above

8. Which of the buttons are available on the tool interface?
 - A. Update button
 - B. Quit button
 - C. Both update and quit buttons
 - D. None of the above

9. The slash (/) in the interface of RPM package management tool represents:
 - A. Total number of packages/ Number of packages installed
 - B. Number of packages installed/ total number of packages
 - C. Package category/ package group
 - D. Package group/ package category

10. What is the extension of the RPM file
 - A. .txt
 - B. .doc
 - C. .rpm
 - D. .pdf

11. How can we start RPM package management tool?
 - A. \$redhat-config-services
 - B. \$redhat-config-packages
 - C. \$redhat-config-management
 - D. None of the above

12. What are the benefits of using RPM?

- A. Package queries
 B. System verification
 C. Security
 D. All of the above
13. With RPM, it is easy to _____ softwares on the computer system.
 A. Install
 B. Uninstall
 C. Upgrade
 D. All: install, uninstall and upgrade
14. From _____, it is possible to install packages network, FTP or HTTP connections.
 A. Command line
 B. Graphical interface
 C. Both command line and graphical interface
 D. None of the above
15. Which of these commands is used for querying the information about a package after installation?
 A. rpm -qi<filename>
 B. rpm -qu<filename>
 C. rpm -qr<filename>
 D. None of the above

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. B | 2. D | 3. C | 4. A | 5. D |
| 6. C | 7. A | 8. C | 9. B | 10. C |
| 11. B | 12. D | 13. D | 14. A | 15. A |

Review Questions:

1. What is RPM? Write the ways and benefits of using RPM.
2. What is RPM package management tool? How can we start it? Explain some details about its interface.
3. How can we add and remove the packages? Explain.
4. What is RPM command line tool? Write its benefits.
5. How can we query a package? Write syntax.
6. Explain the package installation in TAR format. How can we create, view and extract a tar ball?



Further Readings

Mark G. Sobell, A Practical Guide to Fedora and Red Hat Enterprise Linux, Fifth Edition, Pearson Education, Inc.



Web Links

<https://www.redhat.com/sysadmin/create-rpm-package>

Unit 05: Utilities

CONTENTS

- Objectives
- Introduction
- 5.1 Common Utilities
- 5.2 Working With Files
- 5.3 Four More Utilities
- 5.4 Compressing and Archiving Files
- 5.5 Locating Commands
- Summary:
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions:
- Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the basic utilities
- Work with files
- Understand the Pipe
- Understand the compressing and archiving of files
- Understand the locating commands

Introduction

Command-line utilities are often faster, more powerful, or more complete than their GUI counterparts. When you work with a command-line interface, you are working with a shell. One of the important advantages of Linux is that it comes with thousands of utilities that perform innumerable functions.

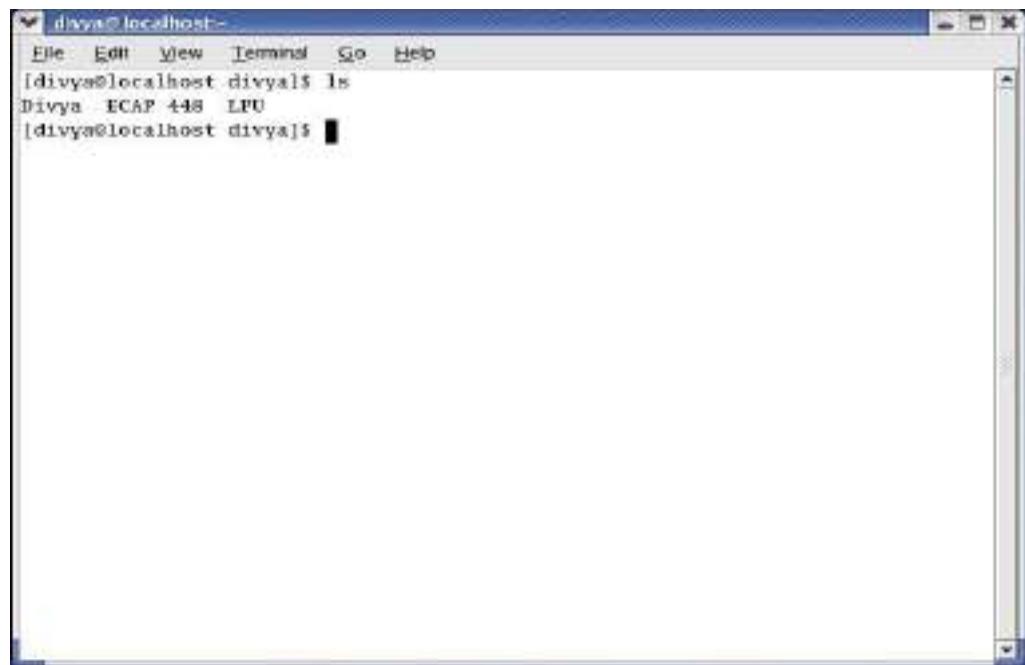
- ls
- cat
- rm
- less
- more

5.1 Common Utilities

ls: Lists the Names of Files

The ls utility lists the names of files which are available. ls is a Linux shell command that lists directory contents of files and directories.

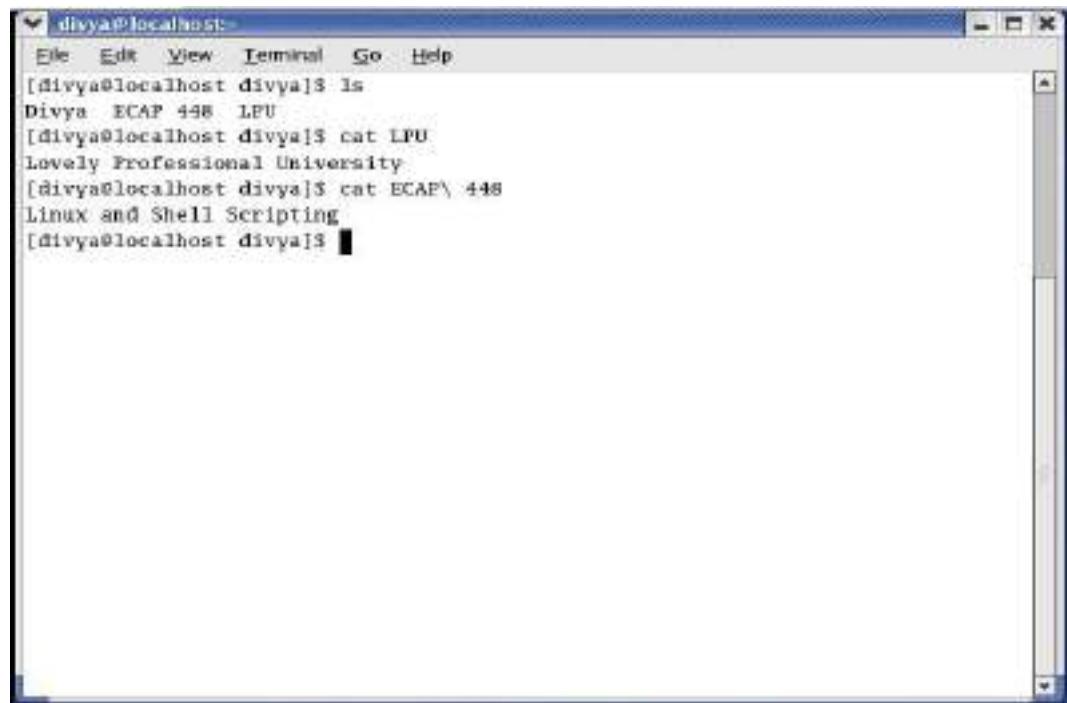
Linux and Shell Scripting



A screenshot of a Linux terminal window titled "divya@localhost:~". The window has a menu bar with File, Edit, View, Terminal, Go, and Help. The terminal prompt is "[divya@localhost divya]~". The user has run the "ls" command, which lists two files: "Divya" and "ECAP 448 LPU". The window has a vertical scroll bar on the right side.

cat: Displays the Text of File

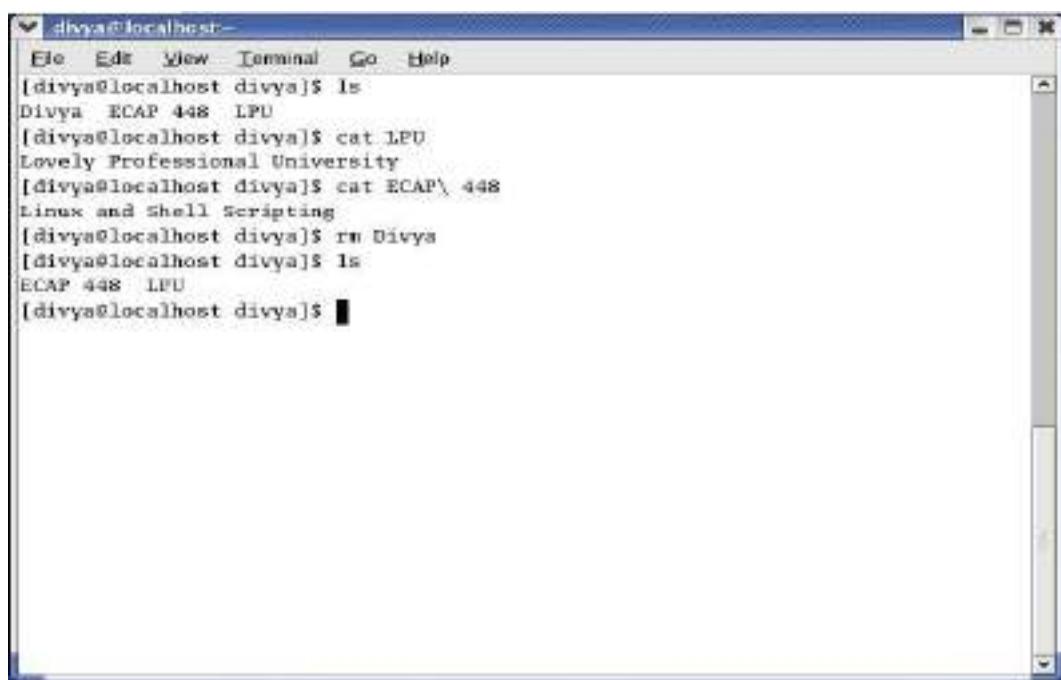
The cat utility displays the contents of a text file. The name of the command is derived from catenate, which means to join, one after the other.



A screenshot of a Linux terminal window titled "divya@localhost:~". The window has a menu bar with File, Edit, View, Terminal, Go, and Help. The terminal prompt is "[divya@localhost divya]~". The user has run the "ls" command to list files, then run "cat LPU" to display the contents of the "LPU" file, which contains "Lovely Professional University". Then, they ran "cat ECAP\ 448" to display the contents of the "ECAP\ 448" file, which contains "Linux and Shell Scripting". The window has a vertical scroll bar on the right side.

rm: Deletes a File

The rm (remove) utility deletes a file.

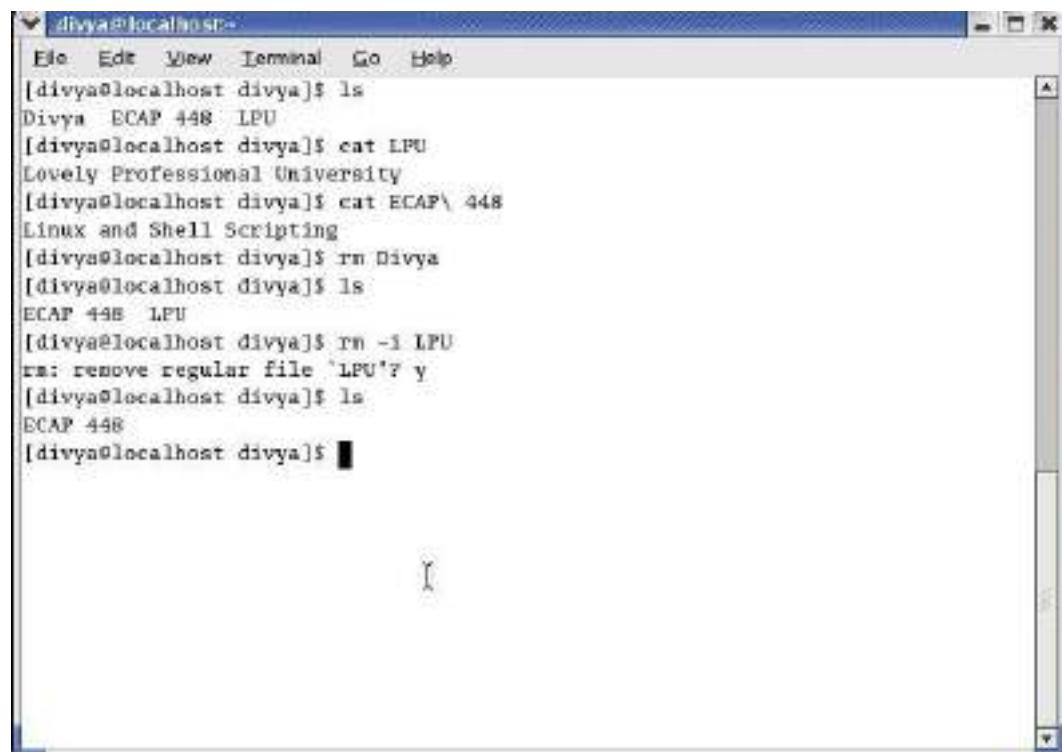
Unit 05: Utilities

```
divya@localhost:~$ ls
Divya ECAP 448 LPU
[divya@localhost divya]$ cat LPU
Lovely Professional University
[divya@localhost divya]$ cat ECAP\ 448
Linux and Shell Scripting
[divya@localhost divya]$ rm Divya
[divya@localhost divya]$ ls
ECAP 448 LPU
[divya@localhost divya]$
```

When you follow rm with the -i option and the name of the file you want to delete, rm displays the name of the file and then waits for you to respond with y (yes) before it deletes the file. It does not delete the file if you respond with a string that begins with a character other than y.



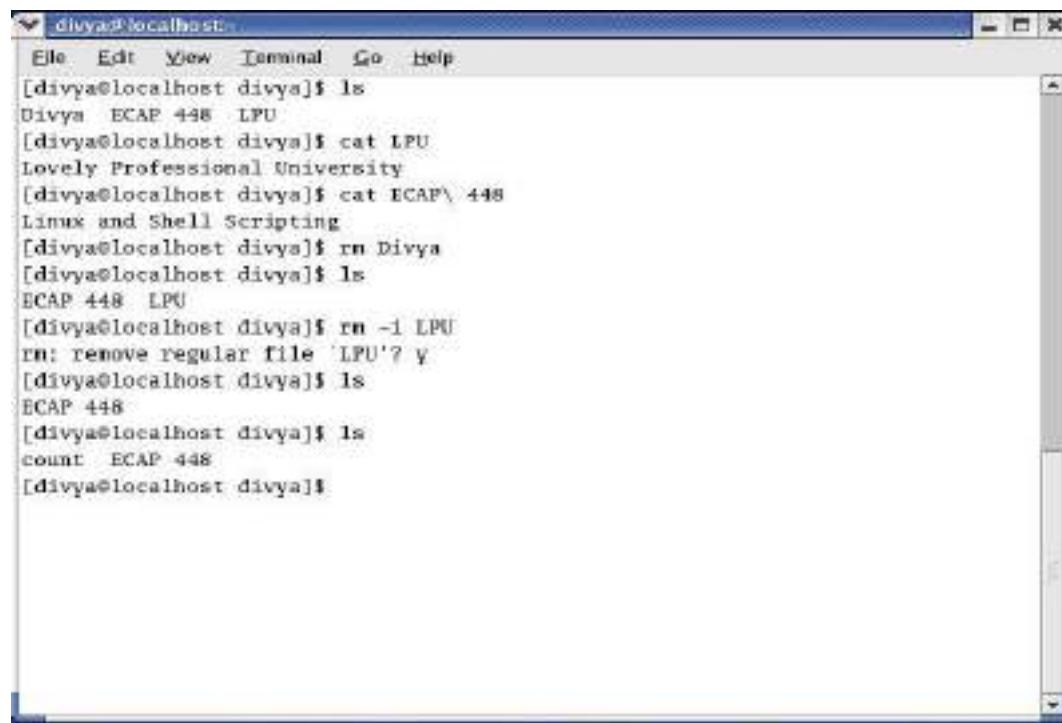
```
divya@localhost:~$ ls
Divya ECAP 448 LPU
[divya@localhost divya]$ cat LPU
Lovely Professional University
[divya@localhost divya]$ cat ECAP\ 448
Linux and Shell Scripting
[divya@localhost divya]$ rm Divya
[divya@localhost divya]$ ls
ECAP 448 LPU
[divya@localhost divya]$ rm -i LPU
rm: remove regular file 'LPU'? y
```



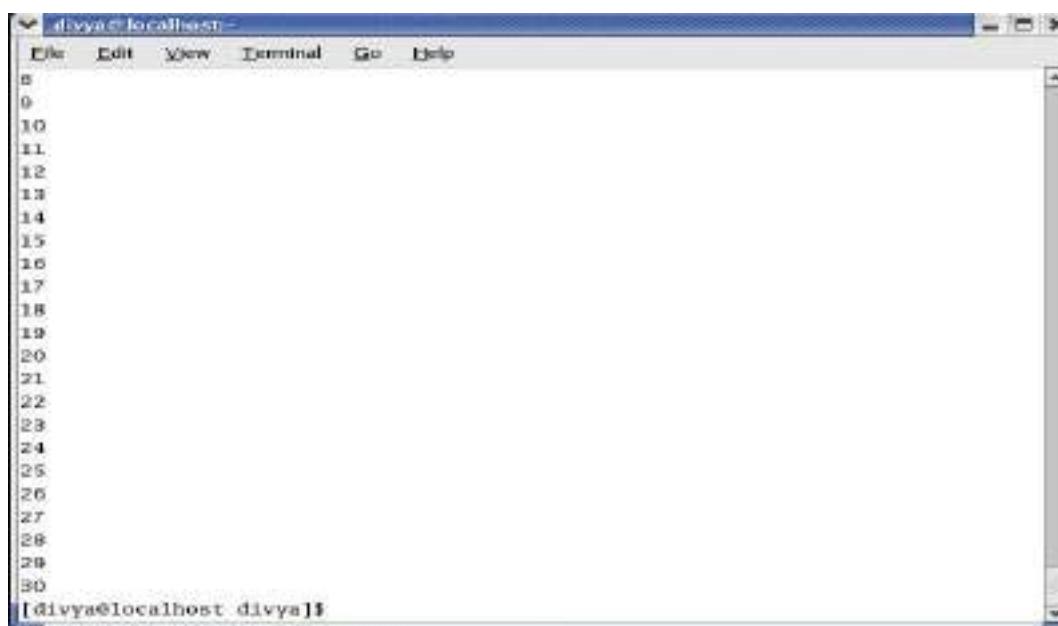
```
[divya@localhost divya]$ ls
Divya ECAP 448 LPU
[divya@localhost divya]$ cat LPU
Lovely Professional University
[divya@localhost divya]$ cat ECAP\ 448
Linux and Shell Scripting
[divya@localhost divya]$ rm Divya
[divya@localhost divya]$ ls
ECAP 448 LPU
[divya@localhost divya]$ rm -i LPU
rm: remove regular file 'LPU'? y
[divya@localhost divya]$ ls
ECAP 448
[divya@localhost divya]$
```

less Is more: Display a Text File One Screen at a Time

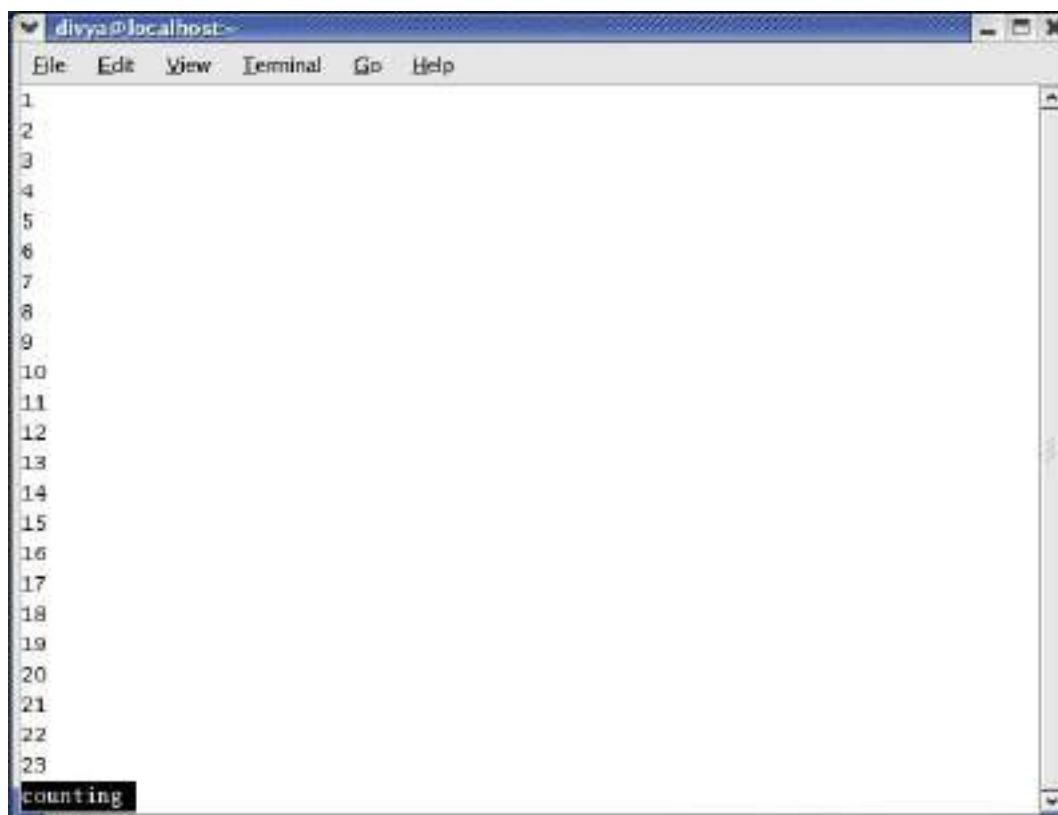
It displays a text file one screen at a time. When you want to view a file that is longer than one screen, you can use either the less utility or the more utility. Each of these utilities pauses after displaying a screen of text; press the SPACE bar to display the next screen of text. Because these utilities show one page at a time, they are called pagers. Although less and more are very similar, they have subtle differences. At the end of the file, for example, less displays an END message and waits for you to press q before returning you to the shell. In contrast, more returns you directly to the shell. While using both utilities you can press h to display a Help screen that lists commands you can use while paging through a file.



```
[divya@localhost divya]$ less LPU
[divya@localhost divya]$ cat LPU
Lovely Professional University
[divya@localhost divya]$ cat ECAP\ 448
Linux and Shell Scripting
[divya@localhost divya]$ rm Divya
[divya@localhost divya]$ ls
ECAP 448 LPU
[divya@localhost divya]$ rm -i LPU
rm: remove regular file 'LPU'? y
[divya@localhost divya]$ ls
count ECAP 448
[divya@localhost divya]$
```

Unit 05: Utilities

A screenshot of a terminal window titled "divya@localhost:~". The window has a menu bar with "File", "Edit", "View", "Terminal", "Go", and "Help". The main area of the terminal is completely blank, showing only the command prompt "[divya@localhost ~]\$".



A screenshot of a terminal window titled "divya@localhost:~". The window has a menu bar with "File", "Edit", "View", "Terminal", "Go", and "Help". The main area of the terminal is mostly blank, showing only the command prompt "[divya@localhost ~]\$". At the bottom of the screen, the word "counting" is partially visible, with its last few letters highlighted in a red box, indicating they are selected or being typed.

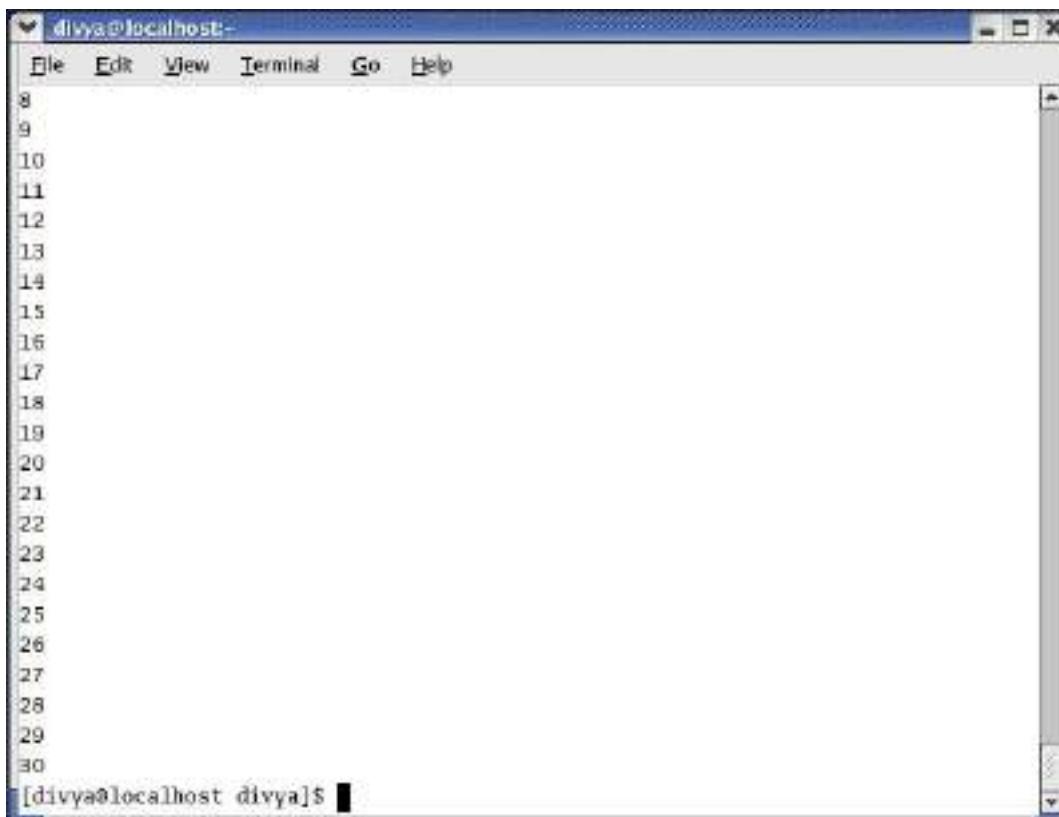
Linux and Shell Scripting

The screenshot shows a terminal window titled 'divya@localhost:-'. The menu bar includes File, Edit, View, Terminal, Go, and Help. The main area displays a numbered list from 8 to 30, with a cursor at position 25. A small vertical bar is positioned between lines 25 and 26. At the bottom left, a button labeled '(END)' is visible.

```
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

The screenshot shows a terminal window titled 'divya@localhost:-'. The menu bar includes File, Edit, View, Terminal, Go, and Help. The prompt '[divya@localhost divya]\$' is followed by the command 'more counting'. The main area displays a numbered list from 1 to 22. A progress bar at the bottom indicates 'More... (70%)'.

```
[divya@localhost divya]$ more counting
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```



The screenshot shows a terminal window titled "divya@localhost:-". The window has a menu bar with File, Edit, View, Terminal, Go, and Help. The main area contains a blank document with line numbers 8 through 30 listed vertically on the left side. The bottom status bar shows "[divya@localhost divya]\$".

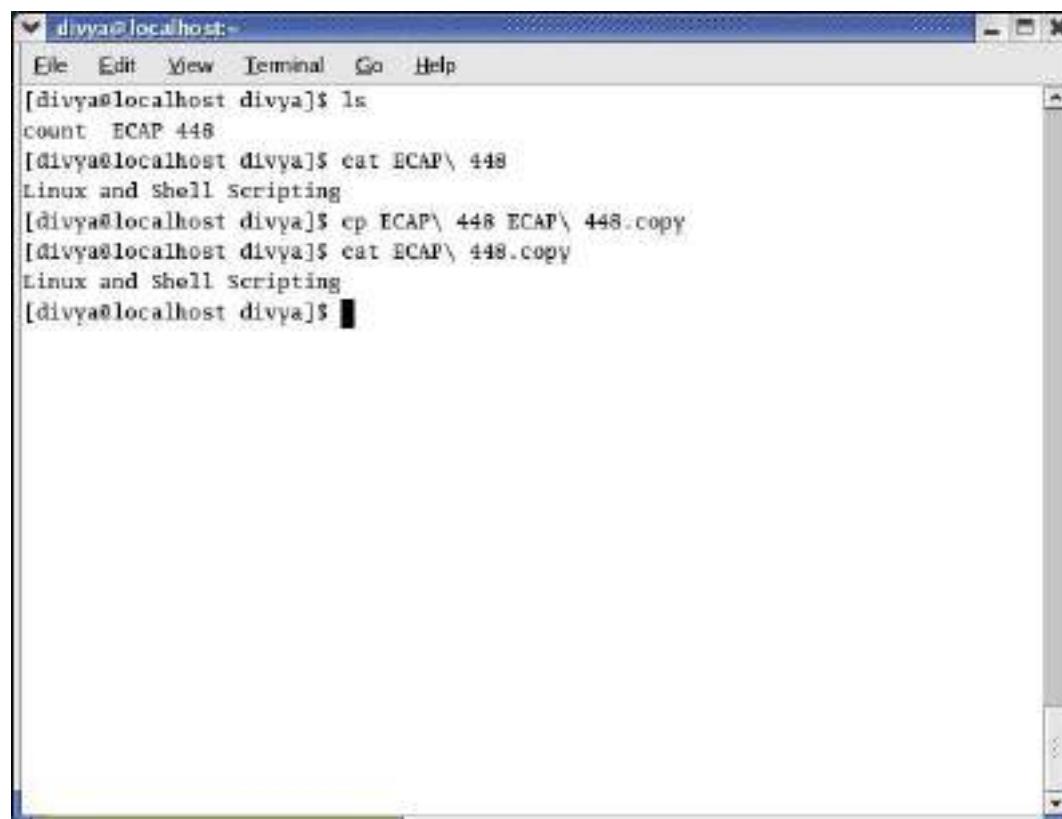
5.2 Working With Files

There are various utilities which we can use when we are working with files. These are:

- cp
- mv
- grep
- head
- tail
- sort
- uniq
- diff
- file

cp: Copies a File

The cp (copy) utility makes a copy of a file. This utility can copy any file, including text and executable program (binary) files. You can use cp to make a backup copy of a file or a copy to experiment with. The cp command line uses the following syntax to specify source and destination files: **cp source-file destination-file**



```
[divya@localhost divya]$ ls
count ECAP 448
[divya@localhost divya]$ cat ECAP\ 448
Linux and Shell Scripting
[divya@localhost divya]$ cp ECAP\ 448 ECAP\ 448.copy
[divya@localhost divya]$ cat ECAP\ 448.copy
Linux and Shell Scripting
[divya@localhost divya]$
```

cp can destroy a file: if the destination-file exists before you give a **cp** command, **cp** overwrites it. The **cp -i** (interactive) option prompts you before it overwrites a file.

mv: Changes the Name of a File

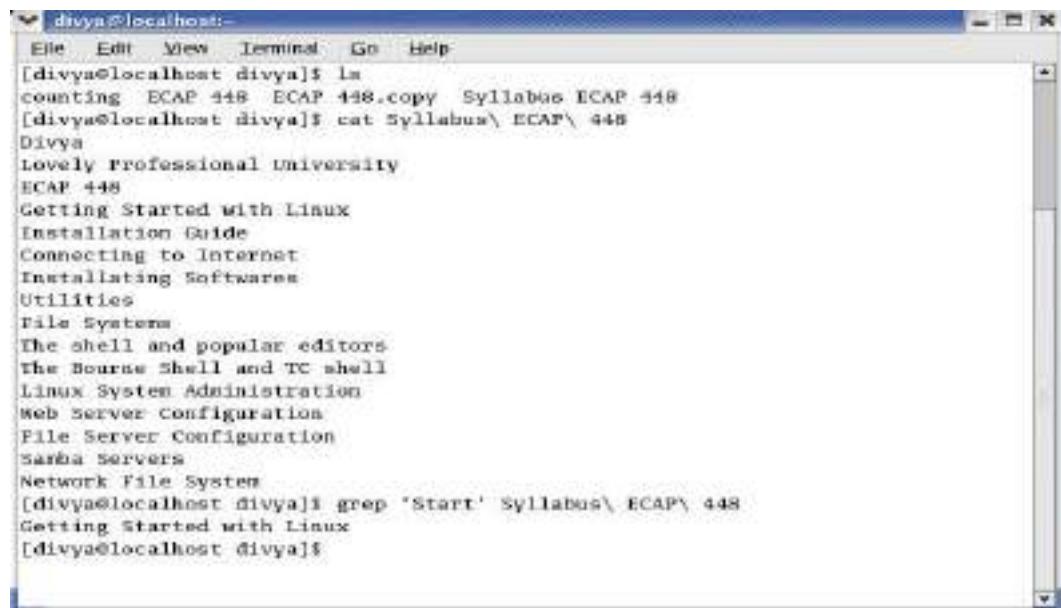
The **mv** (move) utility can rename a file without making a copy of it. The **mv** command line specifies an existing file and a new filename using the same syntax as **cp**: **mv existing-filename new-filename**



```
[divya@localhost divya]$ ls
count ECAP 448
[divya@localhost divya]$ cat ECAP\ 448
Linux and Shell Scripting
[divya@localhost divya]$ cp ECAP\ 448 ECAP\ 448.copy
[divya@localhost divya]$ cat ECAP\ 448.copy
Linux and Shell Scripting
[divya@localhost divya]$ ls
count ECAP 448 ECAP 448.copy
[divya@localhost divya]$ mv count counting
[divya@localhost divya]$ ls
counting ECAP 448 ECAP 448.copy
[divya@localhost divya]$
```

grep: Searches for a String

The grep utility searches through one or more files to see whether any contain a specified string of characters. This utility does not change the file it searches but simply displays each line that contains the string.



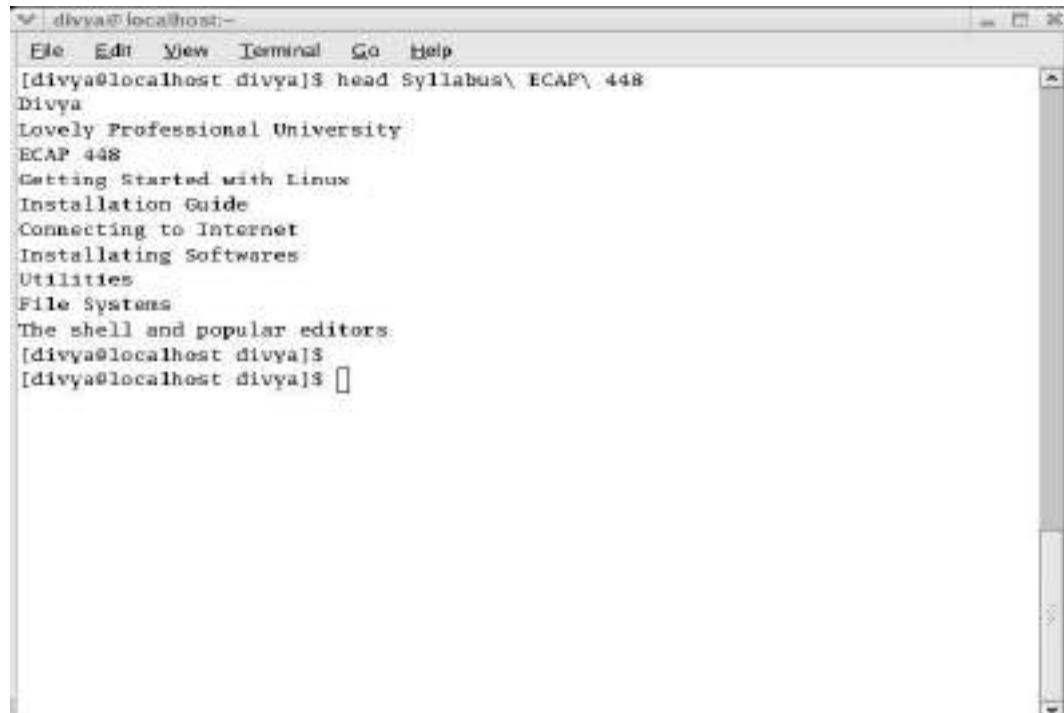
```
[divya@localhost divya]$ ls
counting ECAP 448 ECAP 448.copy Syllabus ECAP 448
[divya@localhost divya]$ cat Syllabus\ ECAP\ 448
Divya
Lovely professional university
ECAP 448
Getting Started with Linux
Installation Guide
Connecting to Internet
Installating Softwares
Utilities
File Systems
The shell and popular editors
the Bourne Shell and TC shell
Linux System Administration
Web Server Configuration
File Server Configuration
Samba servers
Network File System
[divya@localhost divya]$ grep 'Start' Syllabus\ ECAP\ 448
Getting Started with Linux
[divya@localhost divya]$
```



```
[divya@localhost divya]$ cat Syllabus\ ECAP\ 448
Divya
Lovely Professional University
ECAP 448
Getting Started with Linux
Installation Guide
Connecting to Internet
Installating Softwares
Utilities
File Systems
The shell and popular editors
The Bourne Shell and TC shell
Linux System Administration
Web Server Configuration
File Server Configuration
Samba Servers
Network File System
[divya@localhost divya]$ grep 'Start' Syllabus\ ECAP\ 448
Getting Started with Linux
[divya@localhost divya]$ grep 'Server' Syllabus\ ECAP\ 448
Web Server Configuration
File Server Configuration
Samba Servers
[divya@localhost divya]$
```

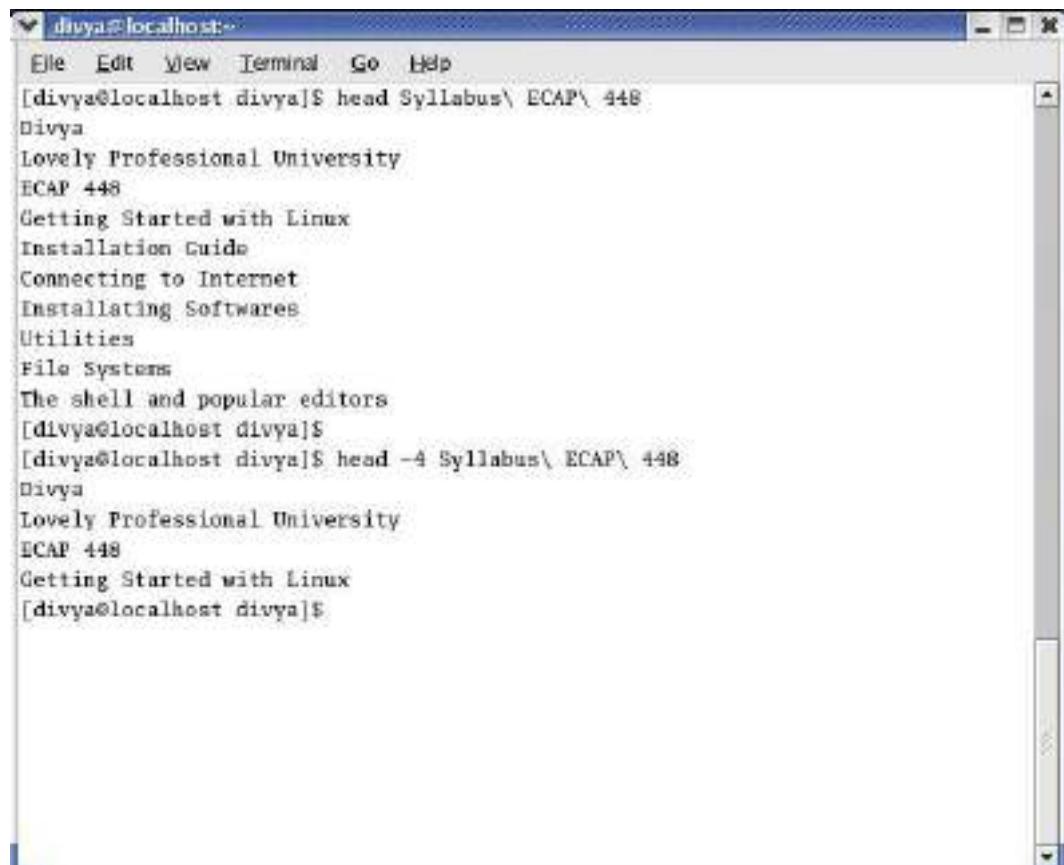
head: Displays the Beginning of a File

By default, the head utility displays the first ten lines of a file. For example, if you have a file named months that lists the 12 months of the year in calendar order, one to a line, then head displays Jan through Oct.



```
[divya@localhost divya]$ head Syllabus\ ECAP\ 448
Divya
Lovely Professional University
ECAP 448
Getting Started with Linux
Installation Guide
Connecting to Internet
Installating Softwares
Utilities
File Systems
The shell and popular editors
[divya@localhost divya]$
[divya@localhost divya]$
```

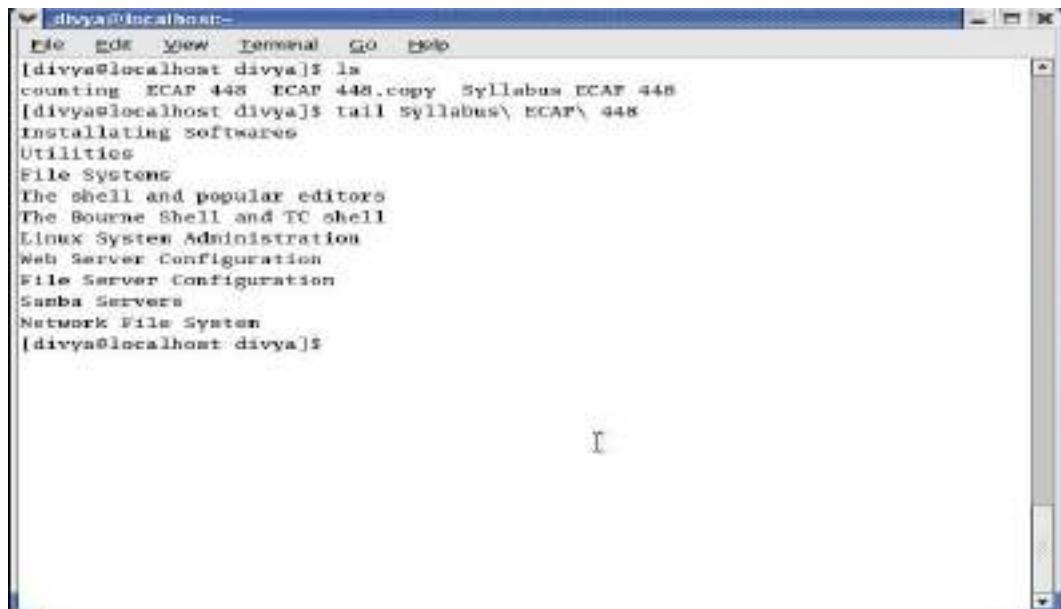
This utility can display any number of lines, so you can use it to look at only the first line of a file, at a full screen, or even more. To specify the number of lines displayed, include a hyphen followed by the number of lines you want head to display.



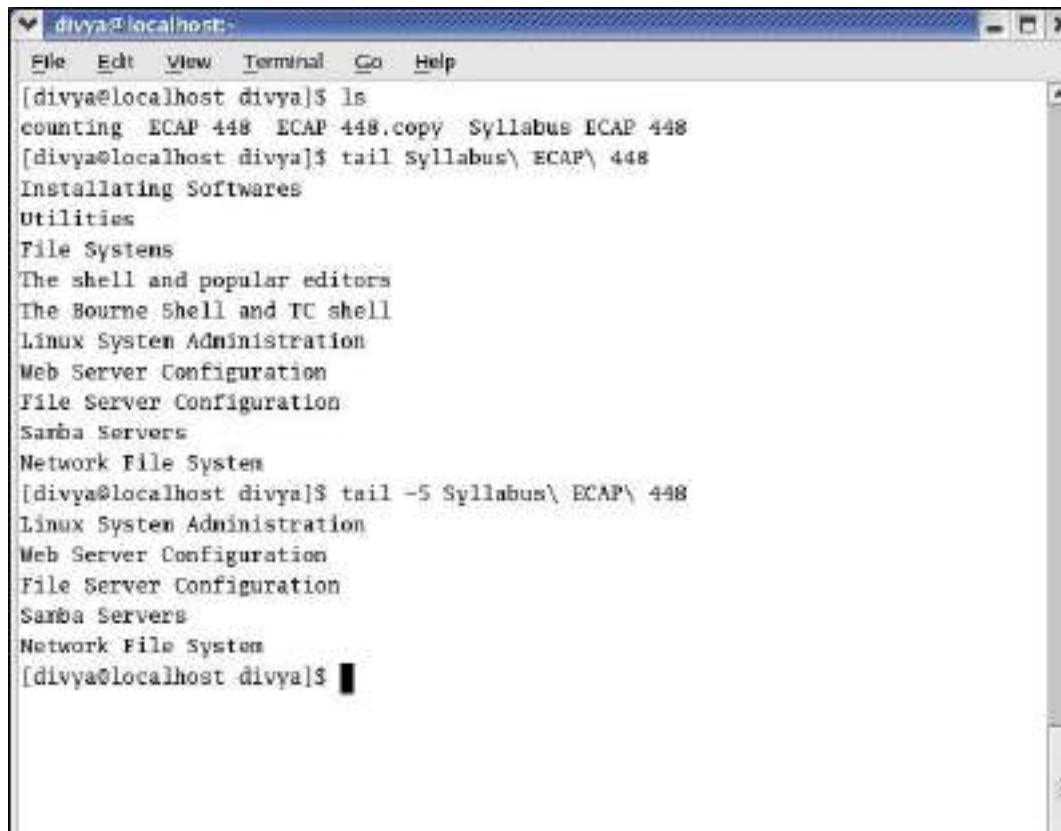
```
[divya@localhost divya]$ head Syllabus\ ECAP\ 448
Divya
Lovely Professional University
ECAP 448
Getting Started with Linux
Installation Guide
Connecting to Internet
Installating Softwares
Utilities
File Systems
The shell and popular editors
[divya@localhost divya]$
[divya@localhost divya]$ head -4 Syllabus\ ECAP\ 448
Divya
Lovely Professional University
ECAP 448
Getting Started with Linux
[divya@localhost divya]$
```

tail: Displays the End of a File

The tail utility is like head but by default displays the last ten lines of a file. Depending on how you invoke it, this utility can display fewer or more than ten lines.



```
divya@localhost:~$ ls
counting ECAP 448 ECAP 448.copy Syllabus ECAP 448
[divya@localhost divya]$ tail Syllabus\ ECAP\ 448
Installing softwares
Utilities
File Systems
The shell and popular editors
The Bourne Shell and TC shell
Linux System Administration
Web Server Configuration
File Server Configuration
Samba Servers
Network File System
[divya@localhost divya]$
```



```
divya@localhost:~$ ls
counting ECAP 448 ECAP 448.copy Syllabus ECAP 448
[divya@localhost divya]$ tail Syllabus\ ECAP\ 448
Installing Softwares
Utilities
File Systems
The shell and popular editors
The Bourne Shell and TC shell
Linux System Administration
Web Server Configuration
File Server Configuration
Samba Servers
Network File System
[divya@localhost divya]$ tail -5 Syllabus\ ECAP\ 448
Linux System Administration
Web Server Configuration
File Server Configuration
Samba Servers
Network File System
[divya@localhost divya]$
```

sort: Displays a File in Order

The sort utility displays the contents of a file in order by lines; it does not change the original file.

Linux and Shell Scripting

```
[divya@localhost divya]$ sort Syllabus\ ECAP\ 448
Connecting to Internet
Divya
ECAP 448
File Server Configuration
File Systems
Getting Started with Linux
Installating Softwares
Installation Guide
Linux System Administration
Lovely Professional University
Network File System
Samba Servers
The Bourne Shell and TC shell
The shell and popular editors
Utilities
Web Server Configuration
[divya@localhost divya]$
```

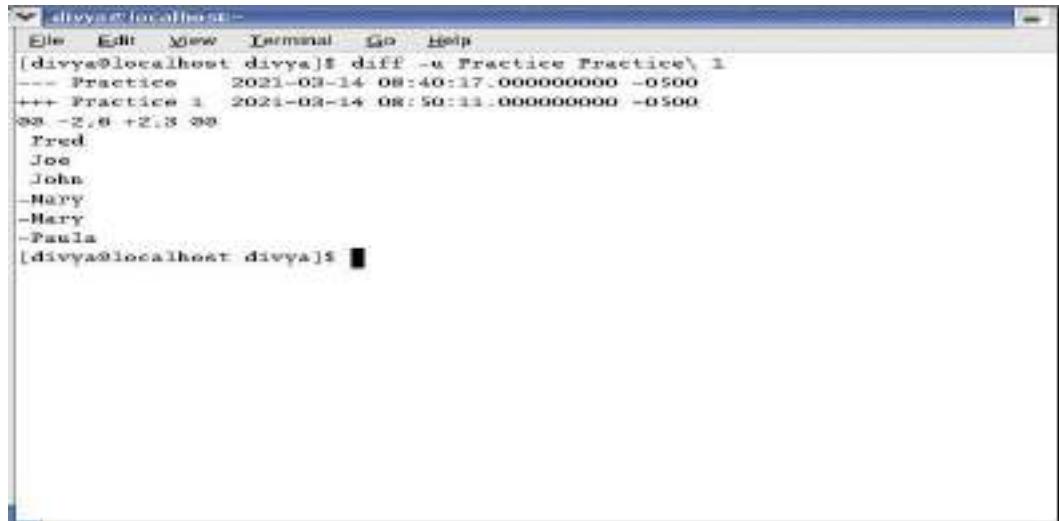
uniq: Removes Duplicate Lines from a File

The uniq (unique) utility displays a file, skipping adjacent duplicate lines, but does not change the original file. If a file contains a list of names and has two successive entries for the same person, uniq skips the duplicate line. If a file is sorted before it is processed by uniq, this utility ensures that no two lines in the file are the same.

```
[divya@localhost divya]$ ls
counting ECAP 448 ECAP 448.copy Practice Syllabus ECAP 448
[divya@localhost divya]$ cat Practice
Cathy
Fred
Joe
John
Mary
Mary
Paula
[divya@localhost divya]$ uniq Practice
Cathy
Fred
Joe
John
Mary
Paula
[divya@localhost divya]$
```

diff: Compares Two Files

The diff (difference) utility compares two files and displays a list of the differences between them. This utility does not change either file; it is useful when you want to compare two versions of a letter or a report or two versions of the source code for a program. The diff utility with the -u (unified output format) option first displays two lines indicating which of the files you are comparing will be denoted by a plus sign (+) and which by a minus sign (-).



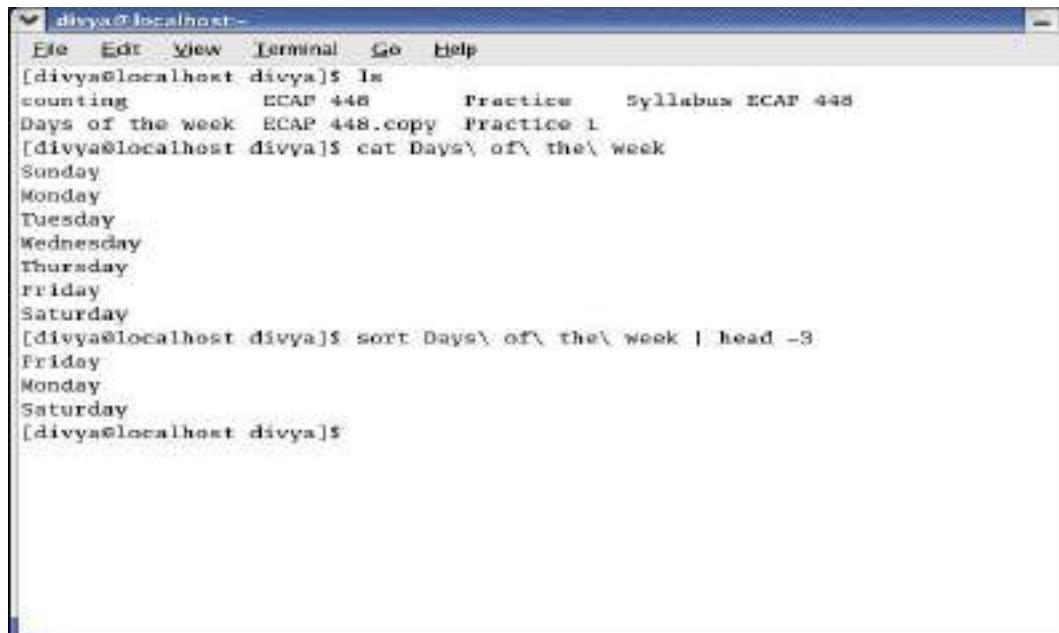
```
(divya@localhost divya]$ diff -u Practice Practice_1
--- Practice 2021-03-14 08:40:17.000000000 -0500
+++ Practice_1 2021-02-14 08:50:33.000000000 -0500
@@ -2,6 +2,3 @@
 Fred
 Joe
 John
-Mary
-Mary
-Paula
[divya@localhost divya]$
```

file: Identifies the Contents of a File

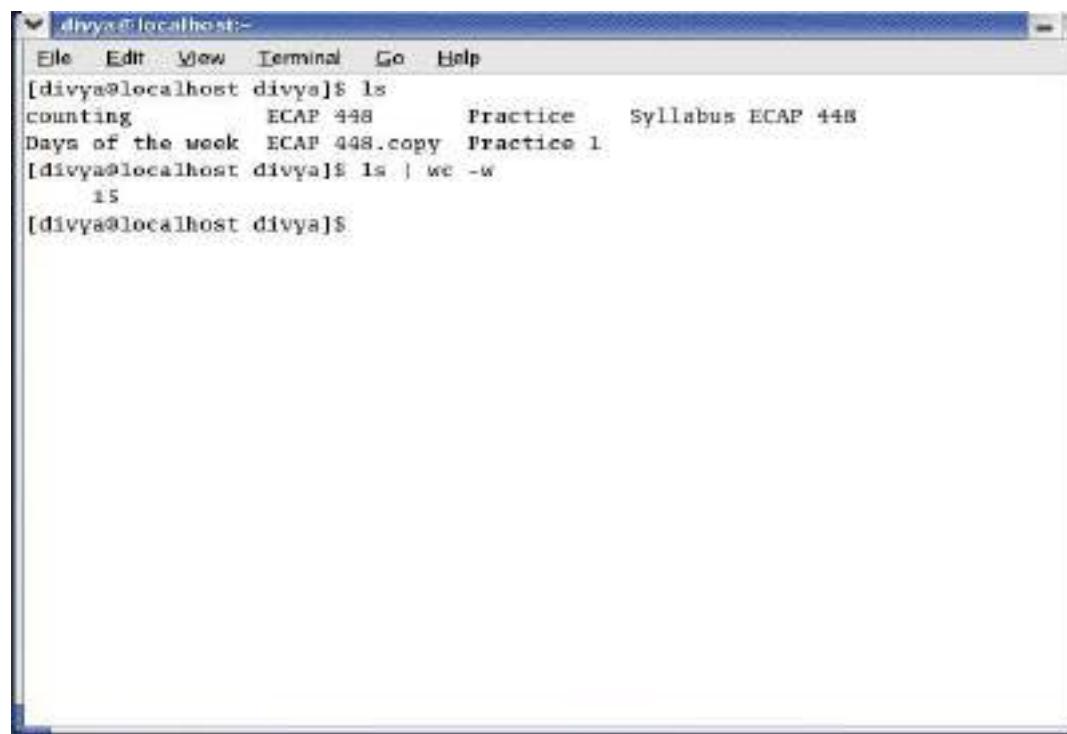
You can use the file utility to learn about the contents of a file without having to open and examine the file yourself.

| (Pipe): Communicates Between Processes

A process is the execution of a command by Linux. Communication between processes is one of the hallmarks of both UNIX and Linux. A pipe (written as a vertical bar [|] on the command line and appearing as a solid or broken vertical line on a keyboard) provides the simplest form of this kind of communication. Simply put, a pipe takes the output of one utility and sends that output as input to another utility.



```
[divya@localhost divya]$ ls
counting      ECAP 448      Practice      Syllabus ECAP 448
Days of the week  ECAP 448.copy  Practice_1
[divya@localhost divya]$ cat Days\ of\ the\ week
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
[divya@localhost divya]$ sort Days\ of\ the\ week | head -3
Friday
Monday
Saturday
[divya@localhost divya]$
```



The screenshot shows a terminal window titled 'divya@localhost:~'. The window has a menu bar with 'File', 'Edit', 'View', 'Terminal', 'Go', and 'Help'. The main area displays the output of the 'ls' command, which lists several files: 'counting', 'ECAP 448', 'Practice', 'Syllabus ECAP 448', 'Days of the week', 'ECAP 448.copy', 'Practice 1', and a file named '15'. Below the file listing, there is a prompt '[divya@localhost divya]\$'.

5.3 Four More Utilities

- echo
- date
- script
- unix2dos

echo: Displays Text

The echo utility copies the characters you type on the command line after echo to the screen. You can also send messages from shell scripts to the screen.



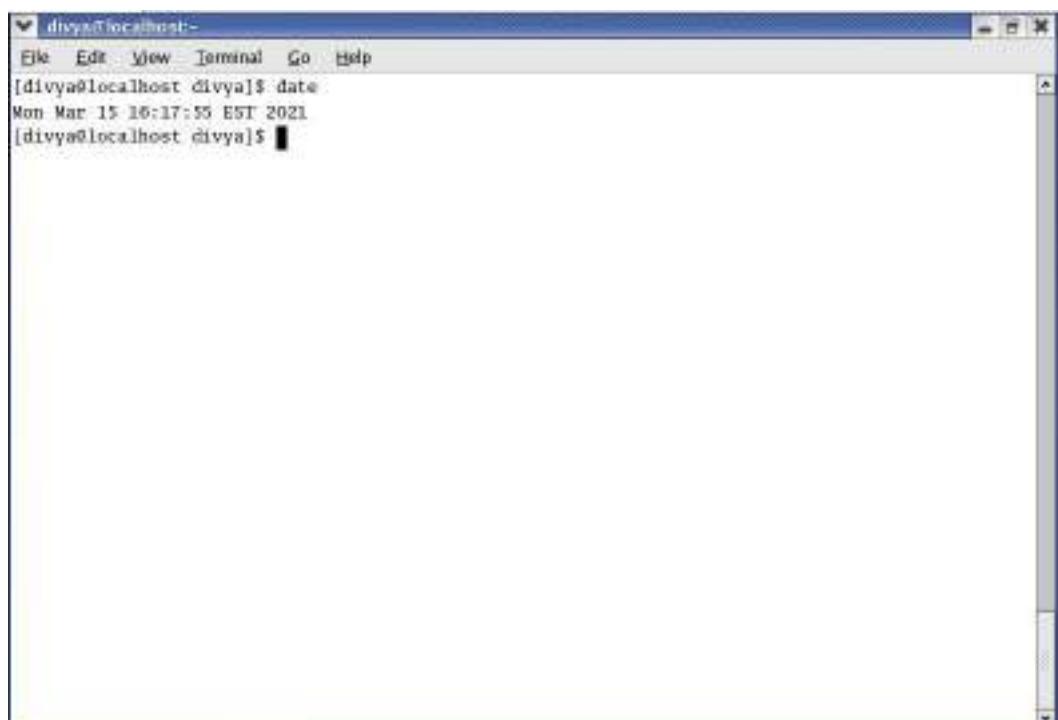
The screenshot shows a terminal window titled 'divya@localhost:~'. The window has a menu bar with 'File', 'Edit', 'View', 'Terminal', 'Go', and 'Help'. The main area displays the output of the 'echo' command, which prints the text 'Good Morning Students. We are studying ECAP 448' followed by 'Good Morning Students. We are studying ECAP 448'. Below the text, there is a prompt '[divya@localhost divya]\$'.



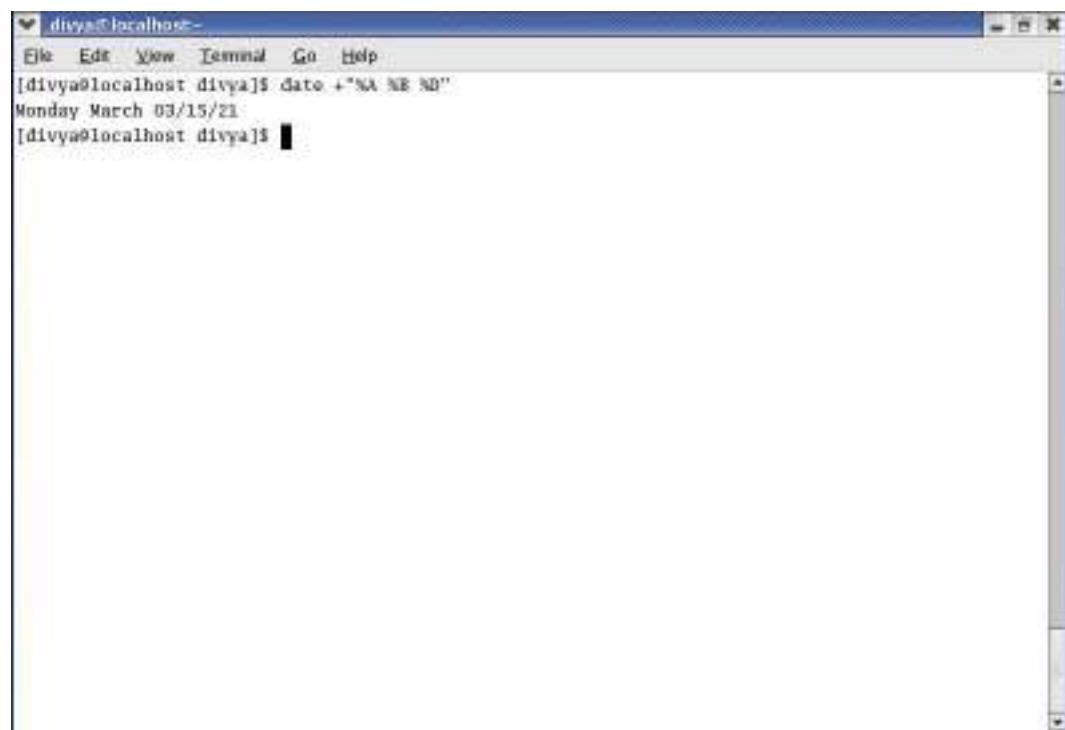
```
divya@localhost:~$ echo 'We are studying the Unit 05- Utilities of course Linux and Shell Scripting' > MyFile1
[divya@localhost divya]$ cat MyFile1
We are studying the Unit 05- Utilities of course Linux and Shell Scripting
[divya@localhost divya]$
```

date: Displays the Time and Date

The date utility displays the current date and time. You can choose the format and select the contents of the output of date.



```
divya@localhost:~$ date
Mon Mar 15 16:17:55 EST 2021
[divya@localhost divya]$
```

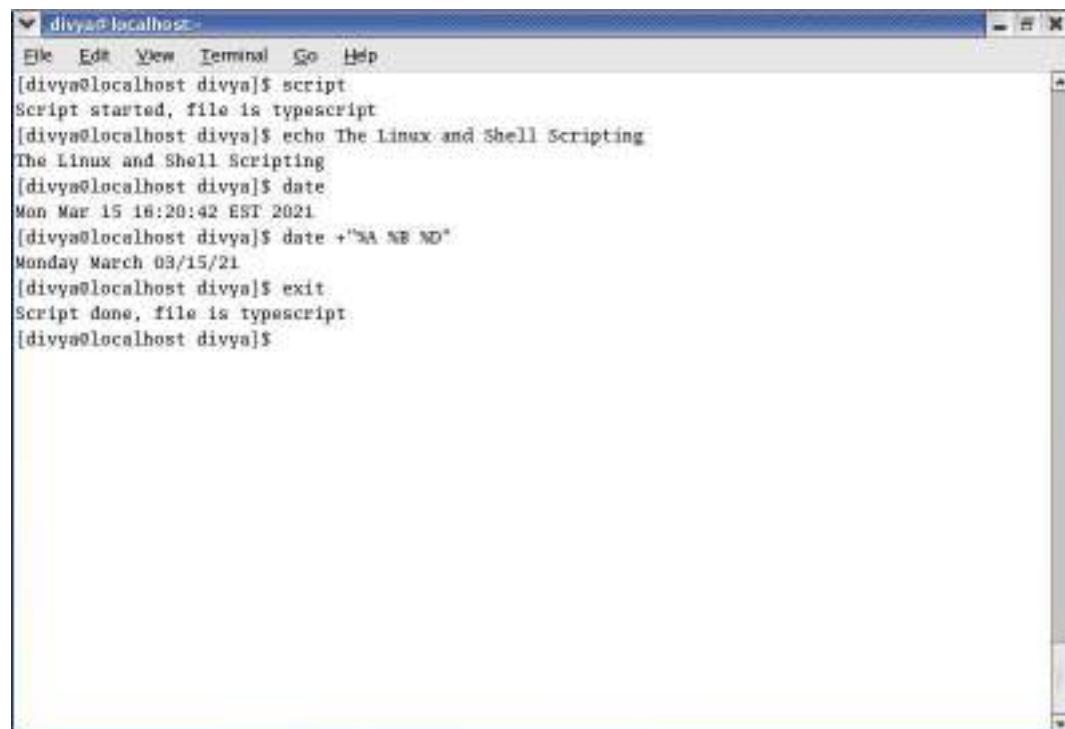


A screenshot of a terminal window titled "divya@localhost". The window has a blue header bar with menu options: File, Edit, View, Terminal, Go, Help. The main area shows a command-line session:

```
[divya@localhost divya]$ date +"%A %B %D"
Monday March 03/15/21
[divya@localhost divya]$
```

script: Records a Shell Session

The script utility records all or part of a login session, including your input and the system's responses. This utility is useful only from character-based devices. By default, script captures the session in a file named typescript. To specify a different filename, follow the script command with a SPACE and the filename



A screenshot of a terminal window titled "divya@localhost". The window has a blue header bar with menu options: File, Edit, View, Terminal, Go, Help. The main area shows a command-line session where the user runs the script command to start recording the session:

```
[divya@localhost divya]$ script
Script started, file is typescript
[divya@localhost divya]$ echo The Linux and Shell Scripting
The Linux and Shell Scripting
[divya@localhost divya]$ date
Mon Mar 15 16:20:42 EST 2021
[divya@localhost divya]$ date +"%A %B %D"
Monday March 03/15/21
[divya@localhost divya]$ exit
Script done, file is typescript
[divya@localhost divya]$
```



The screenshot shows a terminal window titled 'divya@localhost:~'. The window contains the following text:

```
[divya@localhost divya]$ cat typescript
Script started on Mon 15 Mar 2021 04:20:15 PM EST
[divya@localhost divya]$ echo The Linux and Shell Scripting
The Linux and Shell Scripting
[divya@localhost divya]$ date
Mon Mar 15 16:20:42 EST 2021
[divya@localhost divya]$ date +%"%A %B %D"
Monday March 03/15/21
[divya@localhost divya]$ exit

Script done on Mon 15 Mar 2021 04:21:04 PM EST
[divya@localhost divya]$
```

todos/unix2dos: Converts Linux Files to Windows Format

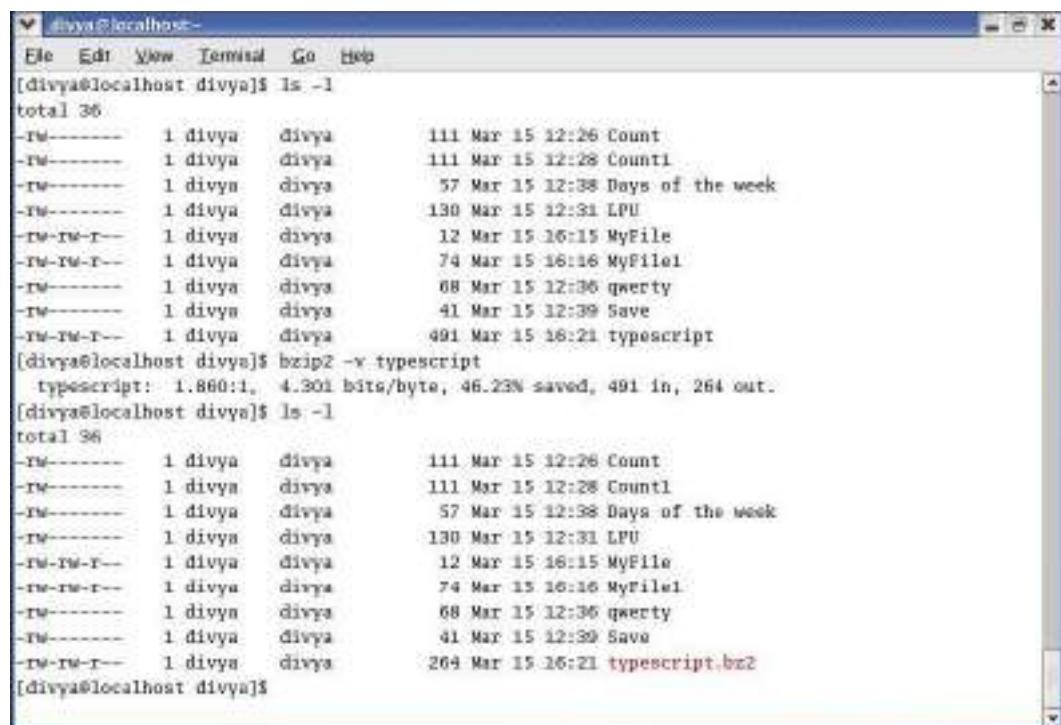
If you want to share a text file you created on a Linux system with someone on a system running Windows, you need to convert the file before the person on the other system can read it easily. The todos (to DOS; part of the tofrodos package) or unix2dos (UNIX to DOS; part of the unix2dos package) utility converts a Linux text file so it can be read on a Windows system. You can use the fromdos (from DOS; part of the tofrodos package) or dos2unix (DOS to UNIX; part of the dos2unix package) utility to convert Windows files so they can be read on a Linux system.

5.4 Compressing and Archiving Files

Large files use a lot of disk space and take longer than smaller files to transfer from one system to another over a network. If you do not need to look at the contents of a large file often, you may want to save it on a CD, DVD, or another medium and remove it from the hard disk. If you have a continuing need for the file, retrieving a copy from another medium may be inconvenient. To reduce the amount of disk space a file occupies without removing the file, you can compress the file without losing any of the information it holds. Similarly, a single archive of several files packed into a larger file is easier to manipulate, upload, download, and email than multiple files. You may download compressed, archived files from the Internet.

bzip2: Compresses a File

The bzip2 utility compresses a file by analyzing it and recoding it more efficiently. The new version of the file looks completely different. In fact, because the new file contains many nonprinting characters, you cannot view it directly. The -v (verbose) option causes bzip2 to report how much it was able to reduce the size of the file.

Linux and Shell Scripting


```
[divya@localhost divya]$ ls -l
total 36
-rw----- 1 divya divya 111 Mar 15 12:26 Count
-rw----- 1 divya divya 111 Mar 15 12:28 Count1
-rw----- 1 divya divya 57 Mar 15 12:38 Days of the week
-rw----- 1 divya divya 130 Mar 15 12:31 LPU
-rw-rw-r-- 1 divya divya 12 Mar 15 16:15 MyFile
-rw-rw-r-- 1 divya divya 74 Mar 15 16:16 MyFile1
-rw----- 1 divya divya 68 Mar 15 12:36 qwerty
-rw----- 1 divya divya 41 Mar 15 12:39 Save
-rw-rw-r-- 1 divya divya 491 Mar 15 16:21 typescript

[divya@localhost divya]$ bzip2 -v typescript
typescript: 1.880:1, 4.301 bits/byte, 46.23% saved, 491 in, 264 out.

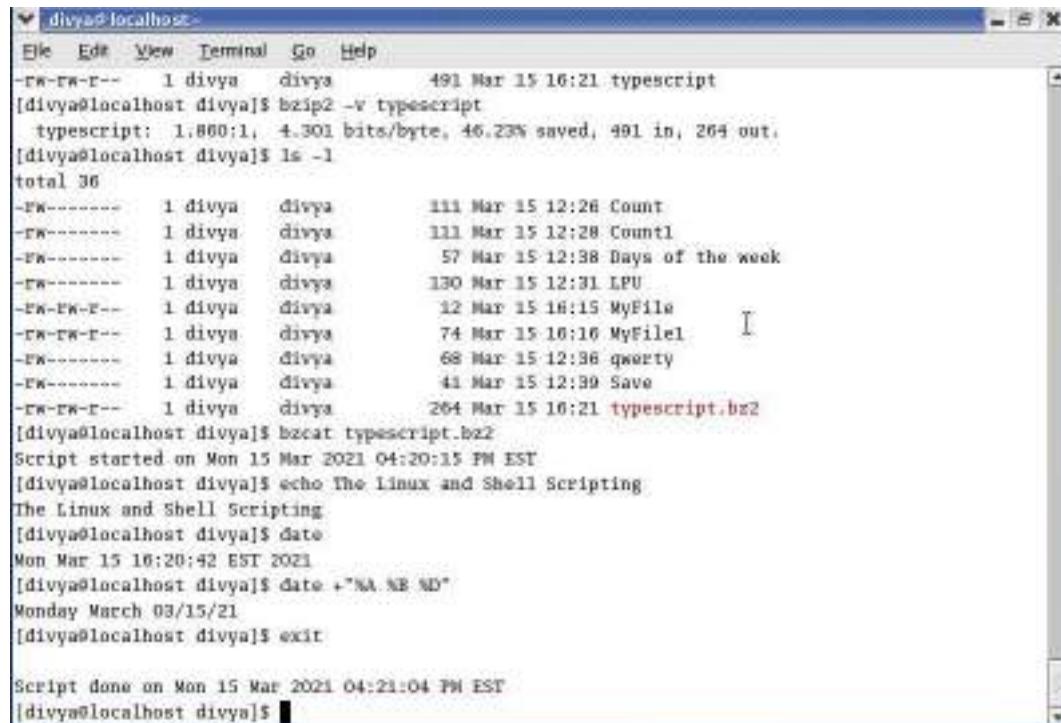
[divya@localhost divya]$ ls -l
total 36
-rw----- 1 divya divya 111 Mar 15 12:26 Count
-rw----- 1 divya divya 111 Mar 15 12:28 Count1
-rw----- 1 divya divya 57 Mar 15 12:38 Days of the week
-rw----- 1 divya divya 130 Mar 15 12:31 LPU
-rw-rw-r-- 1 divya divya 12 Mar 15 16:15 MyFile
-rw-rw-r-- 1 divya divya 74 Mar 15 16:16 MyFile1
-rw----- 1 divya divya 68 Mar 15 12:36 qwerty
-rw----- 1 divya divya 41 Mar 15 12:39 Save
-rw-rw-r-- 1 divya divya 264 Mar 15 16:21 typescript.bz2

[divya@localhost divya]$
```

The bzip2 utility also renamed the file, appending .bz2 to its name. This naming convention reminds you that the file is compressed; you would not want to display or print it, for example, without first decompressing it.

bunzip2 and bzcat: Decompress a File

The bzcat utility displays a file that has been compressed with bzip2. The equivalent of cat for .bz2 files, bzcat decompresses the compressed data and displays the decompressed data. Like cat, bzcat does not change the source file.



```
[divya@localhost divya]$ ls -l
total 36
-rw-rw-r-- 1 divya divya 491 Mar 15 16:21 typescript

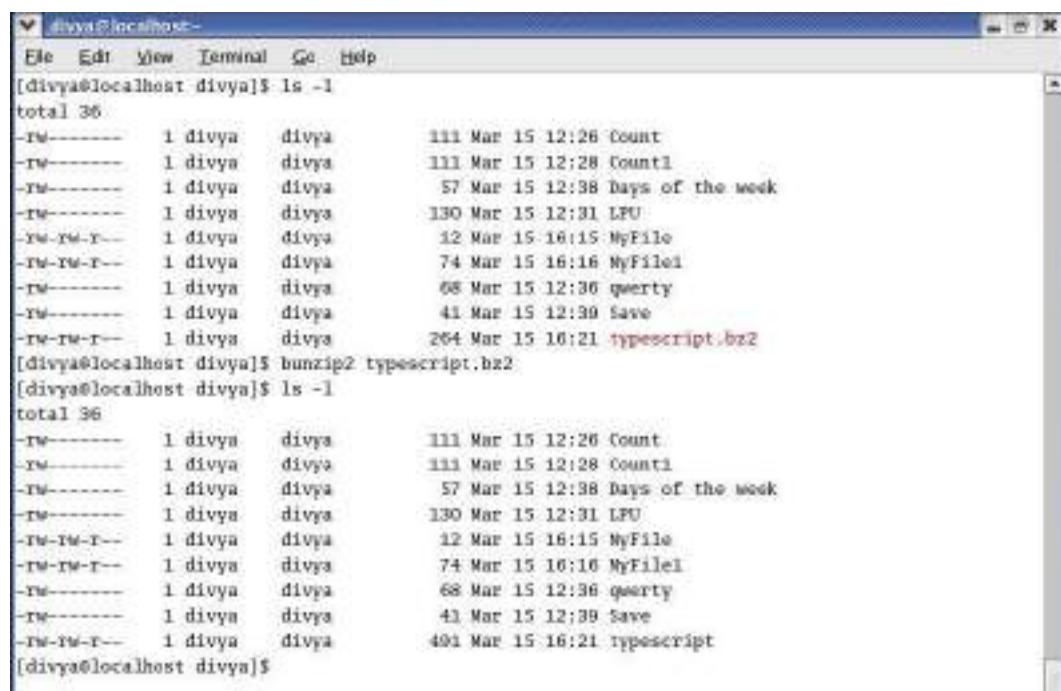
[divya@localhost divya]$ bzip2 -v typescript
typescript: 1.880:1, 4.301 bits/byte, 46.23% saved, 491 in, 264 out.

[divya@localhost divya]$ ls -l
total 36
-rw----- 1 divya divya 111 Mar 15 12:26 Count
-rw----- 1 divya divya 111 Mar 15 12:28 Count1
-rw----- 1 divya divya 57 Mar 15 12:38 Days of the week
-rw----- 1 divya divya 130 Mar 15 12:31 LPU
-rw-rw-r-- 1 divya divya 12 Mar 15 16:15 MyFile
-rw-rw-r-- 1 divya divya 74 Mar 15 16:16 MyFile1
-rw----- 1 divya divya 68 Mar 15 12:36 qwerty
-rw----- 1 divya divya 41 Mar 15 12:39 Save
-rw-rw-r-- 1 divya divya 264 Mar 15 16:21 typescript.bz2

[divya@localhost divya]$ bzcat typescript.bz2
Script started on Mon 15 Mar 2021 04:20:15 PM EST
[divya@localhost divya]$ echo The Linux and Shell Scripting
The Linux and Shell Scripting
[divya@localhost divya]$ date
Mon Mar 15 16:20:42 EST 2021
[divya@localhost divya]$ date +%"%A %B %D"
Monday March 03/15/21
[divya@localhost divya]$ exit

Script done on Mon 15 Mar 2021 04:21:04 PM EST
[divya@localhost divya]$
```

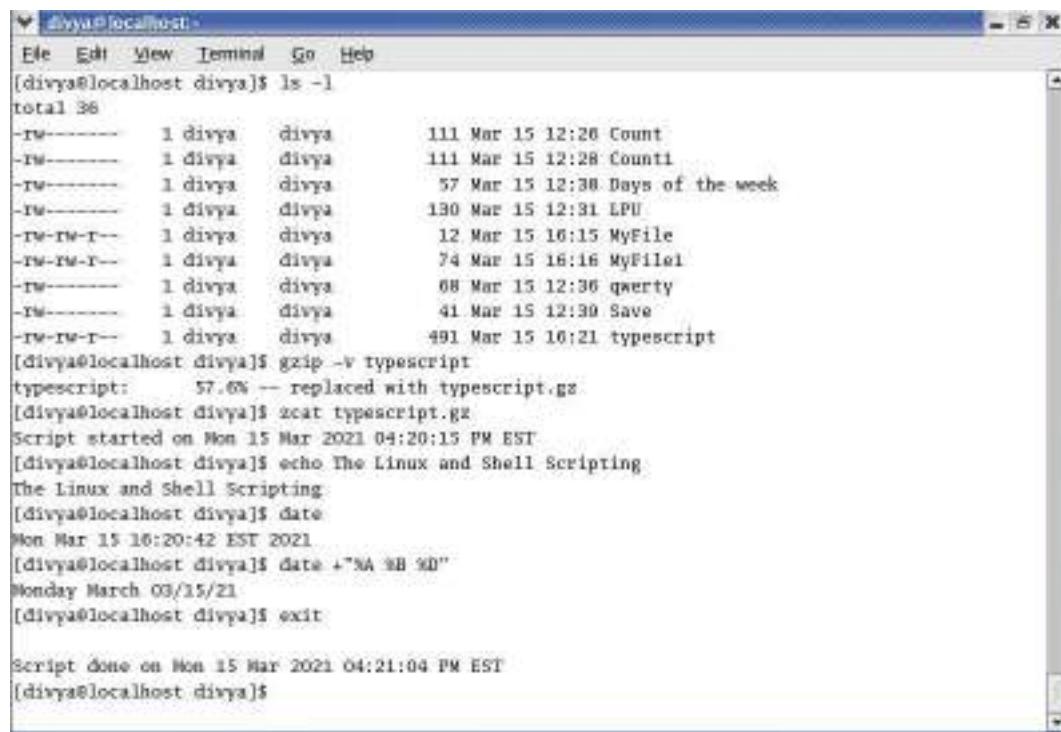
You can use the bunzip2 utility to restore a file that has been compressed with bzip2.



```
[divya@localhost divya]$ ls -l
total 36
-rw----- 1 divya divya 111 Mar 15 12:26 Count
-rw----- 1 divya divya 111 Mar 15 12:28 Count1
-rw----- 1 divya divya 57 Mar 15 12:38 Days of the week
-rw----- 1 divya divya 130 Mar 15 12:31 LPU
-rw-rw-r-- 1 divya divya 12 Mar 15 16:15 MyFile
-rw-rw-r-- 1 divya divya 74 Mar 15 16:16 MyFile1
-rw----- 1 divya divya 68 Mar 15 12:36 qwerty
-rw----- 1 divya divya 41 Mar 15 12:39 Save
-rw-rw-r-- 1 divya divya 264 Mar 15 16:21 typescript.bz2
[divya@localhost divya]$ bunzip2 typescript.bz2
[divya@localhost divya]$ ls -l
total 36
-rw----- 1 divya divya 111 Mar 15 12:26 Count
-rw----- 1 divya divya 111 Mar 15 12:28 Count1
-rw----- 1 divya divya 57 Mar 15 12:38 Days of the week
-rw----- 1 divya divya 130 Mar 15 12:31 LPU
-rw-rw-r-- 1 divya divya 12 Mar 15 16:15 MyFile
-rw-rw-r-- 1 divya divya 74 Mar 15 16:16 MyFile1
-rw----- 1 divya divya 68 Mar 15 12:36 qwerty
-rw----- 1 divya divya 41 Mar 15 12:39 Save
-rw-rw-r-- 1 divya divya 491 Mar 15 16:21 typescript
[divya@localhost divya]$
```

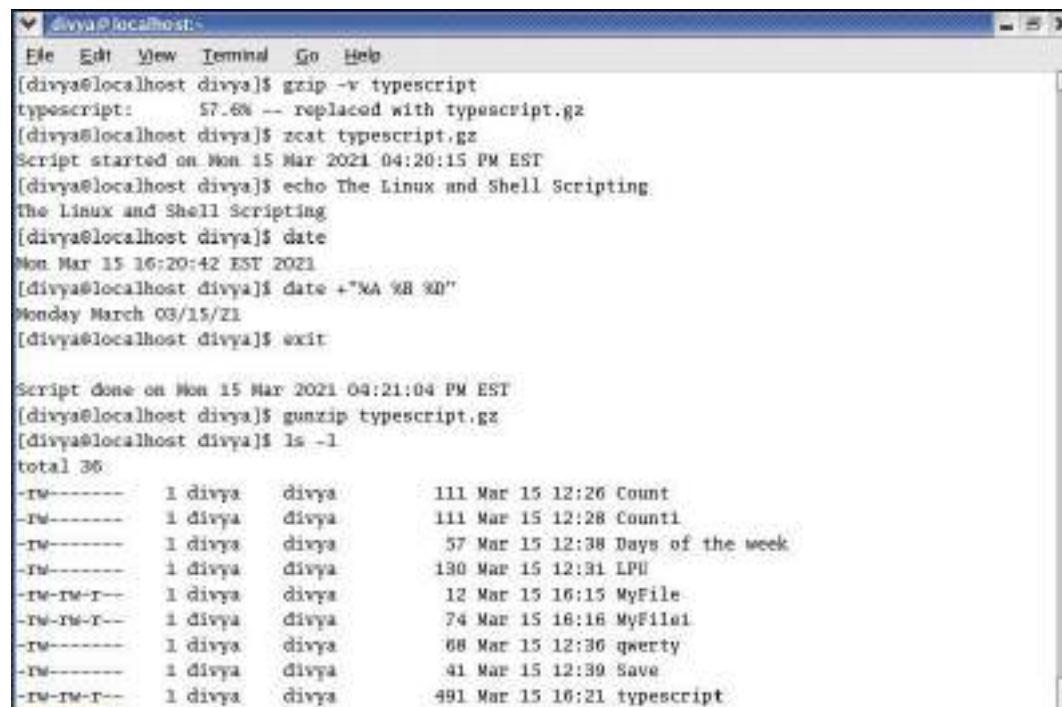
gzip: Compresses a File

The gzip (GNU zip) utility is older and less efficient than bzip2. Its flags and operation are very similar to those of bzip2. A file compressed by gzip is marked by a .gz filename extension. Linux stores manual pages in gzip format to save disk space; likewise, files you download from the Internet are frequently in gzip format. The gunzip utility to restore a file that has been compressed with gzip. zcat decompresses the compressed data and displays the decompressed data.



```
[divya@localhost divya]$ ls -l
total 36
-rw----- 1 divya divya 111 Mar 15 12:26 Count
-rw----- 1 divya divya 111 Mar 15 12:28 Count1
-rw----- 1 divya divya 57 Mar 15 12:38 Days of the week
-rw----- 1 divya divya 130 Mar 15 12:31 LPU
-rw-rw-r-- 1 divya divya 12 Mar 15 16:15 MyFile
-rw-rw-r-- 1 divya divya 74 Mar 15 16:16 MyFile1
-rw----- 1 divya divya 68 Mar 15 12:36 qwerty
-rw----- 1 divya divya 41 Mar 15 12:39 Save
-rw-rw-r-- 1 divya divya 491 Mar 15 16:21 typescript
[divya@localhost divya]$ gzip -v typescript
typescript: 57.6% -- replaced with typescript.gz
[divya@localhost divya]$ zcat typescript.gz
Script started on Mon 15 Mar 2021 04:20:15 PM EST
[divya@localhost divya]$ echo The Linux and Shell Scripting
The Linux and Shell Scripting
[divya@localhost divya]$ date
Mon Mar 15 16:20:42 EST 2021
[divya@localhost divya]$ date +%"%A %B %D"
Monday March 03/15/21
[divya@localhost divya]$ exit

Script done on Mon 15 Mar 2021 04:21:04 PM EST
[divya@localhost divya]$
```

Linux and Shell Scripting


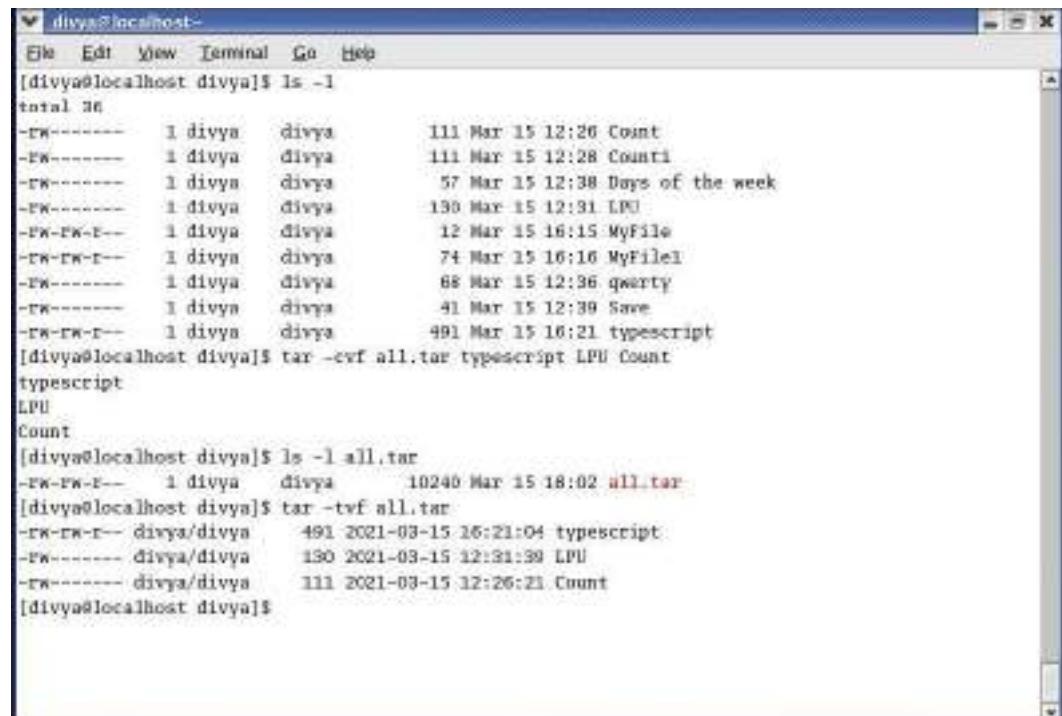
```
[divya@localhost divya]$ gzip -v typescript
typescript:      57.6% -- replaced with typescript.gz
[divya@localhost divya]$ zcat typescript.gz
Script started on Mon 15 Mar 2021 04:20:15 PM EST
[divya@localhost divya]$ echo The Linux and Shell Scripting.
The Linux and Shell Scripting
[divya@localhost divya]$ date
Mon Mar 15 16:20:42 EST 2021
[divya@localhost divya]$ date + "%A %B %d"
Monday March 03/15/21
[divya@localhost divya]$ exit

Script done on Mon 15 Mar 2021 04:21:04 PM EST
[divya@localhost divya]$ gunzip typescript.gz
[divya@localhost divya]$ ls -l
total 36
-rw----- 1 divya divya 111 Mar 15 12:26 Count
-rw----- 1 divya divya 111 Mar 15 12:28 Counti
-rw----- 1 divya divya 57 Mar 15 12:38 Days of the week
-rw----- 1 divya divya 130 Mar 15 12:31 LPU
-rw-rw-r-- 1 divya divya 12 Mar 15 16:15 Myfile
-rw-rw-r-- 1 divya divya 74 Mar 15 16:16 Myfile1
-rw----- 1 divya divya 68 Mar 15 12:36 qwerty
-rw----- 1 divya divya 41 Mar 15 12:39 Save
-rw-rw-r-- 1 divya divya 491 Mar 15 16:21 typescript
```

Do not confuse gzip and gunzip with the zip and unzip utilities. These last two are used to pack and unpack zip archives containing several files compressed into a single file that has been imported from or is being exported to a system running Windows.

tar: Packs and Unpacks Archives

The tar utility performs many functions. Its name is short for tape archive, as its original function was to create and read archive and backup tapes. Today it is used to create a single file (called a tar file, archive, or tarball) from multiple files or directory hierarchies and to extract files from a tar file.



```
[divya@localhost divya]$ ls -l
total 36
-rw----- 1 divya divya 111 Mar 15 12:26 Count
-rw----- 1 divya divya 111 Mar 15 12:28 Counti
-rw----- 1 divya divya 57 Mar 15 12:38 Days of the week
-rw----- 1 divya divya 130 Mar 15 12:31 LPU
-rw-rw-r-- 1 divya divya 12 Mar 15 16:15 Myfile
-rw-rw-r-- 1 divya divya 74 Mar 15 16:16 Myfile1
-rw----- 1 divya divya 68 Mar 15 12:36 qwerty
-rw----- 1 divya divya 41 Mar 15 12:39 Save
-rw-rw-r-- 1 divya divya 491 Mar 15 16:21 typescript
[divya@localhost divya]$ tar -cvf all.tar typescript LPU Count
typescript
LPU
Count
[divya@localhost divya]$ ls -l all.tar
-rw-rw-r-- 1 divya divya 10240 Mar 15 18:02 all.tar
[divya@localhost divya]$ tar -tvf all.tar
-rw-rw-r-- divya/divya 491 2021-03-15 16:21:04 typescript
-rw----- divya/divya 130 2021-03-15 12:31:39 LPU
-rw----- divya/divya 111 2021-03-15 12:26:21 Count
[divya@localhost divya]$
```

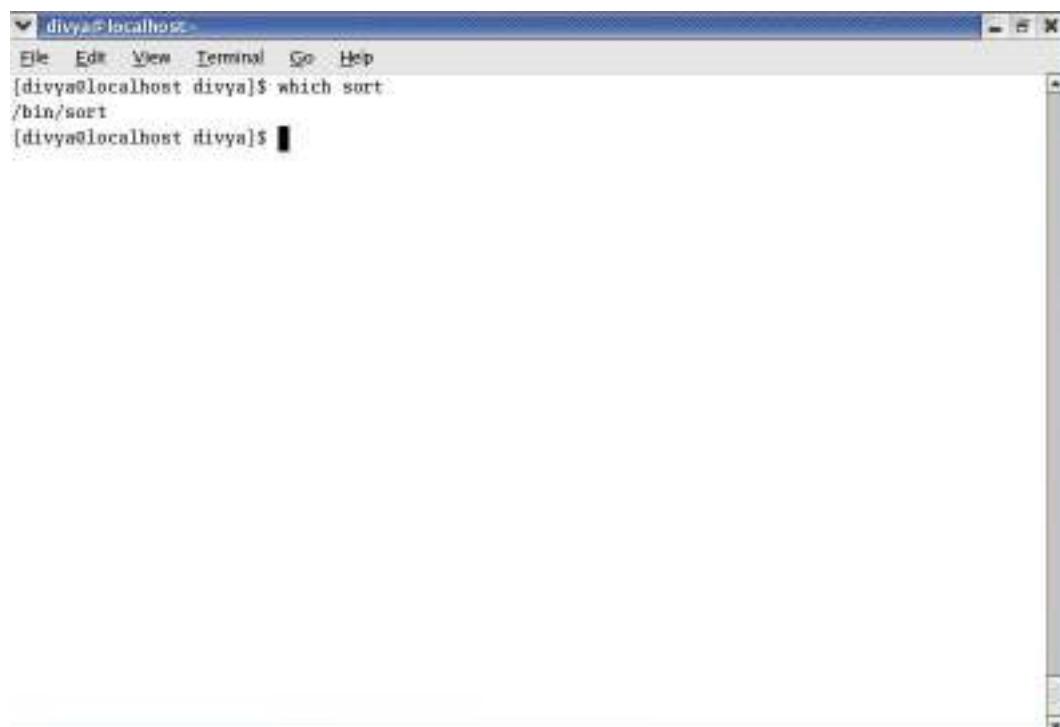
Tar uses the -c (create), -v (verbose), and -f (write to or read from a file) options to create an archive named all.tar from these files. Each line of output displays the name of the file tar is appending to the archive it is creating. The tar utility adds overhead when it creates an archive. The final command uses the -t option to display a table of contents for the archive. You can use bzip2, compress, or gzip to compress tar files, making them easier to store and handle. Many files you download from the Internet will already be in one of these formats. Files that have been processed by tar and compressed by bzip2 frequently have a filename extension of .tar.bz2 or .tbz. Those processed by tar and gzip have an extension of .tar.gz, .tgz, or .gz, whereas files processed by tar and compress use .tar.Z as the extension.

5.5 Locating Commands

The where is and locate utilities can help you find a command whose name you have forgotten or whose location you do not know. When multiple copies of a utility or program are present, which tells you which copy you will run. The slocate utility searches for files on the local system.

Which and where is: Locate a Utility

When you give Linux a command, the shell searches a list of directories for a program with that name and runs the first one it finds. This list of directories is called a search path. If you do not change the search path, the shell searches only a standard set of directories and then stops searching. However, other directories on the system may also contain useful utilities. The which utility locates utilities by displaying the full pathname of the file for the utility. The local system may include several utilities that have the same name. When you type the name of a utility, the shell searches for the utility in your search path and runs the first one it finds. You can find out which copy of the utility the shell will run by using which.



A screenshot of a terminal window titled "divya@localhost". The window has a menu bar with "File", "Edit", "View", "Terminal", "Go", and "Help". The command "which sort" is typed into the terminal, and the output shows the full path "/bin/sort". The terminal window has a blue header bar and a white body with black text.

```
[divya@localhost divya]$ which sort
/bin/sort
[divya@localhost divya]$
```

The where is utility searches for files related to a utility by looking in standard locations instead of using your search path.

Linux and Shell Scripting

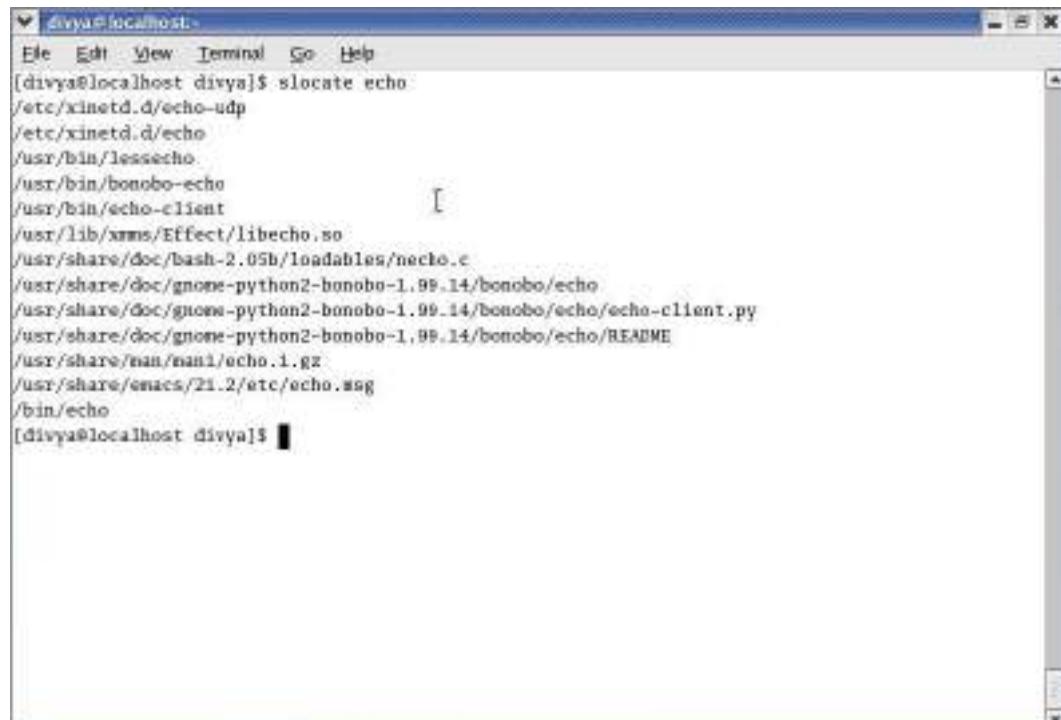


```
[divya@localhost divya]$ whereis sort
sort: /bin/sort /usr/share/man/man1/sort.1.gz /usr/share/man/man3/sort.3pm.gz
[divya@localhost divya]$
```

Where is finds three references to sort: the sort utility file, a sort header file, and the sort man page.

slocate/locate: Searches for a File

The slocate (secure locate) or locate utility searches for files on the local system.



```
[divya@localhost divya]$ slocate echo
/etc/xinetd.d/echo-udp
/etc/xinetd.d/echo
/usr/bin/lessecho
/usr/bin/bonobo-echo
/usr/bin/echo-client
/usr/lib/xmms/Effect/libecho.so
/usr/share/doc/bash-2.05b/loadables/necho.c
/usr/share/doc/gnome-python2-bonobo-1.99.14/bonobo/echo
/usr/share/doc/gnome-python2-bonobo-1.99.14/bonobo/echo/echo-client.py
/usr/share/doc/gnome-python2-bonobo-1.99.14/bonobo/echo/README
/usr/share/man/man1/echo.1.gz
/usr/share/emacs/21.2/etc/echo/msg
/bin/echo
[divya@localhost divya]$
```

Summary:

- When you log in the system, you work in the home directory.

- When you want to view a file that is longer than one screen, you can use either the less utility or the more utility.
- If the destination-file exists before you give a cp command, cp overwrites it. The cp -i (interactive) option prompts you before it overwrites a file.
- If a file contains a list of names and has two successive entries for the same person, uniq skips the duplicate line.
- The diff utility with the -u (unified output format) option first displays two lines indicating which of the files you are comparing will be denoted by a plus sign (+) and which by a minus sign (-).
- You can use the file utility to learn about the contents of a file without having to open and examine the file yourself.
- By default script captures the session in a file named typescript. To specify a different filename, follow the script command with a SPACE and the filename
- The todos (to DOS; part of the tofodos package) or unix2dos (UNIX to DOS; part of the unix2dos package) utility converts a Linux text file so it can be read on a Windows system.
- Linux stores manual pages in gzip format to save disk space; likewise, files you download from the Internet are frequently in gzip format.
- The gunzip utility to restore a file that has been compressed with gzip.
- The whereis and slocate utilities can help you find a command whose name you have forgotten or whose location you do not know.

Keywords

- **Directory:** A directory is a resource that can hold files. On other operating systems, like Windows, a directory is referred to as a folder.
- **ls:** The ls utility lists the names of files which are available.
- **cat:** The cat utility displays the contents of a text file.
- **rm:** The rm (remove) utility deletes a file.
- **cp:** The cp (copy) utility makes a copy of a file. This utility can copy any file, including text and executable program (binary) files.
- **mv:** The mv (move) utility can rename a file without making a copy of it.
- **grep:** The grep utility searches through one or more files to see whether any contain a specified string of characters.
- **sort:** The sort utility displays the contents of a file in order by lines; it does not change the original file.
- **diff:** The diff (difference) utility compares two files and displays a list of the differences between them.
- **uniq:** The uniq (unique) utility displays a file, skipping adjacent duplicate lines, but does not change the original file.
- **Pipe:** A pipe (written as a vertical bar [|] on the command line and appearing as a solid or broken vertical line on a keyboard) provides the simplest form of this kind of communication.
- **echo:** The echo utility copies the characters you type on the command line after echo to the screen.
- **date:** The date utility displays the current date and time.

- **script:** The script utility records all or part of a login session, including your input and the system's responses.
- **bzip2:** The bzip2 utility compresses a file by analyzing it and recoding it more efficiently.
- **bzcat:** The bzcat utility displays a file that has been compressed with bzip2.
- **bunzip2:** You can use the bunzip2 utility to restore a file that has been compressed with bzip2.
- **whereis:** The whereis utility searches for files related to a utility by looking in standard locations instead of using your search path.
- **slocate:** The slocate (secure locate) or locate utility searches for files on the local system.

Self Assessment

1. Which of these utilities is used to convert a Linux text file so that it can be read on a Windows system?
 - A. todos
 - B. dos2unix
 - C. echo
 - D. script
2. Which of these utilities records a shell session?
 - A. echo
 - B. date
 - C. script
 - D. cat
3. Which of these is not used in Linux System?
 - A. zip
 - B. bzip2
 - C. bunzip2
 - D. zcat
4. Which of these utilities is used to compress a file?
 - A. bzip2
 - B. bunzip2
 - C. gunzip
 - D. zcat
5. Which of these utilities records a shell session?
 - A. echo
 - B. date
 - C. script
 - D. cat
6. In less/more, which of these keys should be pressed to display the next screen?

-
- A. SPACE
 - B. ENTER
 - C. CTRL
 - D. ALT
7. Which of these utilities removes duplicate lines from a file?
- A. grep
 - B. uniq
 - C. sort
 - D. None of the above
8. Which option with rm provides the interactive deletion?
- A. -i
 - B. -a
 - C. -b
 - D. -r
9. Which of these utilities are used when you want to view a file that is longer than one page?
- A. more
 - B. less
 - C. Both more and less
 - D. None of the above
10. Which of these utility displays the contents of a text file?
- A. ls
 - B. cat
 - C. rm
 - D. All of the above mentioned
11. Which of these utilities displays the names of files that are available?
- A. ls
 - B. cat
 - C. rm
 - D. All of the above mentioned
12. Which of these utilities compares two files and display the difference between them?
- A. grep
 - B. uniq
 - C. diff
 - D. differ
13. Which symbol is used for representation of a pipe?

- A. #
- B. \$
- C. |
- D. &

14. Which of these represents the basic utilities in Linux?

- A. ls
- B. cat
- C. rm
- D. All of the above mentioned

15. When you log in a Linux system, you work in _____ directory

- A. root
- B. home
- C. var
- D. None of the above

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. A | 2. C | 3. A | 4. A | 5. C |
| 6. A | 7. C | 8. A | 9. C | 10. B |
| 11. A | 12. C | 13. C | 14. D | 15. B |

Review Questions:

1. What is command line utilities? Give some examples and explain the usage of few basic utilities.
2. Which basic utility is used to delete a file? How can we make it interactive?
3. What are pager utilities? How these are useful?
4. Which utilities are used to work with files? Explain any five utilities in detail.
5. Explain the use and syntax of echo, date and script utilities.
6. What is compressing and archiving of files? Explain the utilities used for it.
7. Explain what are locating commands?



Further Readings

Mark G. Sobell, A Practical Guide to Fedora and Red Hat Enterprise Linux, Fifth Edition, Pearson Education



Web Links

<https://www.javatpoint.com/linux-commands>

Unit 06: File Systems

CONTENTS

Objectives

Introduction

6.1 Obtaining User and System Information

6.2 Communicating with Other Users

6.3 The Filesystem

6.4 Pathnames

6.5 Working with Directories

6.6 Linux Access Permissions

6.7 Access Control Lists

6.8 Links

Summary

Keywords

Self Assessment

Answers for Self Assessment

Review Questions

Further Readings

Objectives

After studying this unit, you will be able to:

- Obtain user and system information
- Communicate with other users
- Understand the filesystem
- Understand the pathnames
- Work with directories
- Understand the access permission
- Understand the Access Control Lists and Links

Introduction

There are some utilities that provide information about who is using the system, what those users are doing, and how the system is running. To find out who is using the local system, you can employ one of several utilities that vary in the details they provide and the options they support.

6.1 Obtaining User and System Information

- who
- finger
- w

Linux and Shell Scripting

The oldest utility, who, produces a list of users who are logged in on the local system, the device each person is using, and the time each person logged in. The w and finger utilities show more detail, such as each user's full name and the command line each user is running. You can use the finger utility to retrieve information about users on remote systems if the local system is attached to a network.

who: Lists Users on the System

The who utility displays a list of users who are logged in on the local system.

```
$ who
sam    tty4      2009-07-25 17:18
max    tty2      2009-07-25 16:42
zach   tty1      2009-07-25 16:39
max    pts/4     2009-07-25 17:27 (coffee)
```

The information that who displays is useful when you want to communicate with a user on the local system. If you need to find out which terminal you are using or what time you logged in, you can use the command who am i:

```
$ who am i
max    tty2      2009-07-25 16:42
```

finger: Lists Users on the System

You can use finger to display a list of users who are logged in on the local system. In addition to usernames, finger supplies each user's full name along with information about which device the user's terminal is connected to, how recently the user typed something on the keyboard, when the user logged in, and available contact information. If the user has logged in over the network, the name of the remote system is shown as the user's location.

```
$ finger
Login  Name        Tty      Idle  Login Time  Office ...
max   Max Wild    *tty2      3     Jul 25 16:42
max   Max Wild    pts/4      3     Jul 25 17:27 (coffee)
sam   Sam the Great *tty4      29    Jul 25 17:18
zach  Zach Brill   *tty1      1:07  Jul 25 16:39
```

On systems where security is a concern, the system administrator may disable finger. You can also use finger to learn more about an individual by specifying a username on the command line.

```
$ finger max
Login: max                               Name: Max Wild
Directory: /home/max                         Shell: /bin/tcsh
On since Fri Jul 25 16:42 (PDT) on tty2 (messages off)
On since Fri Jul 25 17:27 (PDT) on pts/4 from coffee
    3 minutes 7 seconds idle
New mail received Sat Jul 25 17:16 2009 (PDT)
    Unread since Sat Jul 25 16:44 2009 (PDT)
Plan:
I will be at a conference in Hawaii all next week.
If you need to see me, contact Zach Brill, x1693.
```

You can also use finger to display a user's username. The finger utility, which is not case sensitive, can search for information on Helen using her first or last name.

```
$ finger HELEN
Login: hls                               Name: Helen Simpson.
...
$ finger simpson
Login: hls                               Name: Helen Simpson.
...
```

w: Lists Users on the System

The w utility displays a list of the users who are logged in on the local system. The information that w displays is useful when you want to communicate with someone at your installation.

```
$ w
17:47:35 up 1 day,  8:10,  4 users,  load average: 0.34, 0.23, 0.26
USER   TTY     FROM      LOGIN@   IDLE    JCPU   PCPU WHAT
sam    tty4    -          17:18    29:14m  0.20s  0.00s vi memo
max    tty2    -          16:42    0.00s  0.20s  0.07s w
zach   tty1    -          16:39    1:07   0.05s  0.00s run_bgdt
max    pts/4   coffee    17:27    3:10m  0.24s  0.24s -bash
```

The first line the w utility displays includes the time of day, the period of time the computer has been running (in days, hours, and minutes), the number of users logged in, and the load average (how busy the system is).

6.2 Communicating with Other Users

write: Sends a Message

These are used to exchange messages and files with other users either interactively or through email. It is enabled as a two-way communication.

The syntax of a write command line is: write username [terminal]. The username is the username of the user you want to communicate with. The terminal is an optional device name that is useful if the user is logged in more than once. You can display the usernames and device names of all users who are logged in on the local system by using who, w, or finger. To establish two-way communication with another user, you and the other user must each execute write, specifying the other's username as the username. The write utility then copies text, line by line, from one keyboard/display to the other. Sometimes it helps to establish a convention, such as typing o (for

Linux and Shell Scripting

"over") when you are ready for the other person to type and typing oo (for "over and out") when you are ready to end the conversation. When you want to stop communicating with the other user, press CONTROL-D at the beginning of a line. Pressing CONTROL-D tells write to quit, displays EOF (end of file) on the other user's terminal, and returns you to the shell. The other user must do the same.

If the Message from banner appears on your screen and obscures something you are working on, press CONTROL- L or CONTROL- R to refresh the screen and remove the banner. Then you can clean up, exit from your work, and respond to the person who is writing to you. You have to remember who is writing to you, however, because the banner will no longer appear on the screen.

mesg: Denies or Accepts Messages

By default, messages to your screen are blocked. Give the following mesg command to allow other users to send you messages: \$ mesg y

If Max had not given this command before Zach tried to send him a message, Zach might have seen the following message: \$ **write max**

```
write: max has messages disabled
```

You can block messages by entering mesg n. Give the command mesg by itself to display is y (for "yes, messages are allowed") or is n (for "no, messages are not allowed"). If you have messages blocked and you write to another user, write displays the following message because, even if you are allowed to write to another user, the user will not be able to respond to you:

```
$ write max
```

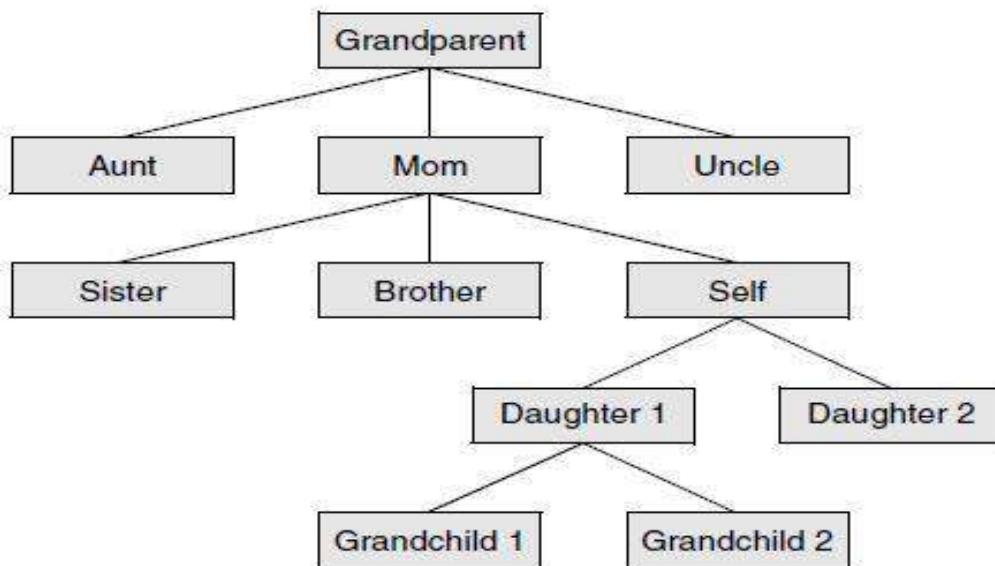
```
write: write: you have write permission turned off.
```

Email

Email enables you to communicate with users on the local system and, if the installation is part of a network, with other users on the network. If you are connected to the Internet, you can communicate electronically with users around the world. Email utilities differ from write in that email utilities can send a message when the recipient is not logged in. In this case the email is stored until the recipient reads it. These utilities can also send the same message to more than one user at a time. Many email programs are available for Linux, including the original character-based mailx program, Mozilla/Thunderbird, pine, mail through emacs, KMail, and evolution. Another popular graphical email program is sylpheed. Two programs are available that can make any email program easier to use and more secure. The procmail program creates and maintains email servers and mailing lists; preprocesses email by sorting it into appropriate files and directories. The GNU Privacy Guard encrypts and decrypts email and makes it almost impossible for an unauthorized person to read.

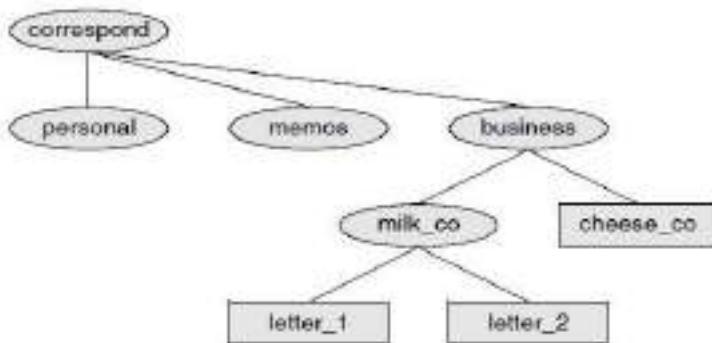
6.3 The Filesystem

A filesystem is a set of data structures that usually resides on part of a disk and that holds directories of files. Filesystems store user and system data that are the basis of users' work on the system and the system's existence. A hierarchical structure frequently takes the shape of a pyramid. One example: a family's lineage. A couple has a child, who may in turn have several children, each of whom may have more children. This hierarchical structure is called a family tree.



Like the family tree it resembles, the Linux filesystem is called a tree. It consists of a set of connected files. This structure allows you to organize files so you can easily find any one. On a standard Linux system, each user starts with one directory, to which the user can add subdirectories to any desired level. By creating multiple levels of subdirectories, a user can expand the structure as needed. Typically each subdirectory is dedicated to a single subject, such as a person, project, or event. The subject dictates whether a subdirectory should be subdivided further. For example, a secretary's subdirectory named `correspond`.

 Example: A secretary's directories



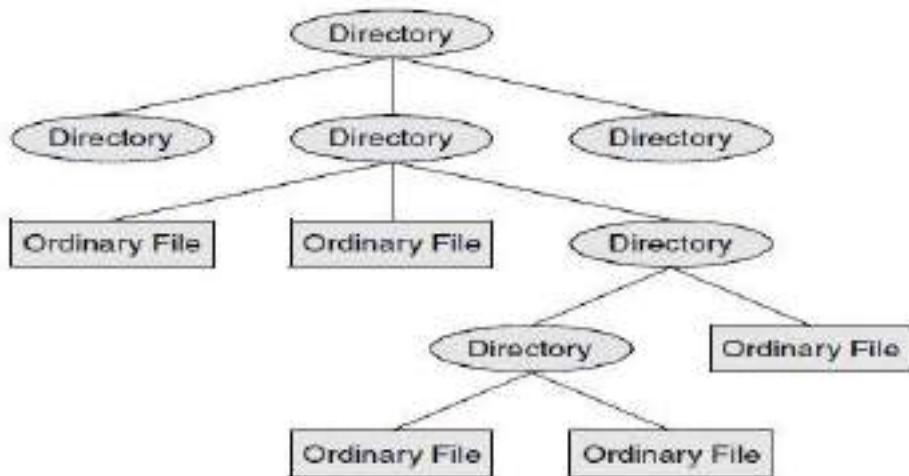
This directory contains three subdirectories: `business`, `memos`, and `personal`. The `business` directory contains files that store each letter the secretary types.

Strength of Linux File System

One major strength of the Linux filesystem is its ability to adapt to users' needs. You can take advantage of this strength by strategically organizing your files so they are most convenient and useful for you.

Directory Files and Ordinary Files

Like a family tree, the tree representing the filesystem is usually pictured upside down, with its root at the top. The tree “grows” downward from the root, with paths connecting the root to each of the other files. At the end of each path is either an ordinary file or a directory file. Ordinary files, or simply files, appear at the ends of paths that cannot support other paths. Directory files, also referred to as directories or folders, are the points that other paths can branch off from. When you refer to the tree, up is toward the root and down is away from the root. Directories directly connected by a path are called parents (closer to the root) and children (farther from the root). A pathname is a series of names that trace a path along branches from one file to another.



Filenames

Every file has a filename. The maximum length of a filename varies with the type of filesystem; Linux supports several types of filesystems. Although most of today’s filesystems allow files with names up to 255 characters long, some filesystems restrict filenames to fewer characters. Choose characters from the following list:

- Uppercase letters (A-Z)
- Lowercase letters (a-z)
- Numbers (0-9)
- Underscore (_)
- Period (.)
- Comma (,)

Like the children of one parent, no two files in the same directory can have the same name. Files in different directories, like the children of different parents, can have the same name. The filenames you choose should mean something. Too often a directory is filled with important files with such unhelpful names as hold1, wombat, and junk, not to mention foo and foobar. Such names are poor choices because they do not help you recall what you stored in a file. The following filenames conform to the suggested syntax and convey information about the contents of the file:

- Correspond
- January
- Davis
- Reports
- 2001
- acct_payable

Unit 06: File Systems

If you keep the filenames short, they are easy to type. The disadvantage of short filenames is that they are typically less descriptive than long filenames. Long filenames enable you to assign descriptive names to files. To help you select among files without typing entire filenames, shells support filename completion. You can use uppercase and/or lowercase letters within filenames, but be careful: Many filesystems are case sensitive. For example, the popular ext family of filesystems and the UFS filesystem are case sensitive, so files named JANUARY, January, and January refer to three distinct files. The FAT family of filesystems (used mostly for removable media) is not case sensitive, so those three filenames represent the same file. The HFS+ filesystem, which is the default OS X filesystem, is case preserving but not case sensitive. Although you can use SPACES within filenames, it is a poor idea. Because a SPACE is a special character, you must quote it on a command line. Use periods or underscores instead of SPACES: joe.05.04.26, new_stuff.

```
$ lpr my\ file  
$ lpr "my file"
```

Filename Extensions

A filename extension is the part of the filename following an embedded period. The filename extensions help describe the contents of the file.

Filename with Extension	Meaning of Extension
compute.c	A C programming language source file
compute.o	The object code file of compute.c
compute	The executable file of compute.c
memo-0410.txt	A text file
memo.pdf	A PDF file, view with xpdf or kpdf
memo.ps	A PostScript file, view with gs or kpdf
memo.z	A file compressed with compress; use uncompress or gunzip to decompress
Memo.tgz or memo.tar.gz	A tar archive of files compressed with gzip
Memo.gz	A file compressed with gzip, view with zcat or decompress with gunzip
Memo.bz2	A file compressed with bzip2; view with bzcat or decompress with bunzip2
Memo.html	A file meant to be viewed using a Web browser, such as firefox

Linux and Shell Scripting

.gif, jpg, jpeg, .bmp, .tif or .tiff

A file containing graphical information, such as picture

Hidden Filenames

A filename that begins with a period is called a hidden filename because ls does not normally display it. The command ls -a displays all filenames, even hidden ones. Names of startup files usually begin with a period so that they are hidden and do not clutter a directory listing. The .plan file is also hidden. Two special hidden entries—a single and double period (. and ..)—appear in every directory.

The Working Directory

While you are logged in on a Linux system, you are always associated with a directory. The directory you are associated with is called the working directory or current directory. Sometimes this association is referred to in a physical sense: “You are in (or working in) the zach directory.” The pwd (print working directory) shell builtin displays the pathname of the working directory.

Your Home Directory

When you first log in, the working directory is your home directory. To display the pathname of your home directory, use pwd just after you log in. Linux home directories are typically located in /home. When used without any arguments, the ls utility displays a list of the files in the working directory. All the files you have created up to this point were created in your home directory.

Startup Files

Startup files, which appear in your home directory, give the shell and other programs information about you and your preferences. Because the startup files have hidden filenames, you must use the ls -a command to see whether one is in your home directory.

6.4 Pathnames

Every file has a pathname, which is a trail from a directory through part of the directory hierarchy to an ordinary file or a directory. Within a pathname, a slash (/) to the right of a filename indicates that the file is a directory file. The file following the slash can be an ordinary file or a directory file.

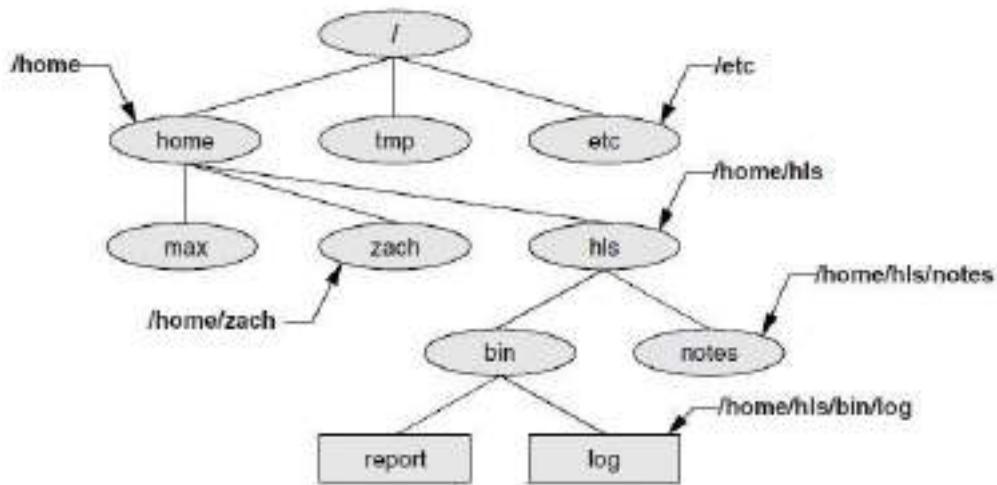
- Absolute Pathnames
- Relative Pathnames

Absolute Pathnames

The root directory of the filesystem hierarchy does not have a name. It is referred to as the root directory. It is represented by a / (slash) standing alone or at the left end of a pathname. An absolute pathname starts with a slash (/), which represents the root directory. The slash is followed by the name of a file located in the root directory. An absolute pathname continues, tracing a path through all intermediate directories, to the file identified by the pathname. String all the filenames in the path together, following each directory with a slash (/). This string of filenames is called an absolute pathname because it locates a file absolutely by tracing a path from the root directory to the file. Using an absolute pathname, you can list or otherwise work with any file on the local system, regardless of the working directory at the time you give the command. For example, Sam can give the following command while working in his home directory to list the files in the /usr/bin directory:

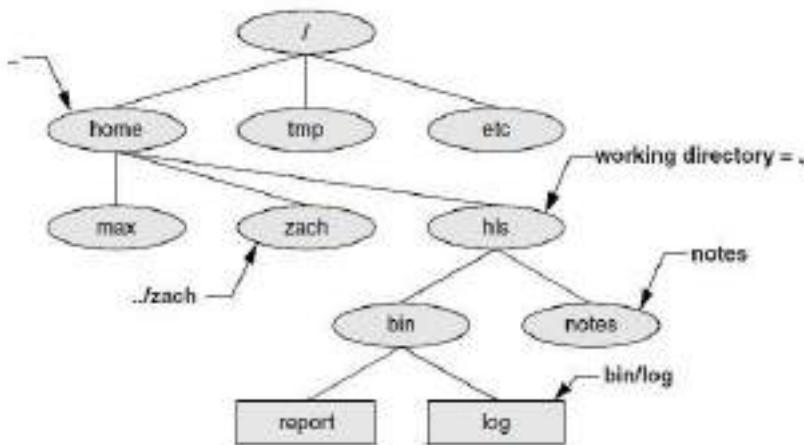
```
$ pwd
/home/sam
```

```
$ ls /usr/bin
7z kwin
7za kwin_ killer_helper
822-date kwin_rules_dialog
```



Relative Pathnames

A relative pathname traces a path from the working directory to a file. The pathname is relative to the working directory. Any pathname that does not begin with the root directory (represented by '/') or a tilde (~) is a relative pathname. Like absolute pathnames, relative pathnames can trace a path through many directories. The simplest relative pathname is a simple filename, which identifies a file in the working directory.



Significance of the Working Directory

To access any file in the working directory, you need only a simple filename. To access a file in another directory, you must use a pathname. Typing a long pathname is tedious and increases the chance of making a mistake. This possibility is less likely under a GUI, where you click filenames or icons. You can choose a working directory for any task to reduce the need for long pathnames.

6.5 Working with Directories

- how to create directories (mkdir),

Linux and Shell Scripting

- switch between directories (cd),
- remove directories (rmdir)
- mv, cp: Move or Copy Files
- mv: Moves a Directory

mkdir: Creates a Directory

The mkdir utility creates a directory. The argument to mkdir becomes the pathname of the new directory.

```
$ pwd
/home/max
$ ls
demo names temp
$ mkdir literature
$ ls
demo literature names temp
$ ls -F
demo literature/ names temp
$ ls literature
$
```

He uses a relative pathname (a simple filename) because he wants the literature directory to be a child of the working directory. Max could have used an absolute pathname to create the same directory: mkdir /home/max/literature or mkdir ~max/literature. There are two ways to create the promo directory as a child of the newly created literature directory.

The first way checks that /home/max is the working directory and uses a relative pathname:

```
$ pwd
/home/max
$ mkdir literature/promo
```

The second way uses an absolute pathname:

```
$ mkdir /home/max/literature/promo
```

cd: Changes to Another Working Directory

The cd (change directory) utility makes another directory the working directory but does not change the contents of the working directory.

```
$ cd /home/max/literature
$ pwd
/home/max/literature
$ cd
$ pwd
/home/max
$ cd literature
$ pwd
/home/max/literature
```

The working directory versus your home directory

The working directory is not the same as your home directory. Your home directory remains the same for the duration of your session and usually from session to session. Immediately after you log in, you are always working in the same directory: your home directory. Unlike your home directory, the working directory can change as often as you like. You have no set working directory, which explains why some people refer to it as the current directory. When you log in and until you change directories using `cd`, your home directory is the working directory.

The . and .. Directory Entries

The `mkdir` utility automatically puts two entries in each directory it creates: a single period (.) and a double period (..). The . is synonymous with the pathname of the working directory and can be used in its place; the .. Is synonymous with the pathname of the parent of the working directory. These entries are hidden because their filenames begin with a period.

rmdir: Deletes a Directory

The `rmdir` (remove directory) utility deletes a directory. You cannot delete the working directory or a directory that contains files other than the . and .. entries. If you need to delete a directory that has files in it, first use `rm` to delete the files and then delete the directory. You do not have to (nor can you) delete the . and .. entries; `rmdir` removes them automatically. The following command deletes the `promo` directory:

\$ `rmdir /home/max/literature/promo`

The `rm` utility has a -r option (`rm -r filename`) that recursively deletes files, including directories, within a directory and also deletes the directory itself.

Although `rm -r` is a handy command, you must use it carefully. Do not use it with an ambiguous file reference such as *. It is shockingly easy to wipe out your entire home directory with a single short command.

mv, cp: Move or Copy Files

You can use this utility to move files from one directory to another (change the pathname of a file) as well as to change a simple filename. When used to move one or more files to a new directory, the `mv` command has this syntax:

\$ `mv existing-file-list directory`

If the working directory is `/home/max`, Max can use the following command to move the files `names` and `temp` from the working directory to the `literature` directory:

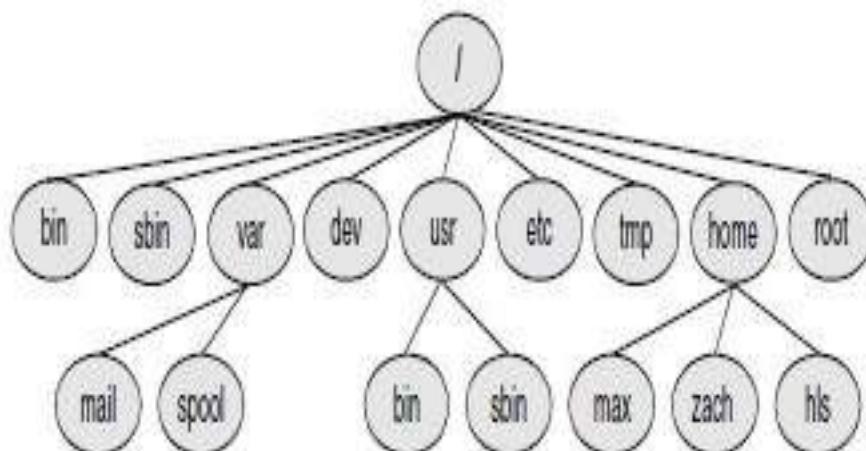
\$ `mv names temp literature`

This command changes the absolute pathnames of the `names` and `temp` files from `/home/max/names` and `/home/max/temp` to `/home/max/literature/names` and `/home/max/literature/temp`, respectively. Like most Linux commands, `mv` accepts either absolute or relative pathnames. The `cp` utility works in the same way as `mv` does, except that it makes copies of the existing-file-list in the specified directory.

mv: Moves a Directory

Just as it moves ordinary files from one directory to another, so mv can move directories. The syntax is similar except that you specify one or more directories, not ordinary files, to move:
mv existing-directory-list new-directory

If new-directory does not exist, the existing-directory-list must contain just one directory name, which mv changes to new-directory (mv renames the directory). Although you can rename directories using mv, you cannot copy their contents with cp unless you use the -r (recursive) option.

A typical FHS-based Linux filesystem structure**Important Standard Directories and Files**

/	Root: The root directory, present in all Linux filesystem structures, is the ancestor of all files in the filesystem.
/bin	Essential command binaries Holds the files needed to bring the system up and run it when it first comes up in single-user or recovery mode.
/boot	Static files of the boot loader Contains all files needed to boot the system.
/dev	Device files Contains all files that represent peripheral devices, such as disk drives, terminals, and printers
/etc	Machine-local system configuration files Holds administrative, configuration, and other system files. One of the most important is /etc/passwd, which contains a list of all users who have permission to use the system.
/etc/opt	Configuration files for add-on software packages kept in /opt
/etc/X11	Machine-local configuration files for the X Window System

Unit 06: File Systems

/home	User home directories
/lib	Shared libraries
/lib/modules	Loadable kernel modules
/mnt	Mount point for temporarily mounting filesystems
/opt	Add-on (optional) software packages
/proc	Kernel and process information virtual filesystem
/root	Home directory for the root account
/sbin	Essential system binaries Utilities used for system administration are stored in /sbin and /usr/sbin. The /sbin directory includes utilities needed during the booting process, and /usr/sbin holds utilities used after the system is up and running.
/sys	Device pseudofilesystem
/tmp	Temporary Files
/usr/games	Games and educational programs
/usr/include	Header files used by C programs
/usr/lib	Libraries
/usr/local	Local hierarchy Holds locally important files and directories that are added to the system. Subdirectories can include bin, games, include, lib, sbin, share, and src.
/usr/sbin	Nonvital system administration binaries See /sbin.
/usr/share	Architecture-independent data Subdirectories can include dict, doc, games, info, locale, man, misc, terminfo, and zoneinfo.
/usr/share/doc	Documentation
/usr/share/info	GNU info system's primary directory

/usr/share/man	Online manuals
/usr/src	Source code
/var	Variable data Files with contents that vary as the system runs are kept in subdirectories under /var. The most common examples are temporary files, system log files, spooled files, and user mailbox files.
/var/log	Log files Contains lastlog (a record of the last login by each user), messages (system messages from syslogd), and wtmp (a record of all logins/logout), among other log files.
/var/spool	Spooled application data Contains anacron, at, cron, lpd, mail, mqueue, samba, and other directories.

6.6 Linux Access Permissions

Linux supports two methods of controlling who can access a file and how they can access it:

- 1) Traditional Linux access permissions
- 2) Access Control Lists (ACLs).

Three types of users can access a file:

- 1) The owner of the file (owner),
- 2) A member of a group that the file is associated with (group),
- 3) Everyone else (other).

Ways to Access an Ordinary File:

- A user can attempt to access an ordinary file in three ways:
- 1) Read from
 - 2) Write to
 - 3) Execute it.

Access Permissions

- ls -l: Displays Permissions
- chmod: Changes Access Permissions
- Setuid and Setgid Permissions
- Directory Access Permissions

ls -l: Displays Permissions:

When you call ls with the -l option and the name of one or more ordinary files, ls displays a line of information about the file.

```
[divya@localhost ~]$ ls
Syllabus
[divya@localhost ~]$ ls -l Syllabus
-rw-r--r-- 1 divya  divya      455 Mar 22 09:36 Syllabus
[divya@localhost ~]$
```

From left to right, the lines that an `ls -l` command displays contain: The type of file (first character), the file's access permissions (the next nine characters), the ACL flag (present if the file has an ACL), the number of links to the file, the name of the owner of the file (usually the person who created the file), the name of the group the file is associated with, the size of the file in characters (bytes), the date and time the file was created or last modified, the name of the file and the type of file is a hyphen (-) because it is an ordinary file.

Character	Meaning
-	Ordinary
b	Block Device
c	Character Device
d	Directory
p	FIFO (names Pipe)
l	Symbolic link

The next three characters specify the access permissions for the owner of the file.

- R indicates read permission,
- w indicates write permission,

Linux and Shell Scripting

- x indicates execute permission.

A – in one of the positions indicates that the owner does not have the permission associated with that position. In a similar manner the next three characters represent permissions for the group. The final three characters represent permissions for other (everyone else). Although execute permission can be allowed for any file, it does not make sense to assign execute permission to a file that contains a document, such as a letter.

chmod: Changes Access Permissions

The Linux file access permission scheme lets you give other users access to the files you want to share yet keep your private files confidential.

- You can allow other users to read from and write to a file.
- You can also allow others only to read from a file.
- Or you can allow others only to write to a file.

A user with root privileges can access any file on the system. Anyone who can gain root privileges has full access to all files, regardless of the file's owner or access permissions. The owner of a file controls which users have permission to access the file and how those users can access it. When you own a file, you can use the chmod (change mode) utility to change access permissions for that file. You can specify symbolic (relative) or numeric (absolute) arguments to chmod.

Symbolic Arguments to chmod


The screenshot shows a terminal window titled "divya@localhost:~". The window contains the following command-line session:

```
[divya@localhost divya]$ ls -l Syllabus
-rw-rw-r-- 1 divya divya 455 Mar 22 09:36 Syllabus
[divya@localhost divya]$ chmod a+w Syllabus
[divya@localhost divya]$ ls -l Syllabus
-rw-rw-rw- 1 divya divya 455 Mar 22 09:36 Syllabus
[divya@localhost divya]$
```



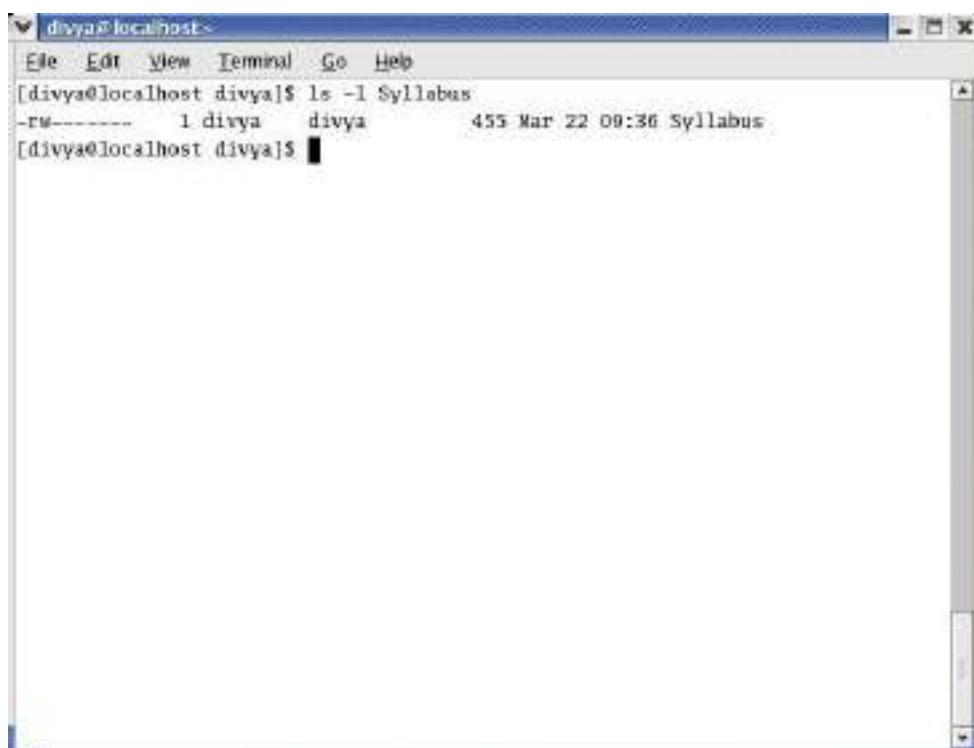
A screenshot of a terminal window titled "divya@localhost". The window shows the following command and its output:

```
[divya@localhost divya]$ ls -l Syllabus
-rw-rw-rw- 1 divya divya 455 Mar 22 09:36 Syllabus
[divya@localhost divya]$ chmod g-rw Syllabus
[divya@localhost divya]$ ls -l Syllabus
-rw----rw- 1 divya divya 455 Mar 22 09:36 Syllabus
[divya@localhost divya]$
```

When using chmod, many people assume that the o stands for owner; it does not. The o stands for other, whereas u stands for owner (user).UGO (user-group-other)

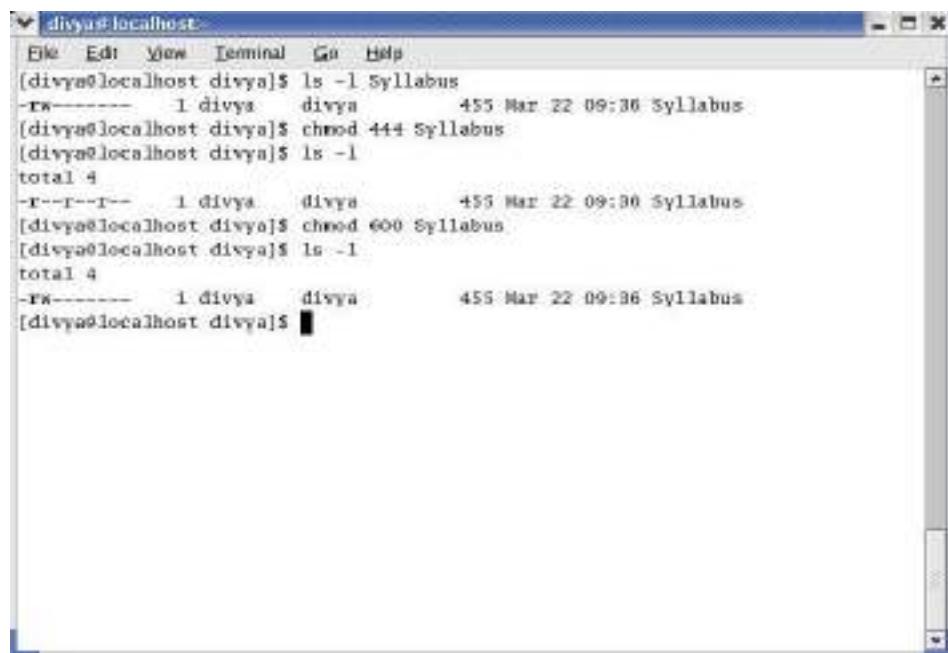
Numeric Arguments to chmod

You can also use numeric arguments to specify permissions with chmod. In place of the letters and symbols specifying permissions used, here numeric arguments comprise three octal digits. The first digit specifies permissions for the owner, the second for the group, and the third for other users. A 1 gives the specified user(s) execute permission, 2 gives write permission, and 4 gives read permission. Construct the digit representing the permissions for the owner, group, or others by ORing (adding) the appropriate values.



A screenshot of a terminal window titled "divya@localhost". The window shows the following command and its output:

```
[divya@localhost divya]$ ls -l Syllabus
-rw----- 1 divya divya 455 Mar 22 09:36 Syllabus
[divya@localhost divya]$
```



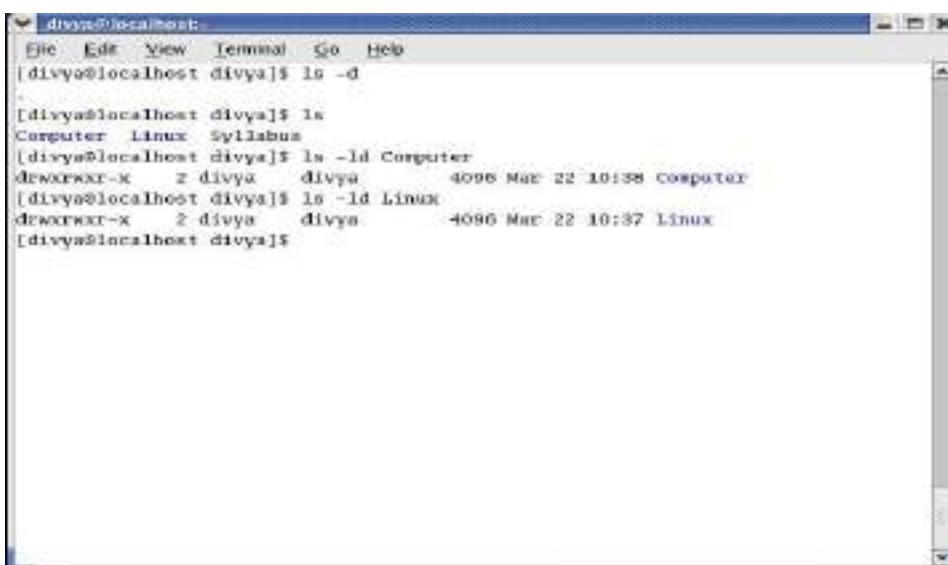
```
[divya@localhost ~]$ ls -l Syllabus
-rw-r--r-- 1 divya divya 455 Mar 22 09:36 Syllabus
[divya@localhost ~]$ chmod 444 Syllabus
[divya@localhost ~]$ ls -l
total 4
-rw-r--r-- 1 divya divya 455 Mar 22 09:36 Syllabus
[divya@localhost ~]$ chmod 600 Syllabus
[divya@localhost ~]$ ls -l
total 4
-rw-r----- 1 divya divya 455 Mar 22 09:36 Syllabus
[divya@localhost ~]$
```

Setuid and Setgid Permissions

When you execute a file that has setuid (set user ID) permission, the process executing the file takes on the privileges of the file's owner. For example, if you run a setuid program that removes all files in a directory, you can remove files in any of the file owner's directories, even if you do not normally have permission to do so. In a similar manner, setgid (set group ID) permission gives the process executing the file the privileges of the group the file is associated with.

Directory Access Permissions

Access permissions have slightly different meanings when they are used with directories. Although the three types of users can read from or write to a directory, the directory cannot be executed. Execute permission is redefined for a directory: It means that you can cd into the directory and/or examine files that you have permission to read from in the directory. It has nothing to do with executing a file. When you have only execute permission for a directory, you can use ls to list a file in the directory if you know its name. You cannot use ls without an argument to list the entire contents of the directory.



```
[divya@localhost ~]$ ls -d
[divya@localhost ~]$ ls
Computer Linux Syllabus
[divya@localhost ~]$ ls -ld Computer
drwxrwxr-x 2 divya divya 4096 Mar 22 10:38 Computer
[divya@localhost ~]$ ls -ld Linux
drwxrwxr-x 2 divya divya 4096 Mar 22 10:37 Linux
[divya@localhost ~]$
```

The d at the left end of the line that ls displays indicates that Linux is a directory. The person has read, write, and execute permissions; members of the group also have read, write and execute permissions; and other users have only read and execute permissions.

6.7 Access Control Lists

The ACLs provide finer-grained control over which users can access specific directories and files than do traditional Linux permissions. Using ACLs you can specify the ways in which each of several users can access a directory or file.

Reduced Performance

Because ACLs can reduce performance, do not enable them on filesystems that hold system files, where the traditional Linux permissions are sufficient. Also be careful when moving, copying, or archiving files: Not all utilities preserve ACLs. In addition, you cannot copy ACLs to filesystems that do not support ACLs.

Rules

An ACL comprises a set of rules. **Rule:** It specifies how a specific user or group can access the file that the ACL is associated with. There are two kinds of rules: **access rules** and **default rules**.

- **Access rule:** It specifies access information for a single file or directory.
- **Default ACL:** It pertains to a directory only; it specifies default access information (an ACL) for any file in the directory that is not given an explicit ACL.

Most utilities do not preserve ACLs

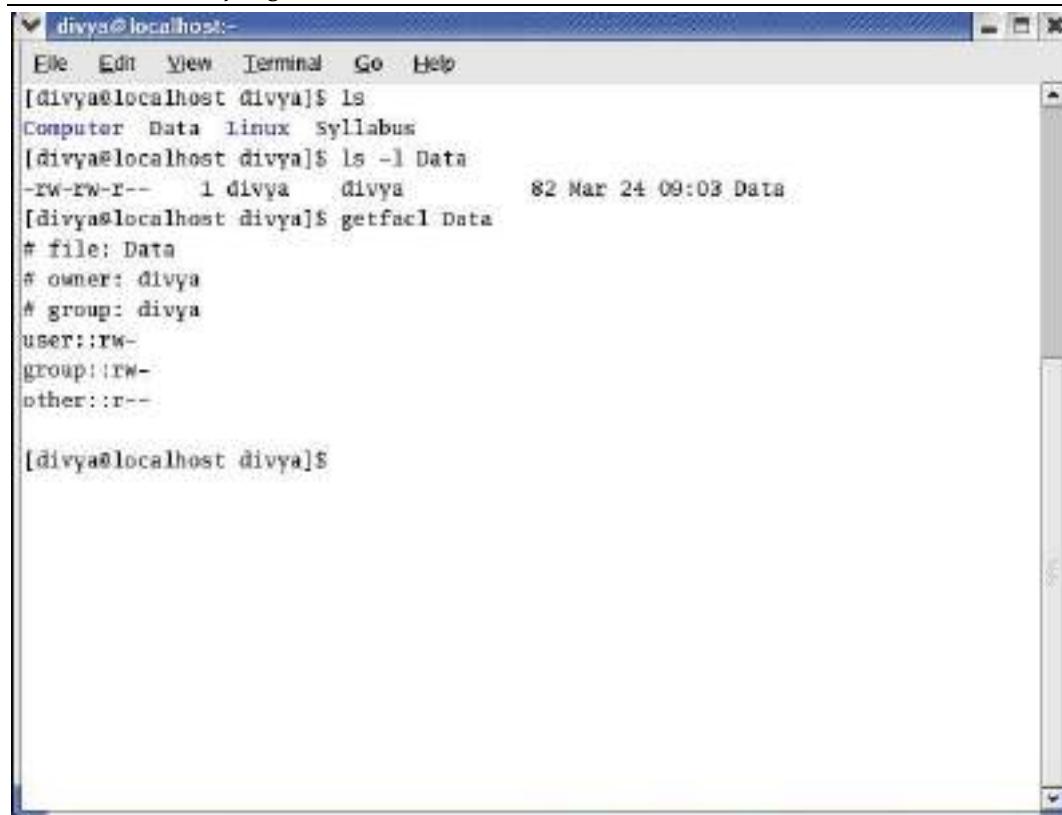
- **cp utility:** When used with the -p (preserve) or -a (archive) option, cp preserves ACLs when it copies files.
- **mv utility:** It also preserves ACLs.
- Other utilities, such as tar, cpio, and dump, do not support ACLs.

Enabling ACLs

Before you can use ACLs you must install the acl software package. Linux officially supports ACLs on ext2, ext3, and ext4 filesystems only, although informal support for ACLs is available on other filesystems. To use ACLs on an ext filesystem, you must mount the device with the acl option. For example, if you want to mount the device represented by /home so you can use ACLs on files in /home, you can add acl to its options list in /etc/fstab:

```
$ grep home /etc/fstab
LABEL=/home /home ext3 defaults,acl 1 2
```

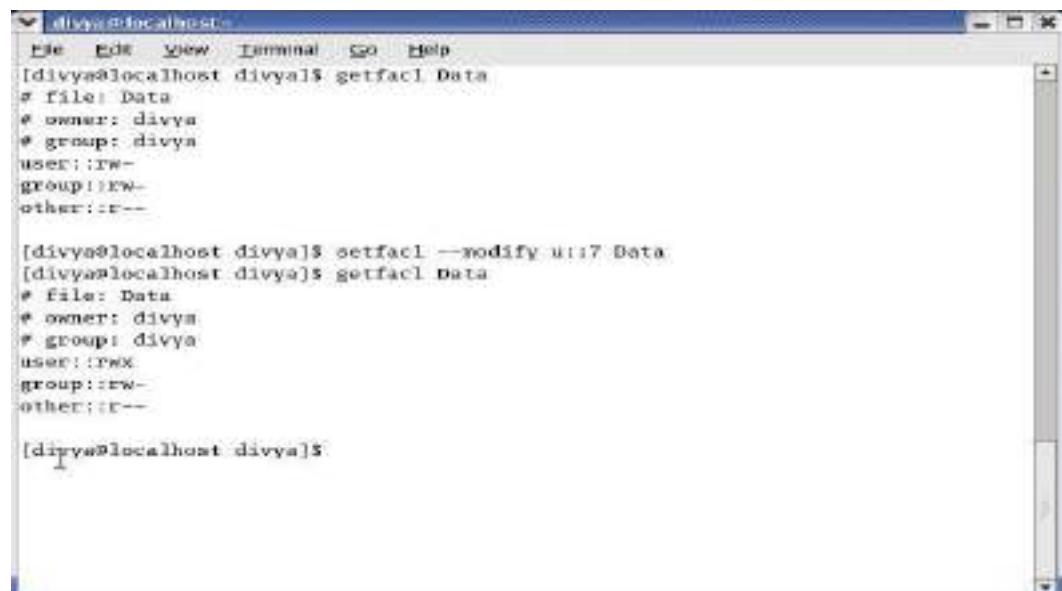
After changing fstab, you need to remount /home before you can use ACLs.

Linux and Shell Scripting


```
[divya@localhost ~]$ ls
Computer Data linux Syllabus
[divya@localhost ~]$ ls -l Data
-rw-rw-r-- 1 divya divya 82 Mar 24 09:03 Data
[divya@localhost ~]$ getfacl Data
# file: Data
# owner: divya
# group: divya
user::rw-
group::rw-
other::r--
```

The terminal window shows the output of the 'ls' command listing a file named 'Data'. Then it runs 'ls -l' to show detailed file information, including permissions (rw-rw-r--), owner ('divya'), and group ('divya'). Finally, it runs 'getfacl Data' which displays the extended ACL for the 'Data' file, showing the same permissions and ownership information.

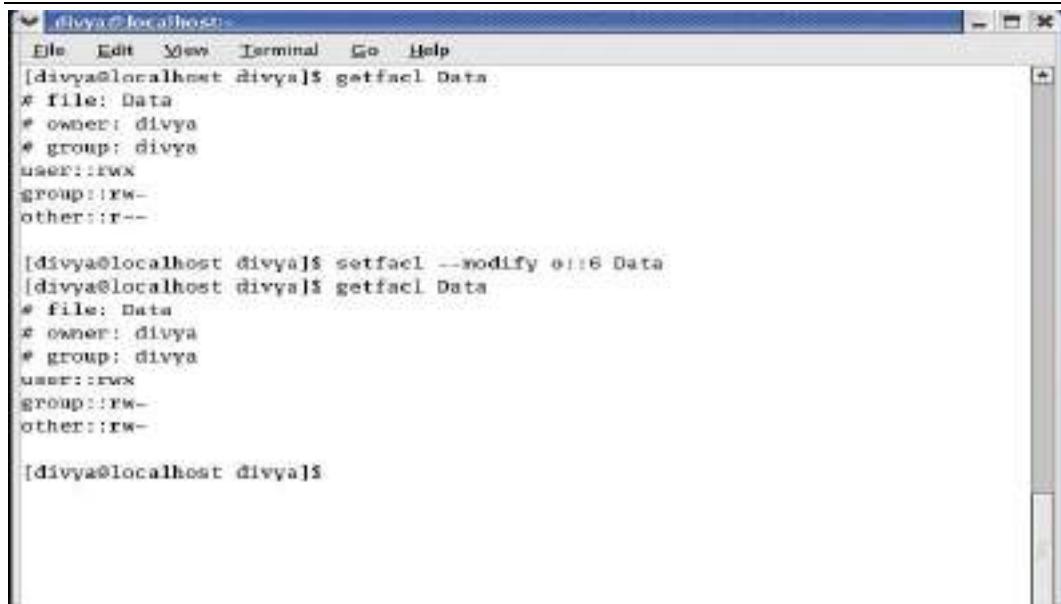
The first three lines of the getfacl output comprise the header; they specify the name of the file, the owner of the file, and the group the file is associated with. In the line that starts with user, the two colons (:) with no name between them indicate that the line specifies the permissions for the owner of the file. Similarly, the two colons in the group line indicate that the line specifies permissions for the group the file is associated with. The two colons following other are there for consistency: No name can be associated with other.



```
[divya@localhost ~]$ getfacl Data
# file: Data
# owner: divya
# group: divya
user::rw-
group::rw-
other::r--
```

```
[divya@localhost ~]$ setfacl --modify u::7 Data
[divya@localhost ~]$ getfacl Data
# file: Data
# owner: divya
# group: divya
user::rwx
group::rw-
other::r--
```

The terminal window shows the initial extended ACL for 'Data' (user:rw-, group:rw-, other:r--). It then runs 'setfacl --modify u::7 Data' to change the user permission to rwx. Finally, it runs 'getfacl Data' again to show the updated extended ACL, where the user permission is now rwx.



```
[divya@localhost divya]$ getfacl Data
# file: Data
# owner: divya
# group: divya
user::rwx
group::rw-
other::r-- 

[divya@localhost divya]$ setfacl --modify o::6 Data
[divya@localhost divya]$ getfacl Data
# file: Data
# owner: divya
# group: divya
user::rwx
group::rw-
other::rw- 

[divya@localhost divya]$
```

Setting up ACL :

1) To add permission for user

`setfacl -m "u:user:permissions" /path/to/file`

2) To add permissions for a group

`setfacl -m "g:group:permissions" /path/to/file`

3) To allow all files or directories to inherit ACL entries from the directory it is within

`setfacl -dm "entry" /path/to/dir`

4) To remove a specific entry

`setfacl -x "entry" /path/to/file`

5) To remove all entries

`setfacl -b path/to/file`

Modifying ACL using setfacl:

To add permissions for a user (user is either the user name or ID): `# setfacl -m "u:user:permissions"`. To add permissions for a group (group is either the group name or ID): `# setfacl -m "g:group:permissions"`. To allow all files or directories to inherit ACL entries from the directory it is within: `# setfacl -dm "entry"`.

View ACL :

To show permissions : `# getfacl filename`

Remove ACL :

If you want to remove the set ACL permissions, use setfacl command with `-b` option.

Setting Default Rules for a Directory

The setfacl command uses the `-d` (default) option to add two default rules to the ACL for dir. These rules apply to all files in the dir directory that do not have explicit ACLs.

Linux and Shell Scripting

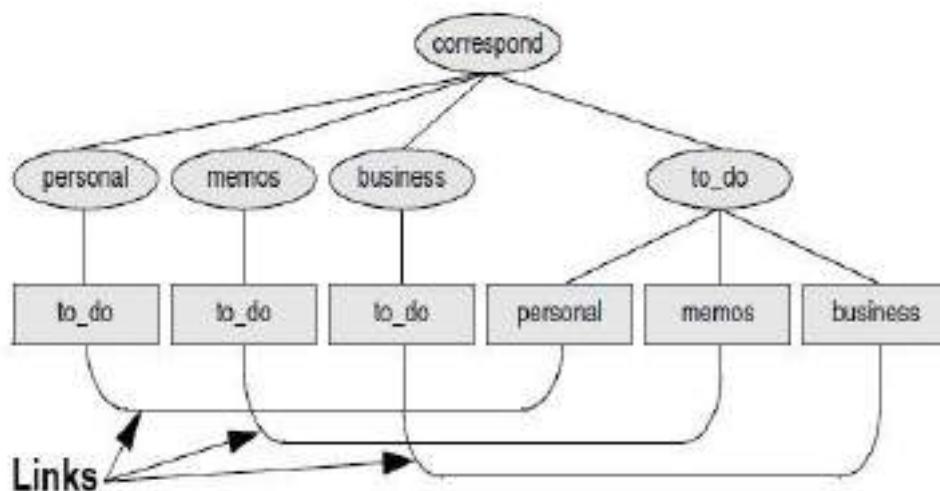
```
[divya@localhost ~]
File Edit View Terminal Go Help
[divya@localhost ~]$
[divya@localhost ~]$ ls
Computer Data Linux Syllabus
[divya@localhost ~]$
[divya@localhost ~]$ ls -ld Computer
drwxrwxr-x 2 divya divya 4096 Mar 22 10:38 Computer
[divya@localhost ~]$ getfacl Computer/
# file: Computer
# owner: divya
# group: divya
user::rwx
group::rwx
other::r-x

[divya@localhost ~]$

```

6.8 Links

A link is a pointer to a file. Every time you create a file by using vim, touch, cp, or any other means, you are putting a pointer in a directory. This pointer associates a filename with a place on the disk. When you specify a filename in a command, you are indirectly pointing to the place on the disk that holds the information you want.

Using links to cross-classify files

Using links

Sharing files can be useful when two or more people are working on the same project and need to share some information. You can make it easy for other users to access one of your files by creating additional links to the file. To share a file with another user, first give the user permission to read from and write to the file. You may also have to change the access permissions of the parent directory of the file to give the user read, write, or execute permission. Once the permissions are appropriately set, the user can create a link to the file so that each of you can access the file from your separate directory hierarchies. A link can also be useful to a single user with a large directory hierarchy. You can create links to cross-classify files in your directory hierarchy, using different classifications for different tasks.

Hard Links

A hard link to a file appears as another file. If the file appears in the same directory as the linked-to file, the links must have different filenames because two files in the same directory cannot have the same name. You can create a hard link to a file only from within the filesystem that holds the file. A hard link is a direct link to the data on disk. This means data can be accessed directly via an original filename or a hard link. Both the original file and the hard link are direct links to the data on disk. The use of a hard link allows multiple filenames to be associated with the same data on disk.

Soft Links

These are also known as symbolic links or symlink. It refers to a symbolic path indicating the abstract location of another file. A symbolic link (also sometimes known as a soft link) does not link directly to the data on disk but to another link to the data on disk. On most operating systems folders may only be linked using a symlink. Symbolic links can link across file systems to link a folder on an external hard drive.

Hard link vs. Soft link in Linux

Hard links cannot link directories and cross file system boundaries. Soft or symbolic links are just like hard links. It allows to associate multiple filenames with a single file. However, symbolic links allows: to create links between directories and can cross file system boundaries. These links behave differently when the source of the link is moved or removed. Symbolic links are not updated. Hard links always refer to the source, even if moved or removed.

Summary

- The oldest utility, who, produces a list of users who are logged in on the local system, the device each person is using, and the time each person logged in. The w and finger utilities show more detail, such as each user's full name and the command line each user is running.
- On systems where security is a concern, the system administrator may disable finger.
- Email enables you to communicate with users on the local system and, if the installation is part of a network, with other users on the network. If you are connected to the Internet, you can communicate electronically with users around the world.
- The popular graphical email program is sylpheed in Linux.
- On a standard Linux system, each user starts with one directory, to which the user can add subdirectories to any desired level.
- One major strength of the Linux filesystem is its ability to adapt to users' needs.
- When you refer to the tree, up is toward the root and down is away from the root.
- If you keep the filenames short, they are easy to type. The disadvantage of short filenames is that they are typically less descriptive than long filenames.

Keywords

- Finger: This utility is used to retrieve information about users on remote systems if the local system is attached to a network.
- Filesystem: A filesystem is a set of data structures that usually resides on part of a disk and that holds directories of files.
- Ordinary files: These are simply files, appear at the ends of paths that cannot support other paths.
- Directory files: These are also referred to as directories or folders, are the points that other paths can branch off from.
- Pathname: A pathname is a series of names that trace a path along branches from one file to another.
- Startup files: These appear in your home directory, give the shell and other programs information about you and your preferences.
- Cd utility: The cd (change directory) utility makes another directory the working directory but does not change the contents of the working directory.

Self Assessment

1. Which of these utilities is used to change to another working directory?
 - A. mkdir
 - B. cd
 - C. mv
 - D. None of the above

2. Any pathname that does not begin with `_` is a relative pathname.
 - A. `/`
 - B. `~`
 - C. Either `/` or `~`
 - D. None of the above

3. When you refer to the tree, _____ is towards the root and _____ is away from the root.
 - A. up, down
 - B. down, up
 - C. left, right
 - D. right, left

4. Which of these files appear at the ends of paths that cannot support other paths?
 - A. Directory files
 - B. Ordinary files
 - C. Base files
 - D. None of the above

5. Which of these builtin is used to display the pathname of the working directory?
 - A. pwd
 - B. work
 - C. path

-
- D. None of the above
6. With a slash (/), we represent
- A. Home directory
 - B. Root directory
 - C. Path directory
 - D. None of the above
7. Which of these utilities is used for communication when the recipient is not logged in?
- A. echo
 - B. email
 - C. cat
 - D. None of the above
8. The utility *who* produces the
- A. List of users who are logged in on the local system
 - B. Device each person is using
 - C. The time each person is logged in
 - D. All of the above mentioned
9. On those systems, where security is concern, the system administrator can disable
-
- A. echo
 - B. finger
 - C. ls
 - D. None of the above
10. Which of these is used for establishing and ending the conversation?
- A. o
 - B. oo
 - C. Both of the above
 - D. None of the above
11. Which of these utilities preserve ACLs?
- A. cp
 - B. mv
 - C. Both cp and mv
 - D. None of the above
12. Which of these links exists in Linux?
- A. Hard links
 - B. Soft links
 - C. Water links

Linux and Shell Scripting

- D. Any of these
13. Which of these files are the points that other paths can branch off from?

- A. Directory files
- B. Ordinary files
- C. Base files
- D. None of the above

14. Which of these builtin is used to display the pathname of the working directory?

- A. pwd
- B. work
- C. path
- D. None of the above

15. Which of these files appear at the ends of paths that cannot support other paths?

- A. Directory files
- B. Ordinary files
- C. Base files
- D. None of the above

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. B | 2. C | 3. A | 4. B | 5. A |
| 6. B | 7. B | 8. D | 9. B | 10. C |
| 11. C | 12. C | 13. A | 14. A | 15. B |

Review Questions

1. Which utilities can be used to obtain system and user information? Explain.
2. How can we communicate with other users in Linux?
3. What is a filesystem? Write the strengths of linux filesystem.
4. What are directory files and ordinary files? Explain their differences with examples.
5. What is a pathname? Write its types.
6. What is a working directory? Explain its significance. Write the utilities which are used to work with directories.
7. What are access permissions? Explain in detail.
8. What is an access control lists? Write its features. How can we set up ACLs?
9. What is a link? Explain the difference between hard links and soft links.



Further Readings

Mark G. Sobell, A Practical Guide to Fedora and Red Hat Enterprise Linux, Fifth Edition, Pearson Education, Inc.



Web Links

<https://www.redhat.com/sysadmin/linux-access-control-lists>

<https://www.geeksforgeeks.org/soft-hard-links-unixlinux/>

Unit 07: The Shell and popular Editors

CONTENTS

- Objectives
- Introduction
- 7.1 The Shell
- 7.2 Standard Input and Standard Output
- 7.3 Redirection
- 7.4 Pipes
- 7.5 Filters
- 7.6 Running a Command in the Background
- 7.7 Filename Generation/Pathname Expansion
- 7.8 Builtins
- 7.9 vim
- 7.10 Modes in vim
- 7.11 Introduction to vim Features
- 7.12 Command Mode
- 7.13 Input Mode
- 7.14 Emacs
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions:
- Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the Command Line
- Understand the Standard Input and Standard Output
- Understand about filename generation and pathname extension
- Understand about builtins
- Know how to use and features of vim
- Understand the command mode and input mode
- Know the difference between emacs and vim
- Understand the functionalities and basic editing commands in emacs

Introduction

The important component of any computer system is an operating system. An operating system is the software that directly manages a system's hardware and resources, like CPU, memory, and storage. The OS sits between applications and hardware and makes the connections between

Linux and Shell Scripting

all your software and the physical resources that do the work. Think about an OS like a car engine. An engine can run on its own, but it becomes a functional car when it's connected with a transmission, axles, and wheels. Without the engine running properly, the rest of the car won't work. An Operating system is made of many components, but its two prime components are - kernel and shell.

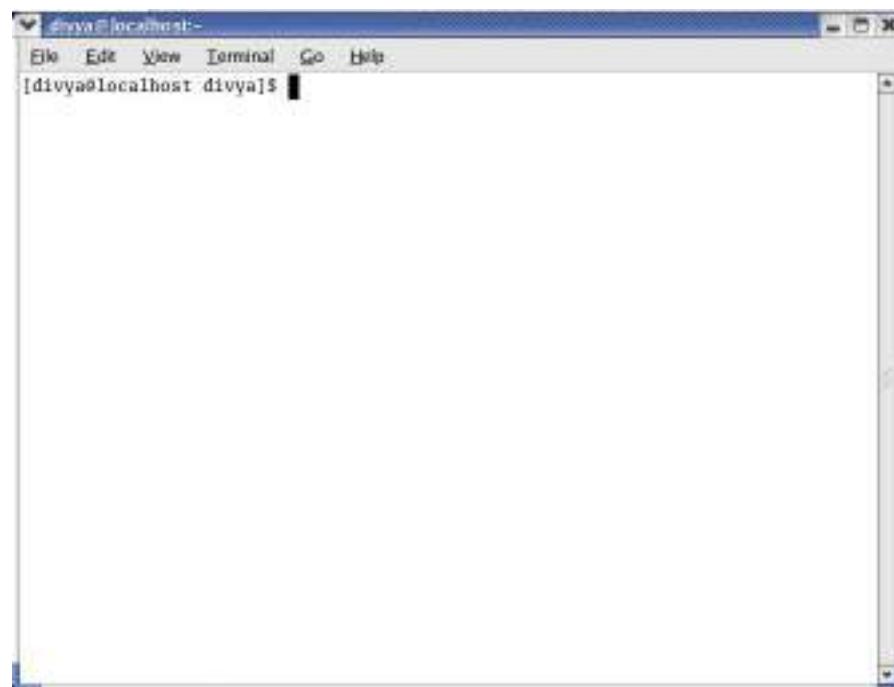
A Kernel is at the nucleus of a computer. It makes the communication between the hardware and software possible. While the Kernel is the innermost part of an operating system, a shell is the outermost one. It is the base component of the OS. Without it, the OS doesn't work. The kernel manages the system's resources and communicates with the hardware. It's responsible for memory, process, and file management. A shell in a Linux operating system takes input from you in the form of commands, processes it, and then gives an output. It is the interface through which a user works on the programs, commands, and scripts. A shell is accessed by a terminal which runs it. When you run the terminal, the Shell issues a command prompt (usually \$), where you can type your input, which is then executed when you hit the Enter key. The output or the result is thereafter displayed on the terminal. The Shell wraps around the delicate interior of an OS protecting it from accidental damage. Hence the name Shell.

7.1 The Shell

A Shell provides you with an interface to the Unix system. It gathers input from you and executes programs based on that input. When a program finish executing, it displays that program's output. Shell is an environment in which we can run our commands, programs, and shell scripts. There are different flavors of a shell, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions. You can cause the shell to execute various types of programs—such as shell scripts, application programs, and programs you have written—in the same way. The line that contains the command, including any arguments, is called the command line. The syntax for a basic command line is

command [arg1] [arg2] ... [argn] RETURN

The prompt, \$, which is called the command prompt, is issued by the shell. While the prompt is displayed, you can type a command. Shell reads your input after you press Enter



Rules for commands

There are some rules for writing the commands. These are:

Unit 07: The Shell and Popular Editors

- One or more SPACES must separate elements on the command line.
- The command is the name of the command, arg1 through argn are arguments, and RETURN is the keystroke that terminates all command lines.
- The brackets in the command-line syntax indicate that the arguments they enclose are optional.
- Not all commands require arguments.

Command Name

Some useful Linux command lines consist of only the name of the command without any arguments. Commands that require arguments typically give a short error message, called a usage message.

Arguments

On the command line each sequence of nonblank characters is called a token or word. An argument is a token, such as a filename, string of text, number, or other object that a command acts on. For example, the argument to a vim or emacs command is the name of the file you want to edit. The following command line shows cp copying the file named temp to temp copy: \$ cp temp temp copy. Arguments are numbered starting with the command itself, which is argument zero. cp argument-0, temp argument-1 and temp copy argument-2. The cp utility requires at least two arguments on the command line.

Options

An option is an argument that modifies the effects of a command. You can frequently specify more than one option, modifying the command in several different ways. Most utilities require you to prefix options with a single hyphen. However, this requirement is specific to the utility and not the shell. GNU program options are frequently preceded by two hyphens in a row. For example, --help generates a (sometimes extensive) usage message.

```
[divya@localhost divya]$ ls
Computer Data Linux Syllabus
[divya@localhost divya]$ ls -r
Syllabus Linux Data Computer
[divya@localhost divya]$
```

Combining options

When you need to use several options, you can usually group multiple single-letter options into one argument that starts with a single hyphen; do not put SPACES between the options. You cannot combine options that are preceded by two hyphens in this way. Specific rules for combining options depend on the program you are running.

Linux and Shell Scripting**Option arguments**

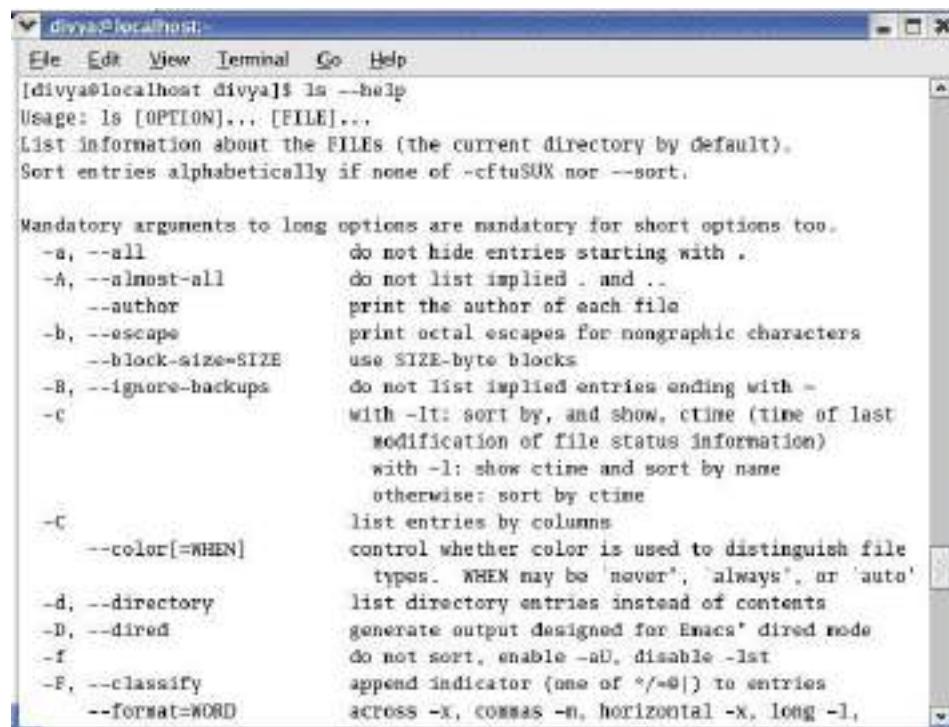
Some utilities have options that themselves require arguments. For example, the gcc utility has a -o option that must be followed by the name you want to give the executable file that gcc generates. Typically, an argument to an option is separated from its option letter by a SPACE.

Arguments that start with a hyphen

Another convention allows utilities to work with arguments, such as filenames, that start with a hyphen. If a file's name is -l, the following command is ambiguous:

```
$ ls -l
```

This command could mean you want ls to display a long listing of all files in the working directory or a listing of the file named -l..



The screenshot shows a terminal window titled 'divya@localhost'. The command 'ls --help' is run, displaying the usage information for the ls command. The output includes the usage string 'Usage: ls [OPTION]... [FILE]...', a brief description of the command, and a detailed list of options and their descriptions. The options listed include -a, -A, -b, -B, -c, -C, -d, -D, -f, -F, and --format=WORD. The descriptions explain what each option does, such as '-a' for not hiding entries starting with '.', '-C' for listing entries by columns, and '--format=WORD' for appending indicators to entries across -x, commas -n, horizontal -x, and long -l.

```
[divya@localhost divya]$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILES (the current directory by default).
Sort entries alphabetically if none of -cftusUX nor --sort.

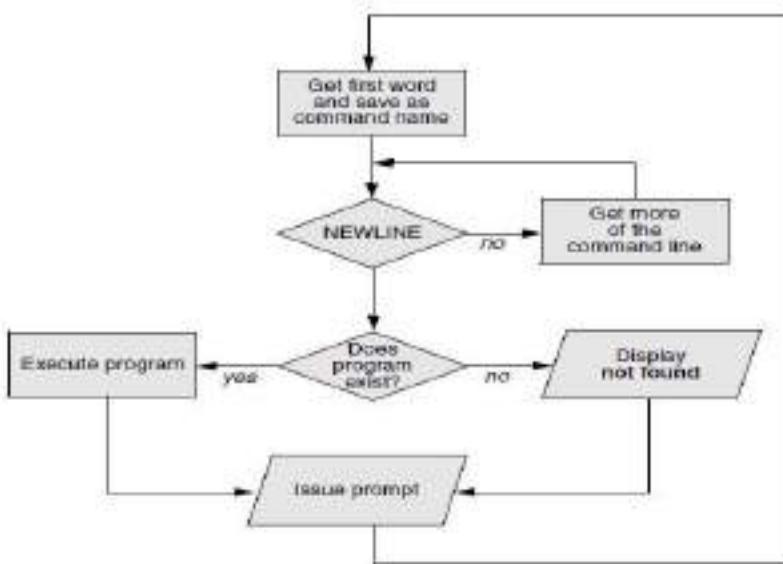
Mandatory arguments to long options are mandatory for short options too.
-a, --all          do not hide entries starting with .
-A, --almost-all   do not list implied . and ..
--author          print the author of each file
-b, --escape        print octal escapes for nongraphic characters
--block-size=SIZE   use SIZE-byte blocks
-B, --ignore-backups  do not list implied entries ending with -
-c                with -lt: sort by, and show, ctime (time of last
                   modification of file status information)
                   with -l: show ctime and sort by name
                   otherwise: sort by ctime
-C                list entries by columns
--color[=WHEN]      control whether color is used to distinguish file
                   types. WHEN may be 'never', 'always', or 'auto'
-d, --directory    list directory entries instead of contents
-D, --dired         generate output designed for Emacs' dired mode
-f                do not sort, enable -al, disable -lst
-F, --classify      append indicator (one of */=@!) to entries
--format=WORD       across -x, commas -n, horizontal -x, long -l,
```

Processing the Command Line

As you enter a command line, the Linux tty device driver (part of the Linux kernel) examines each character to see whether it must take immediate action.

Key	Result
CONTROL-H	To erase a character
CONTROL-U	To kill a line
CONTROL-W	To erase a word

When the character you entered does not require immediate action, the device driver stores the character in a buffer and waits for additional characters. When you press RETURN, the device driver passes the command line to the shell for processing.



Parsing the command line

When the shell processes a command line, it looks at the line as a whole and parses (breaks) it into its component parts. Next, the shell looks for the name of the command. Usually, the name of the command is the first item on the command line after the prompt (argument zero). The shell typically takes the first characters on the command line up to the first blank (TAB or SPACE) and then looks for a command with that name. The command name (the first token) can be specified on the command line either as a simple filename or as a pathname. For example, you can call the ls command in either of the following ways:

```
$ ls
$ /bin/ls
```

Executing the Command Line

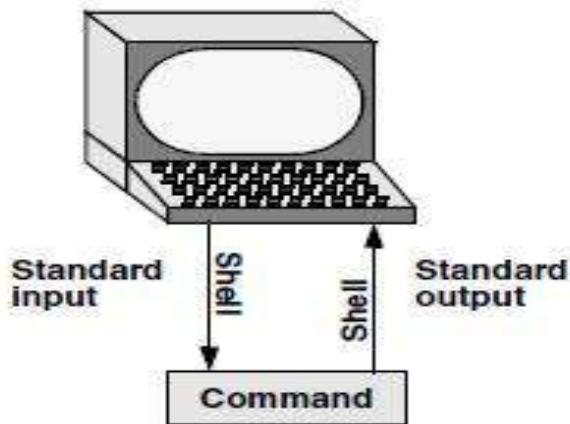
If it finds an executable file with the same name as the command, the shell starts a new process. A process is the execution of a command by Linux. The shell makes each command-line argument, including options and the name of the command, available to the called program. While the command is executing, the shell waits for the process to finish. At this point the shell is in an inactive state called sleep. When the program finishes execution, it passes its exit status to the shell. The shell then returns to an active state (wakes up), issues a prompt, and waits for another command.

Editing the Command Line

You can repeat and edit previous commands and edit the current command line.

7.2 Standard Input and Standard Output

Standard output is a place a program can send information, such as text. The program never "knows" where the information it sends to standard output is going. The information can go to a printer, an ordinary file, or the screen. By default, the shell directs standard output from a command to the screen.



Standard input is a place that a program gets information from. As with standard output the program never “knows” where the information comes from. In addition to standard input and standard output, a running program normally has a place to send error messages: standard error.

The Screen as a File

Linux have an additional type of file: a device file. A device file resides in the Linux file structure, usually in the /dev directory, and represents a peripheral device, such as a screen and keyboard, printer, or disk drive. The device name that the who utility displays after your username is the filename of the screen and keyboard.

The Keyboard and Screen as Standard Input and Standard Output

When you first log in, the shell directs standard output of your commands to the device file that represents the screen. Directing output in this manner causes it to appear on the screen.

cat utility

The cat utility provides a good example of the way the keyboard and screen function as standard input and standard output, respectively. When you use cat, it copies a file to standard output. Because the shell directs standard output to the screen, cat displays the file on the screen. Up to this point cat has taken its input from the filename (argument) you specify on the command line.

A screenshot of a terminal window titled "Terminal". The window contains the following text:

```

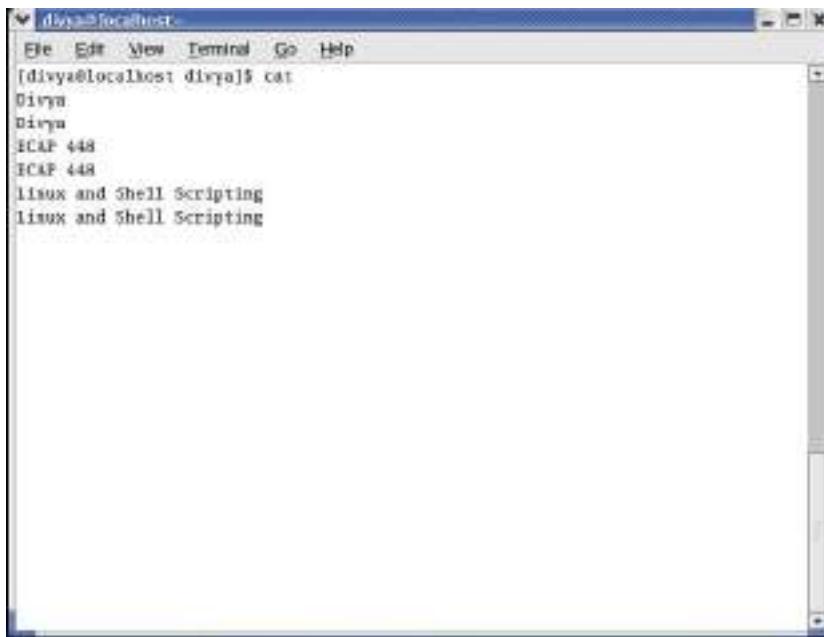
File Edit View Terminal Go Help
[divya@localhost divya]$ ls
Ac_Info Computer Data Linux Syllabus
Address Course_Info Grocery Office_Info
[divya@localhost divya]$ cat Syllabus
Unit 01- Getting started with Linux
Unit 02- Installation Guide
Unit 03- Connecting to Internet
Unit 04- Installing software
Unit 05- Utilities
Unit 06- File Systems
Unit 07- The Shell and popular editors
Unit 08- The Bourne Again Shell and TC Shell
Unit 09- Programming the Bourne Again Shell
Unit 10- Linux System Administration
Unit 11- Web Server Configuration
Unit 12- File Server Configuration
Unit 13- Samba Servers
Unit 14- Network File System

[divya@localhost divya]$

```

Unit 07: The Shell and Popular Editors

When you do not give cat any argument (that is, when you give the command cat followed immediately by RETURN), cat takes its input from standard input. Thus, when called without an argument, cat copies standard input to standard output, one line at a time.



```
[divya@localhost: divya]$ cat
Divya
Divya
ICAP 448
ICAP 448
Linux and Shell Scripting
Linux and Shell Scripting
```

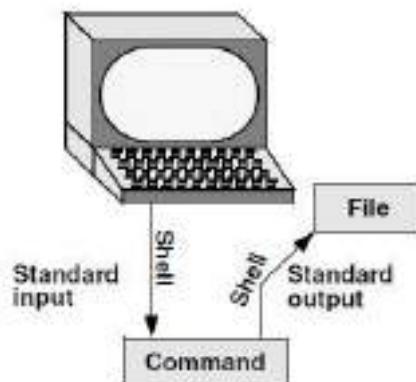
Because the shell associates cat's standard input with the keyboard and cat's standard output with the screen, when you type a line of text cat copies the text from standard input (the keyboard) to standard output (the screen). The cat utility keeps copying text until you enter CONTROL-D on a line by itself. Pressing CONTROL-D sends an EOF (end of file) signal to cat to indicate that it has reached the end of standard input and there is no more text for it to copy. The cat utility then finishes execution and returns control to the shell, which displays a prompt.

7.3 Redirection

The term redirection encompasses the various ways you can cause the shell to alter where standard input of a command comes from and where standard output goes to. By default, the shell associates' standard input and standard output of a command with the keyboard and the screen. You can cause the shell to redirect standard input or standard output of any command by associating the input or output with a command or file other than the device file representing the keyboard and the screen.

Redirecting Standard Output

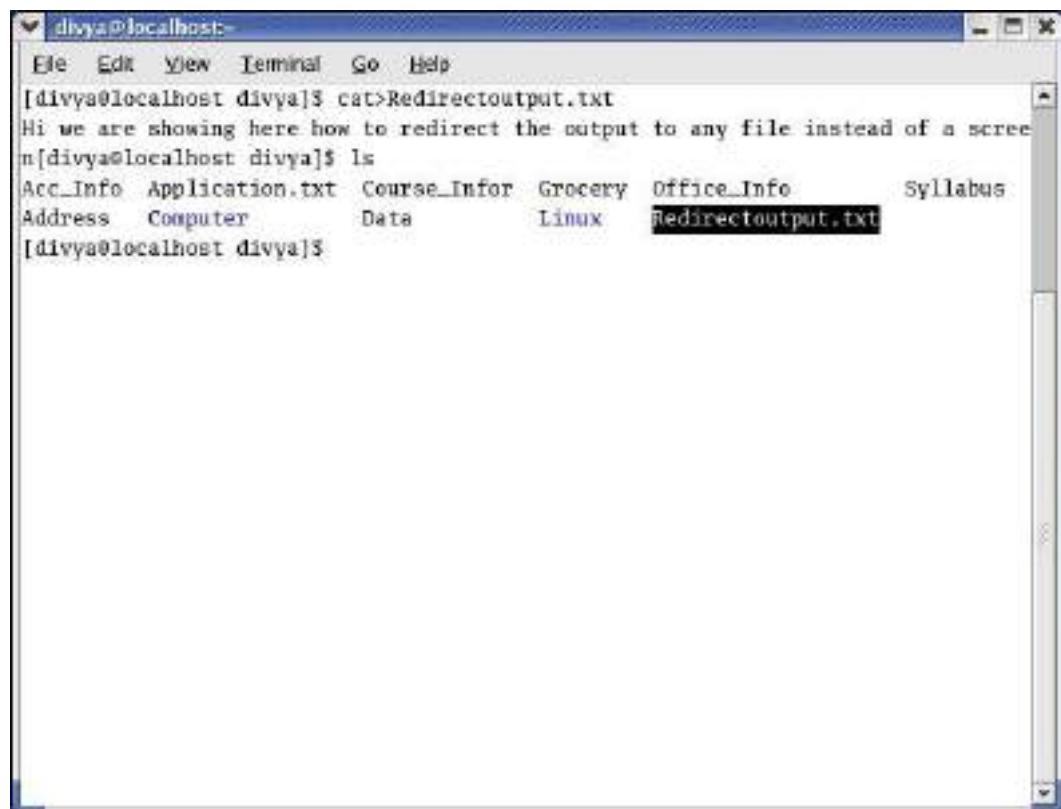
The redirect output symbol (>) instructs the shell to redirect the output of a command to the specified file instead of to the screen.



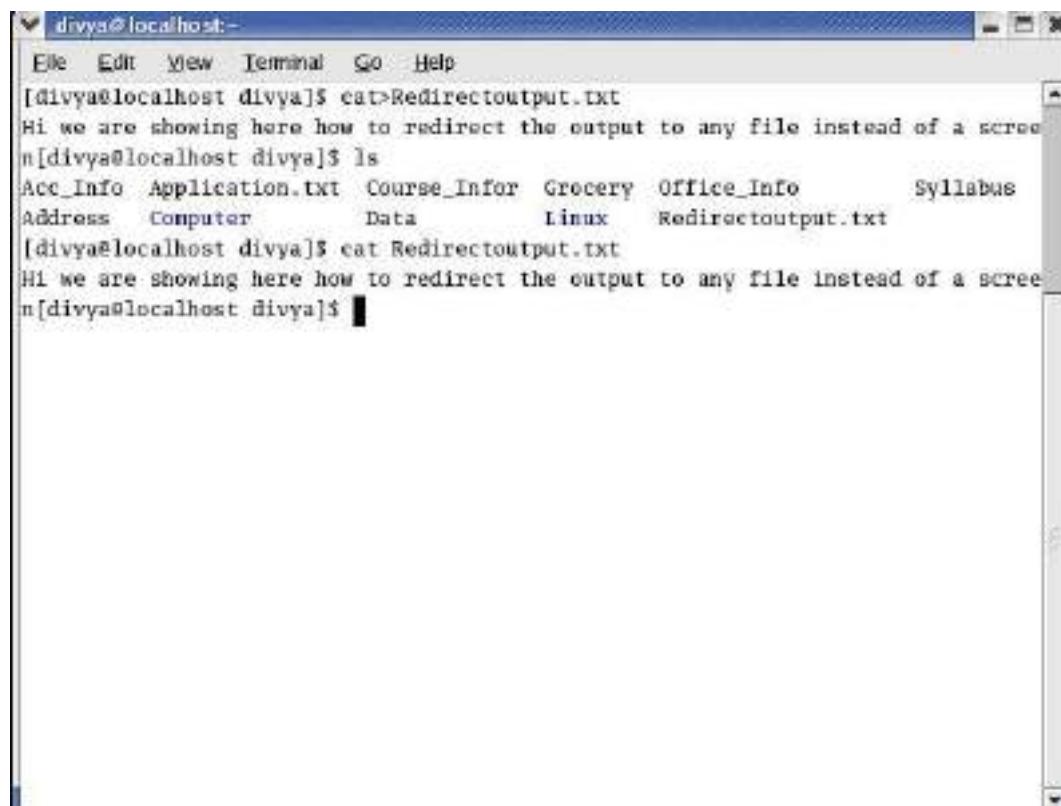
Linux and Shell Scripting

The format of a command line that redirects output is command [arguments] > filename

where command is any executable program (such as an application program or a utility), arguments are optional arguments, and filename is the name of the ordinary file the shell redirects the output to.

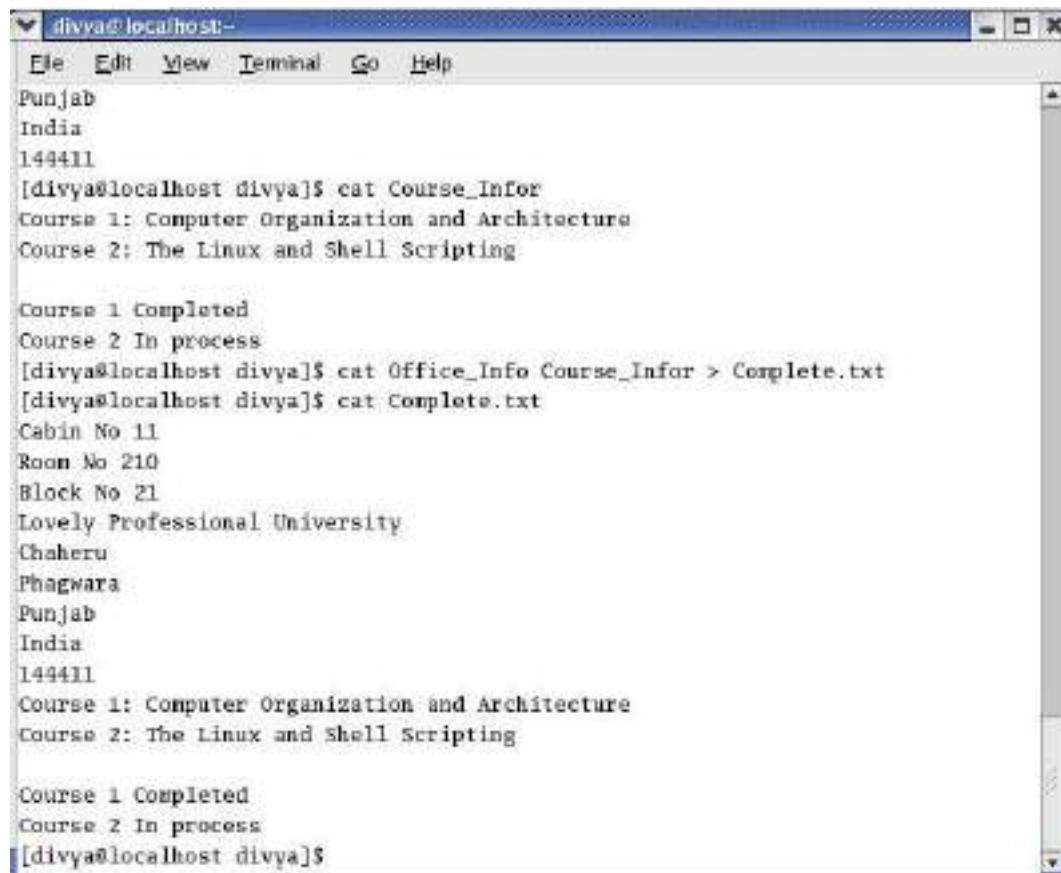


A screenshot of a Linux terminal window titled "divya@localhost:~". The window has a menu bar with "File", "Edit", "View", "Terminal", "Go", and "Help". The terminal prompt is "[divya@localhost divya]\$". The user runs the command "cat>Redirectoutput.txt". The output of the command is displayed in the terminal, showing the text "Hi we are showing here how to redirect the output to any file instead of a screen". After the command, the user runs "ls" to list files in the current directory. The files listed are Acc_Info, Application.txt, Course_Infor, Grocery, Office_Info, Syllabus, Address, Computer, Data, Linux, and Redirectoutput.txt. The "Redirectoutput.txt" file is highlighted with a black border.



A screenshot of a Linux terminal window titled "divya@localhost:~". The window has a menu bar with "File", "Edit", "View", "Terminal", "Go", and "Help". The terminal prompt is "[divya@localhost divya]\$". The user runs the command "cat>Redirectoutput.txt". The output of the command is displayed in the terminal, showing the text "Hi we are showing here how to redirect the output to any file instead of a screen". After the command, the user runs "cat Redirectoutput.txt". The output of this command is displayed in the terminal, showing the same text "Hi we are showing here how to redirect the output to any file instead of a screen". The terminal prompt "[divya@localhost divya]\$" is visible at the bottom.

The redirect output symbol on the command line causes the shell to associate cat's standard output with the redirectoutput.txt file specified on the command line. Redirecting standard output from cat is a handy way to create a file without using an editor. The drawback is that once you enter a line and press RETURN, you cannot edit the text. While you are entering a line, the erase and kill keys work to delete text. This procedure is useful for creating short, simple files. The cat is used and the redirect output symbol to catenate (join one after the other – the derivation of the name of the cat utility) several files into one larger file.



```

divya@localhost:~-
File Edit View Terminal Go Help
Punjab
India
144411
[divya@localhost divya]$ cat Course_Info
Course 1: Computer Organization and Architecture
Course 2: The Linux and Shell Scripting

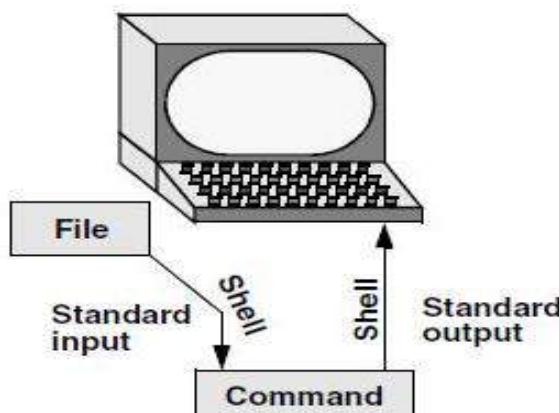
Course 1 Completed
Course 2 In process
[divya@localhost divya]$ cat Office_Info Course_Info > Complete.txt
[divya@localhost divya]$ cat Complete.txt
Cabin No 11
Room No 210
Block No 21
Lovely Professional University
Chaheru
Phagwara
Punjab
India
144411
Course 1: Computer Organization and Architecture
Course 2: The Linux and Shell Scripting

Course 1 Completed
Course 2 In process
[divya@localhost divya]$

```

Redirecting Standard Input

The redirect input symbol (<) instructs the shell to redirect a command's input to come from the specified file instead of from the keyboard.



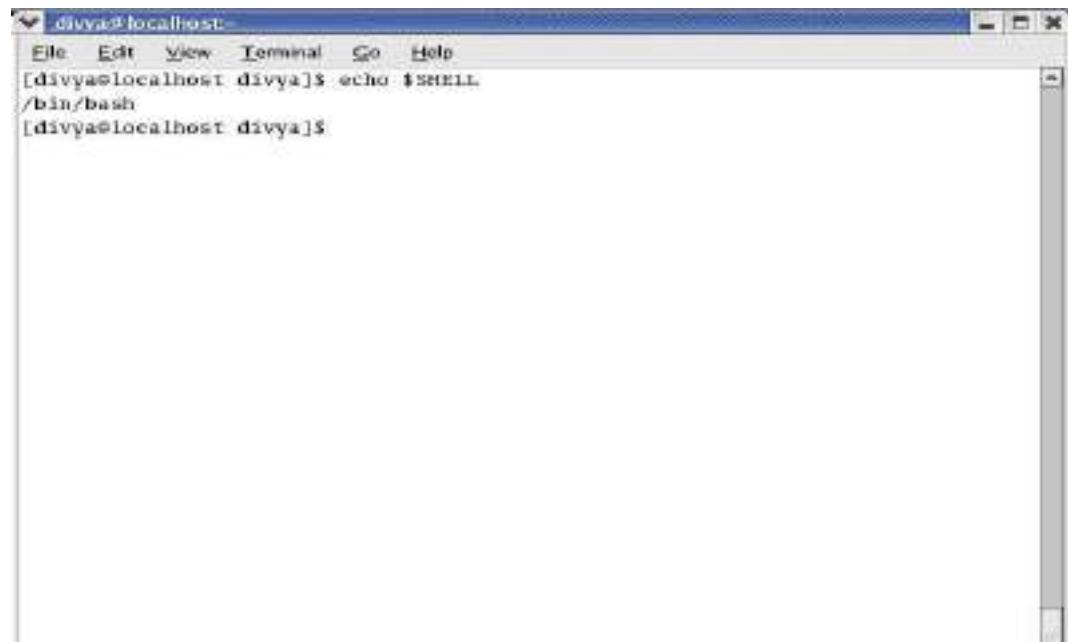
The format of a command line that redirects input is: command [arguments] < filename

Linux and Shell Scripting

where command is any executable program (such as an application program or a utility), arguments are optional arguments, and filename is the name of the ordinary file the shell redirects the input from.

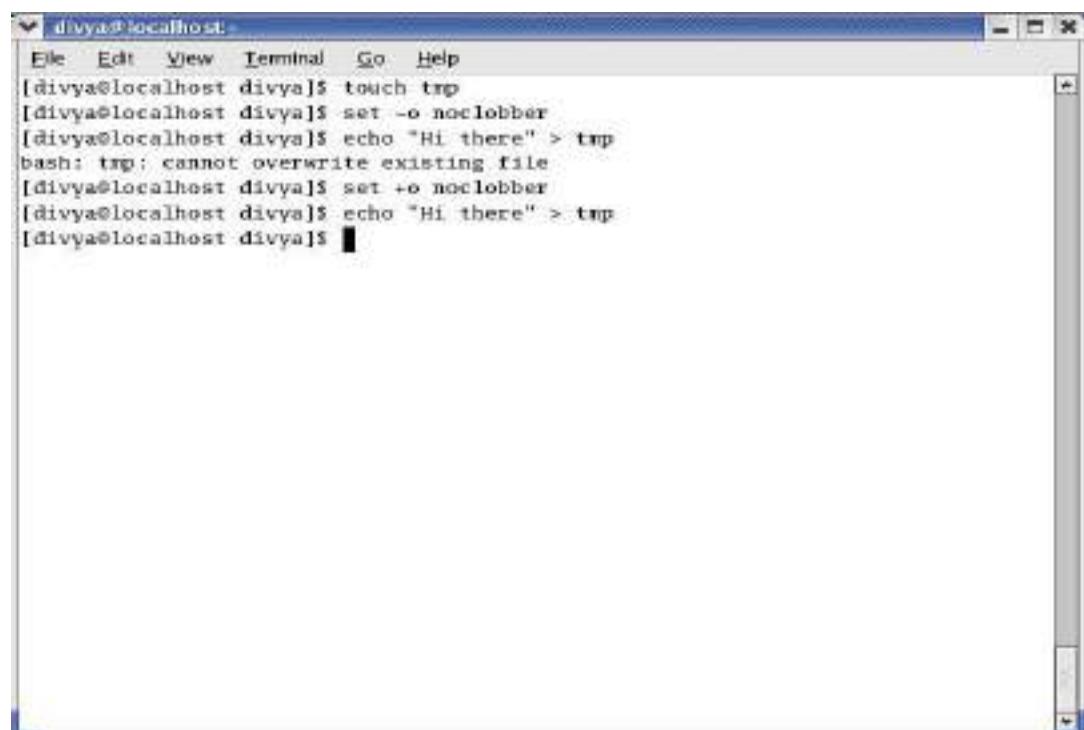
noclobber: Avoids Overwriting Files

The shell provides the noclobber feature that prevents overwriting a file using redirection. Under bash you can enable this feature by setting noclobber using the command set -onoclobber. The same command with +o unsets noclobber. Under tcsh use set noclobber and unset noclobber. With noclobber set, if you redirect output to an existing file, the shell displays an error message and does not execute the command.



A screenshot of a Linux terminal window titled "divya@localhost". The window has a standard Windows-style title bar with icons for minimize, maximize, and close. The menu bar includes "File", "Edit", "View", "Terminal", "Go", and "Help". The terminal area shows the following command and its output:

```
[divya@localhost divya]$ echo $SHELL  
/bin/bash  
[divya@localhost divya]$
```

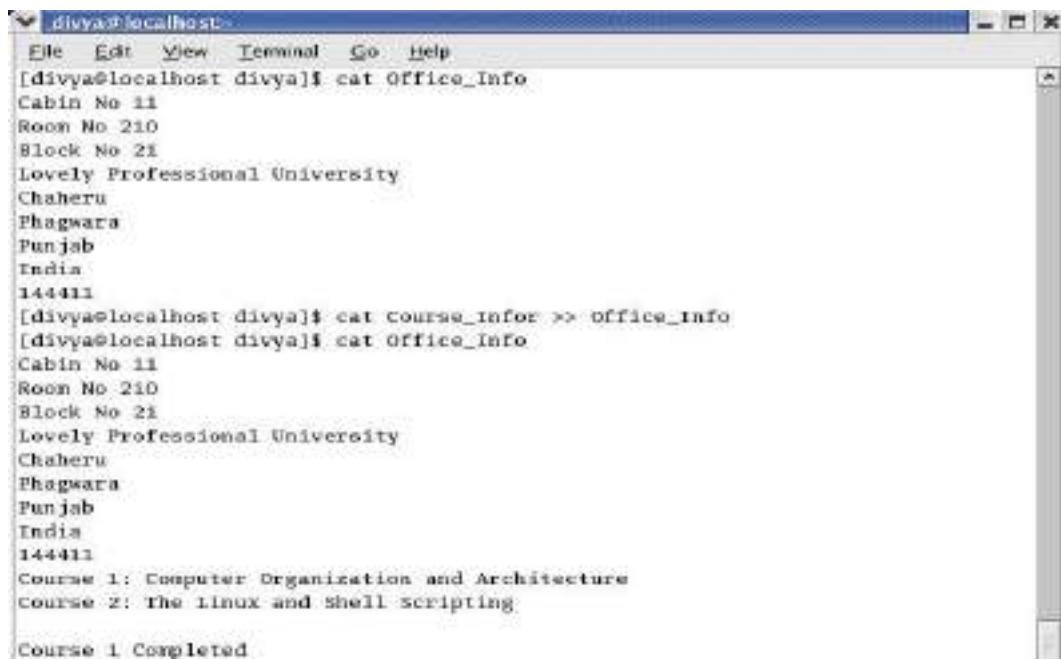


A screenshot of a Linux terminal window titled "divya@localhost". The window has a standard Windows-style title bar with icons for minimize, maximize, and close. The menu bar includes "File", "Edit", "View", "Terminal", "Go", and "Help". The terminal area shows the following sequence of commands and their results:

```
[divya@localhost divya]$ touch tmp  
[divya@localhost divya]$ set -o noclobber  
[divya@localhost divya]$ echo "Hi there" > tmp  
bash: tmp: cannot overwrite existing file  
[divya@localhost divya]$ set +o noclobber  
[divya@localhost divya]$ echo "Hi there" > tmp  
[divya@localhost divya]$
```

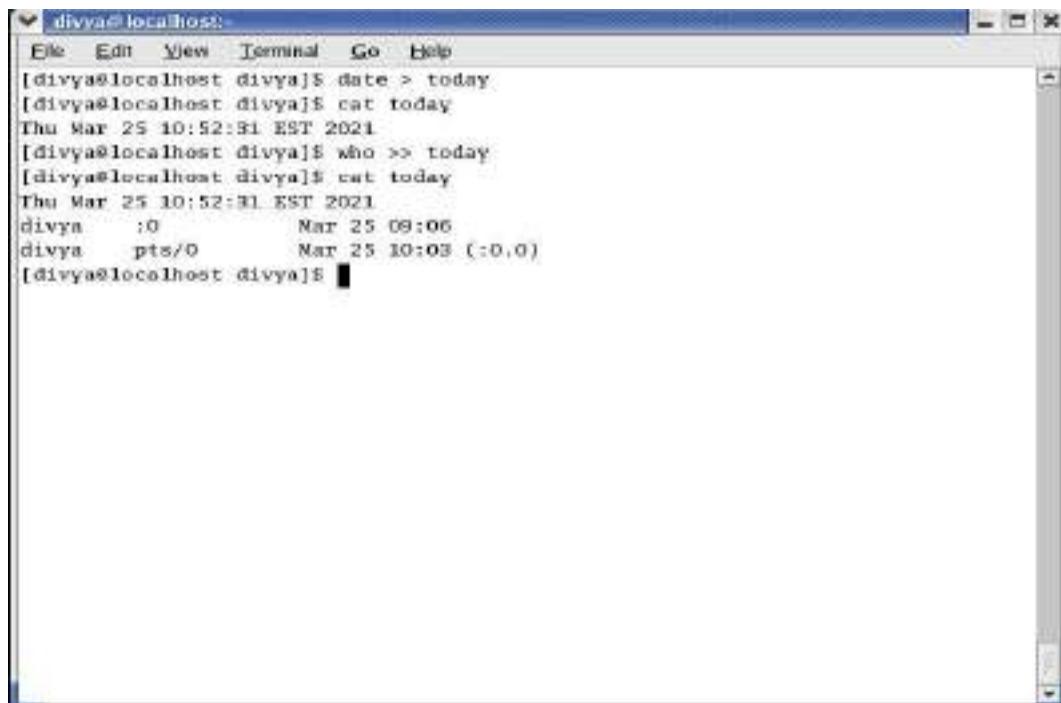
Appending Standard Output to a File

The append output symbol (>>) causes the shell to add new information to the end of a file, leaving existing information intact. This symbol provides a convenient way of catenating two files into one.



```
[divya@localhost divya]$ cat Office_Info
Cabin No 11
Room No 210
Block No 21
Lovely Professional University
Chaheru
Phagwara
Punjab
India
144411
[divya@localhost divya]$ cat course_infor >> office_info
[divya@localhost divya]$ cat Office_Info
Cabin No 11
Room No 210
Block No 21
Lovely Professional University
Chaheru
Phagwara
Punjab
India
144411
Course 1: Computer Organization and Architecture
Course 2: The linux and shell scripting

Course 1 Completed
```



```
[divya@localhost divya]$ date > today
[divya@localhost divya]$ cat today
Thu Mar 25 10:52:31 EST 2021
[divya@localhost divya]$ who >> today
[divya@localhost divya]$ cat today
Thu Mar 25 10:52:31 EST 2021
divya :0 Mar 25 09:06
divya pts/0 Mar 25 10:03 (:0.0)
[divya@localhost divya]$
```

/dev/null: Making Data Disappear

The /dev/null device is a data sink, commonly referred to as a bit bucket. You can redirect output that you do not want to keep or see to /dev/null and the output will disappear without a trace.

Linux and Shell Scripting

```
[divya@localhost divya]$ echo "Hi there" > /dev/null
[divya@localhost divya]$
```

7.4 Pipes

The shell uses a pipe to connect standard output of one command to standard input of another command. A pipe (sometimes referred to as a pipeline) has the same effect as redirecting standard output of one command to a file and then using that file as standard input to another command. The symbol for a pipe is a vertical bar (|). The syntax of a command line using a pipe is

```
command_a [arguments] | command_b [arguments]
```

The preceding command line uses a pipe on a single command line to generate the same result as the following three command lines:

```
command_a [arguments] > temp
```

```
command_b [arguments] < temp
```

```
rm temp
```

7.5 Filters

A filter is a command that processes an input stream of data to produce an output stream of data. A command line that includes a filter uses a pipe to connect standard output of one command to the filter's standard input. Another pipe connects the filter's standard output to standard input of another command. Not all utilities can be used as filters. In this example, sort is a filter, taking standard input from standard output of who and using a pipe to redirect standard output to standard input of lpr. This command line sends the sorted output of who to the printer:\$ who | sort | lpr

tee: Sends Output in Two Directions

The tee utility copies its standard input both to a file and to standard output. This utility is aptly named: It takes a single stream of input and sends the output in two directions.

```
$ who | tee who.out | grep sam
sam      console   Mar 24 05:00
$ cat who.out
sam      console   Mar 24 05:00
max     pts/4      Mar 24 12:23
max     pts/5      Mar 24 12:33
zach    pts/7      Mar 23 08:45
```

The output of who is sent via a pipe to standard input of tee. The tee utility saves a copy of standard input in a file named who.out and sends a copy to standard output. Standard output of tee goes via a pipe to standard input of grep, which displays only those lines containing the string sam.

7.6 Running a Command in the Background

All commands up to this point have been run in the foreground. When you run a command in the foreground, the shell waits for it to finish before displaying another prompt and allowing you to continue. When you run a command in the background, you do not have to wait for the command to finish before running another command. A job is a series of one or more commands that can be connected by pipes. You can have only one foreground job in a window or on a screen, but you can have many background jobs. Multitasking is the running of more than one job at a time.

To run a job in the background, type an ampersand (&) just before the RETURN that ends the command line. The shell assigns a small number to the job and displays this job number between brackets. Following the job number, the shell displays the process identification (PID) number—a larger number assigned by the operating system. Each of these numbers identifies the job running in the background. The shell then displays another prompt, and you can enter another command. When the background job finishes, the shell displays a message giving both the job number and the command line used to run the command. This command runs in the background; it sends the output of ls through a pipe to lpr, which sends it to the printer.

```
$ ls -l | lpr&
[1] 22092
$
```

The [1] following the command line indicates that the shell has assigned job number 1 to this job. The 22092 is the PID number of the first command in the job. When this background job completes execution, the shell displays the message [1]+ Done ls -l | lp

Moving a Job from the Foreground to the Background

You can suspend a foreground job (stop it from running without aborting the job) by pressing the suspend key, usually CONTROL-Z. The shell then stops the process and disconnects standard input from the keyboard. You can put a suspended job in the background and restart it by using the bg command followed by the job number. You do not need to specify the job number when there is only one stopped job. Only the foreground job can take input from the keyboard.

To connect the keyboard to a program running in the background, you must bring it to the foreground. To do so, type fg without any arguments when only one job is in the background. When more than one job is in the background, type fg, or a percent sign (%), followed by the number of the job you want to bring into the foreground. The shell displays the command you used to start the job (promptme in the following example), and you can enter any input the program requires to continue:

```
bash $ fg 1
promptme
```

kill: Aborting a Background Job

The interrupt key (usually CONTROL-C) cannot abort a background process; you must use kill for this purpose. Follow kill on the command line with either the PID number of the process you want to abort or a percent sign (%) followed by the job number. If you forget a PID number, you can use the ps (process status) utility to display it. The following example runs a tail -f outfile command as a background job, uses ps to display the PID number of the process, and aborts the job with kill:

```
$ tail -f outfile&
[1] 18228
$ ps | grep tail
18228 pts/4 00:00:00 tail
$ kill 18228
[1]+ Terminated tail -f outfile
$
```

If you forget a job number, you can use the jobs command to display a list of job numbers. The next example is like the previous one except it uses the job number instead of the PID number to identify the job to be killed. Sometimes the message saying the job is terminated does not appear until you press RETURN after the RETURN that executes the kill command.

```
$ tail -f outfile&
[1] 18236
$ bigjob&
[2] 18237
$ jobs
[1]- Running tail -f outfile&
[2]+ Running bigjob&
$ kill %1
$ RETURN
[1]- Terminated tail -f outfile
$
```

7.7 Filename Generation/Pathname Expansion

When you give the shell abbreviated filenames that contain special characters, also called metacharacters, the shell can generate filenames that match the names of existing files. These special characters are also referred to as wildcards because they act much as the jokers do in a deck of cards. When one of these characters appears in an argument on the command line, the shell expands that argument in sorted order into a list of filenames and passes the list to the program called by the command line. Filenames that contain these special characters are called ambiguous file references because they do not refer to any one specific file. The process that the shell performs on these filenames is called pathname expansion or globbing.

Ambiguous file references refer to a group of files with similar names quickly, saving the effort of typing the names individually. They can also help find a file whose name you do not remember in its entirety. If no filename matches the ambiguous file reference, the shell generally passes the unexpanded reference—special characters and all—to the program.

The ? Special Character

The question mark (?) is a special character that causes the shell to generate filenames. It matches any single character in the name of an existing file. The following command uses this special character in an argument to the lpr utility: \$ lprmemo?.

Unit 07: The Shell and Popular Editors

The shell expands the memo? argument and generates a list of files in the working directory that have names composed of memo followed by any single character. The shell then passes this list to lpr. The lpr utility never “knows” the shell generated the filenames it was called with. If no filename matches the ambiguous file reference, the shell passes the string itself (memo?) to lpr or, if it is set up to do so, passes a null string. The following example uses ls first to display the names of all files in the working directory and then to display the filenames that memo? matches:

```
$ ls
mem memo12 memo9 memomax newmemo5
memo memo5 memoa memos

$ ls memo?
memo5 memo9 memoa memos
```

The memo? ambiguous file reference does not match mem, memo, memo12, memomax, or newmemo5. You can also use a question mark in the middle of an ambiguous file reference:

```
$ ls
7may4report may4report mayqreportmay_report
may14report may4report.79 mayreportmay.report

$ ls may?report
may.report may4report may_reportmayqreport
```

The * Special Character

The asterisk (*) performs a function like that of the question mark but matches any number of characters, including zero characters, in a filename.

```
$ ls
amemo memo.0612 memosally memsam user.memo mem memoa memosam.0620
sallymemo memo memorandum memosam.keep typescript

$ echo memo*
memo memo.0612 memoa memorandum memosally memosam.0620 memosam.keep

$ echo *mo
amemo memo sallymemouser.memo

$ echo *sam*
memosam.0620 memosam.keepmemsam
```

The ambiguous file reference memo* does not match amemo, mem, sallymemo, or user.memo. Like the question mark, an asterisk does not match a leading period in a filename. The -a option causes ls to display hidden filenames. The command echo * does not display . (the working directory), .. (the parent of the working directory), .aaa, or .profile. In contrast, the command echo .* displays only those four names:

```
$ ls
aaamemo.sally sally.0612 thurs
memo.0612 report saturday
```

```
$ ls -a
.
.. .aaaaamemo.sally  sally.0612  thurs
.. .profile  memo.0612  report  saturday

$ echo *
aaa  memo.0612  memo.sally  report sally.0612 saturdaythurs

$ echo *
.
.. .aaa  .profile
```

In the following example, `.p*` does not match `memo.0612`, `private`, `reminder`, or `report`. The `ls .*` command causes `ls` to list `.private` and `.profile` in addition to the contents of the `.` directory (the working directory) and the `..` directory (the parent of the working directory). When called with the same argument, `echo` displays the names of files (including directories) in the working directory that begin with a dot (`.`), but not the contents of directories.

```
$ ls -a
.
.. .private memo.0612 reminder
.. .profile private report

$ echo .p*
.private .profile

$ ls *
.private .profile
.:
memo.0612 private reminder report
.:
.:

$ echo .*
. .. .private .profile
```

You can plan to take advantage of ambiguous file references when you establish conventions for naming files. For example, when you end all text filenames with .txt, you can reference that group of files with *.txt. The next command uses this convention to send all text files in the working directory to the printer. The ampersand (&) causes lpr to run in the background.

```
$ lpr *.txt &
```

The [] Special Characters

A pair of brackets surrounding a list of characters causes the shell to match filenames containing the individual characters. Whereas `memo?` matches `memo` followed by any character, `memo[1?]` is more restrictive: It matches only `memo1`, `memo7`, and `memoa`. The brackets define a character class that includes all the characters within the brackets. The shell expands an argument that includes a

Unit 07: The Shell and Popular Editors

character-class definition, by substituting each member of the character class, one at a time, in place of the brackets and their contents. The shell then passes the list of matching filenames to the program it is calling. Each character-class definition can replace only a single character within a filename. The brackets and their contents are like a question mark that substitutes only the members of the character class. The first of the following commands lists the names of all files in the working directory that begin with a, e, i, o, or u. The second command displays the contents of the files named page2.txt, page4.txt, page6.txt, and page8.txt.

```
$ echo [aeiou]*  
...  
$ less page[2468].txt  
...
```

A hyphen within brackets defines a range of characters within a character-class definition. For example, [6-9] represents [6789], [a-z] represents all lowercase letters in English, and [a-zA-Z] represents all letters, both uppercase and lowercase, in English. The following command lines show three ways to print the files named part0, part1, part2, part3, and part5. Each of these command lines causes the shell to call lpr with five filenames:

```
$ lpr part0 part1 part2 part3 part5  
$ lpr part[01235]  
$ lpr part[0-35]
```

The following command line prints 39 files, part0 through part38:

```
$ lprpart[0-9] part[12][0-9] part3[0-8]
```

The first of the following commands lists the files in the working directory whose names start with a through m.

```
$ echo [a-m]*  
...
```

The second lists files whose names end with x, y, or z.

```
$ echo *[x-z]  
...
```

The ls utility cannot interpret ambiguous file references. First ls is called with an argument of ?old. The shell expands ?old into a matching filename, hold, and passes that name to ls.

```
$ ls ?old  
hold
```

The second command is the same as the first, except the ? is quoted.

```
$ ls \?old  
ls: ?old: No such file or directory
```

The shell does not recognize this question mark as a special character and passes it to ls. The ls utility generates an error message saying that it cannot find a file named ?old (because there is no file named ?old).

7.8 Builtins

A builtin is a utility (also called a command) that is built into a shell. Each of the shells has its own set of builtins. When it runs a builtin, the shell does not fork a new process. Consequently, builtins run more quickly and can affect the environment of the current shell. Because builtins are used in the same way as utilities, you will not typically be aware of whether a utility is built into the shell or is a standalone utility. The echo utility, for example, is a shell builtin. The shell always executes a shell builtin before trying to find a command or utility with the same name.

Listing bash builtins

To display a list of bash builtins, give the command info bash and select the Shell Builtins menu. Then select the Bourne Shell Builtins and/or Bash Builtins menus. The bash info page is part of the bash-doc package—you can view only the man page (even using info) if this package is not installed. Because bash was written by GNU, the info page has better information than does the man page.

Listing tcshbuiltins

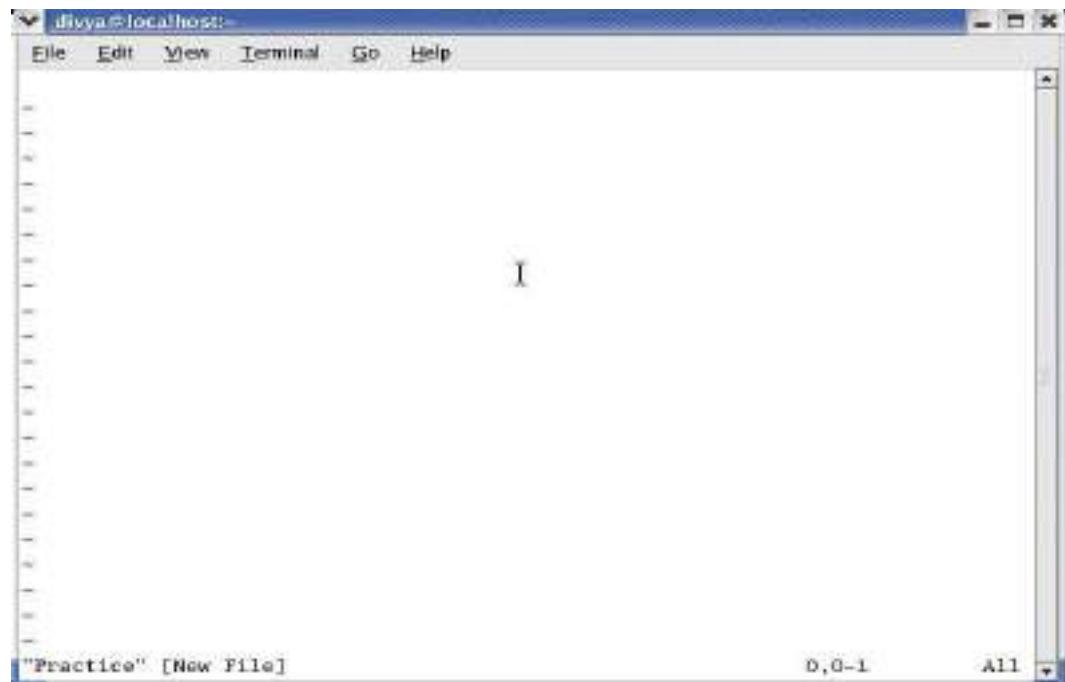
For tcsh, give the command man tcsh to display the tcsh man page and then search with the following command: /Builtin commands\$ (search for the string at the end of a line).

7.9 vim

The vim editor is not a text formatting program. It does not justify margins or provide the output formatting features of a sophisticated word processing system such as OpenOffice.org Writer. Vim is a sophisticated text editor meant to be used to write code (C, HTML, Java, etc.), short notes, and input to a text formatting system, such as groff or troff.

Starting vim

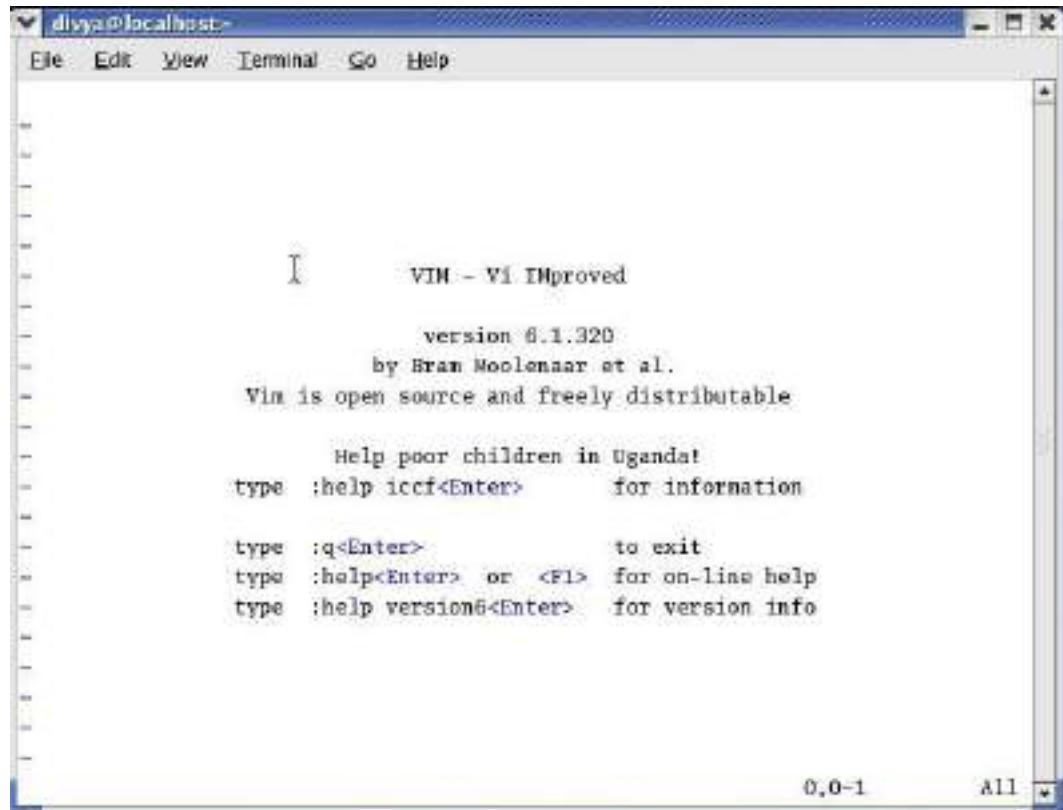
Start vim with the following command to create and edit a file named practice: \$ vim practice. The tildes in the starting indicates that the page is blank.



Unit 07: The Shell and Popular Editors

If you call vim without specifying a filename on the command line, vim assumes that you are a novice and tells you how to get started.

\$ vim



The screenshot shows a terminal window titled "dixia@localhost:~". The window contains the Vim startup message:

```

VIM - Vi IMproved
version 8.1.320
by Bram Moolenaar et al.
Vim is open source and freely distributable

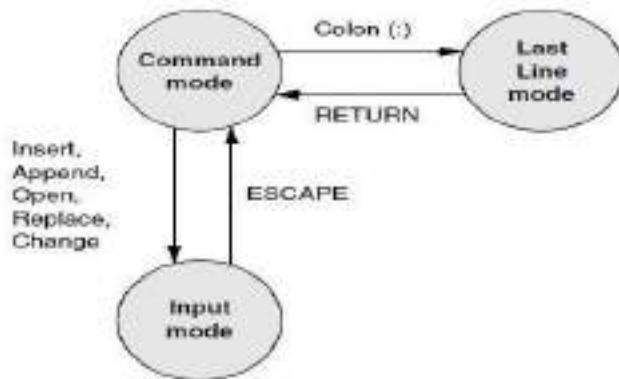
      Help poor children in Uganda!
type :help iccf<Enter>      for information

type :q<Enter>              to exit
type :help<Enter> or <F1> for on-line help
type :help version6<Enter>   for version info

```

The status bar at the bottom right shows "0,0-1" and "All".

7.10 Modes in vim



Two of vim's modes of operation are Command mode (also called Normal mode) and Input mode. While vim is in Command mode, you can give vim commands. For example, you can delete text or exit from vim. You can also command vim to enter Input mode. In Input mode, vim accepts anything you enter as text and displays it on the screen. The colon (:) in the preceding command puts vim into another mode, Last Line mode. While in this mode, vim keeps the cursor on the bottom line of the screen. When you press RETURN to finish entering the command, vim restores the cursor to its place in the text. Press ESCAPE to return vim to Command mode. By default, the vim editor informs you about which mode it is in: It displays INSERT at the lower-left corner of the screen while it is in Insert mode.

Entering Text

When you start vim, you must put it in Input mode before you can enter text. To put vim in Input mode, press the i (insert before cursor) key or the a (append after cursor) key. If you are not sure whether vim is in Input mode, press the ESCAPE key; vim returns to Command mode if it is in Input mode or beeps, flashes, or does nothing if it is already in Command mode. You can put vim back in Input mode by pressing the i or a key again. While vim is in Input mode, you can enter text by typing on the keyboard. If the text does not appear on the screen as you type, vim is not in Input mode.

Correcting Text

The keys that back up and correct a shell command line serve the same functions when vim is in Input mode.

Key	Results
CONTROL-H	Erase
CONTROL-U	Line kill
CONTROL-W	Word kill
Dd, dw, x	Remove the incorrect text
I, a, o, O	Insert the correct text

To change the word pressing to hitting in, you might use the ARROW keys to move the cursor until it is on top of the p in pressing. Then give the command dw to delete the word pressing. Put vim in Input mode by giving an i command, enter the word hitting followed by a SPACE, and press ESCAPE. The word is changed, and vim is in Command mode, waiting for another command. A shorthand for the two commands dw followed by the i command is cw (Change word). The cw command puts vim into Input mode.

Ending the Editing Session

While you are editing, vim keeps the edited text in an area named the Work buffer. Use the ZZ command (you must use uppercase Zs) to write the newly entered text to the disk. You can exit with :q! if you do not want to save your work.

Moving the Cursor

To delete, insert, and correct text, you need to move the cursor on the screen. While vim is in Command mode, you can use the RETURN key, the SPACE bar, and the ARROW keys to move the cursor. You can use the h, j, k, and l (lowercase "l") keys to move the cursor left, down, up, and right, respectively.

Deleting Text

If you want to delete a single character, then press x. If a word needs to be deleted, press dw and for the deletion of line of text, press dd.

Undoing Mistakes

Give the command u (Undo) immediately after the command you want to undo. If you give the u command again, vim will undo the command you gave before the one it just undid. You can use this technique to back up over many of your actions. If you undo a command, you did not mean to undo, give a Redo command: CONTROL-R or :redo (followed by a RETURN). As with the Undo command, you can give the Redo command many times in a row.

Entering Additional Text

When you want to insert new text within existing text, move the cursor so it is on the character that follows the new text you plan to enter. Then give the i (Insert) command to put vim in Input mode, enter the new text, and press ESCAPE to return vim to Command mode. Alternatively, you can position the cursor on the character that precedes the new text and use the a (Append) command. To enter one or more lines, position the cursor on the line above where you want the new text to go. Give the command o (Open). The vim editor opens a blank line below the line the cursor was on, puts the cursor on the new, empty line, and enters Input mode. Enter the new text, ending each line with a RETURN. When you are finished entering text, press ESCAPE to return vim to Command mode. The O command works in the same way as the o command, except it opens a blank line above the current line.

7.11 Introduction to vim Features

There are various useful features of vim which are of great help:

- Online help,
- Modes of operation,
- The work buffer,
- Emergency procedures,
- Other vim features.

Online Help

Give the command: help feature to display information about feature. As you scroll through the various help texts, you will see words with a bar on either side, such as |tutor|. These words are active links: Move the cursor on top of an active link and press CONTROL-] to jump to the linked text. Use CONTROL-o (lowercase "o") to jump back to where you were in the help text. You can also use the active link words in place of feature. For example, you might see the reference |credits|; you could enter :help credits RETURN to read more about credits. Enter :q! to close a help window. You can also give the command :help doc-file-list to view a complete list of the help files.

Modes of Operation

The current character is the character the cursor is on. The current line is the line the cursor is on. The status line is the last or bottom line of the screen. This line is reserved for Last Line mode and status information. Text you are editing does not appear on this line. The vim editor is part of the ex-editor, which has five modes of operation:

- ex Command mode
- ex Input mode
- vim Command mode
- vim Input mode
- vim Last Line mode

While in Command mode, vim accepts keystrokes as commands, responding to each command as you enter it. It does not display the characters you type in this mode. While in Input mode, vim accepts and displays keystrokes as text that it eventually puts into the file you are editing. All commands that start with a colon (:) put vim in Last Line mode. The colon moves the cursor to the status line of the screen, where you enter the rest of the command. When you give a command in Command mode, you do not terminate the command with a RETURN. In contrast, you must terminate all Last Line mode commands with a RETURN. When an editing session begins, vim is in Command mode. Several commands, including Insert and Append, put vim in Input mode. When you press the ESCAPE key, vim always reverts to Command mode. The Change and Replace commands combine the Command and Input modes. The Change command deletes the text you

Linux and Shell Scripting

want to change and puts vim in Input mode so you can insert new text. The Replace command deletes the character(s) you overwrite and inserts the new one(s) you enter.

The vim editor displays status information on the bottom line of the display area. This information includes error messages, information about the deletion or addition of blocks of text, and file status information. In addition, vim displays Last Line mode commands on the status line. Sometimes the screen may become garbled or overwritten. When vim puts characters on the screen, it sometimes leaves @ on a line instead of deleting the line. When output from a program becomes intermixed with the display of the Work buffer, things can get even more confusing. The output does not become part of the Work buffer but affects only the display. If the screen gets overwritten, press ESCAPE to make sure vim is in Command mode, and press CONTROL-L to redraw (refresh) the screen. If the end of the file is displayed on the screen, vim marks lines that would appear past the end of the file with a tilde (~) at the left of the screen. While vim is in Input mode, you can use the erase and line kill keys to back up over text so you can correct it. You can also use CONTROL-W to back up over words.

Work Buffer

The vim editor does all its work in the Work buffer. At the beginning of an editing session, vim reads the file you are editing from the disk into the Work buffer. During the editing session, it makes all changes to this copy of the file but does not change the file on the disk until you write the contents of the Work buffer back to the disk. Normally when you end an editing session, you tell vim to write the contents of the Work buffer, which makes the changes to the text final. If you accidentally end an editing session without writing out the contents of the Work buffer, your work is lost. However, if you unintentionally make some major changes (such as deleting the entire contents of the Work buffer), you can end the editing session without implementing the changes. To look at a file but not to change it while you are working with vim, you can use the view utility: \$ view filename. Calling the view utility is the same as calling the vim editor with the -R (readonly) option.

The vim editor operates on files of any format. The total length of the file is limited only by available disk space and memory. The vim editor allows you to open, close, and hide multiple windows, each of which allows you to edit a different file. Give the command :help windows to display a complete list of windows commands.

Abnormal Termination of an Editing Session and recovering after a crash (Emergency Procedures)

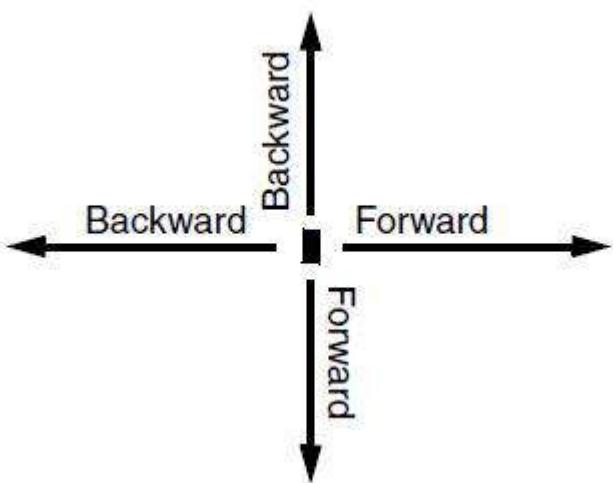
You can end an editing session in one of two ways: When you exit from vim, you can save the changes you made during the editing session or you can abandon those changes. You can use the ZZ or :wq command from Command mode to save the changes and exit from vim. To end an editing session without writing out the contents of the Work buffer, give the following command:

:q!. Use the :q! command cautiously. When you use this command to end an editing session, vim does not preserve the contents of the Work buffer, so you will lose any work you did since the last time you wrote the Work buffer to disk.:w filename: Use this to save the file with a name.

The vim editor temporarily stores the file you are working on in a swap file. If the system crashes while you are editing a file with vim, you can often recover its text from the swap file. If someone else is editing the file, quit or open the file as a read only file. With the -r option, vim displays a list of swap files it has saved (some may be old). If your work was saved, give the same command followed by a SPACE and the name of the file. Give the command :w filename immediately to save the salvaged copy of the Work buffer to disk under a name different from the original file; then check the recovered file to make sure it is OK.

7.12 Command Mode

While vim is in Command mode, you can position the cursor over any character on the screen.

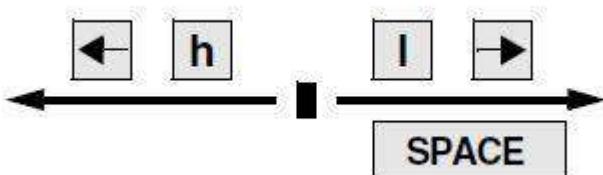


Forward means toward the right and bottom of the screen and the end of the file. Backward means toward the left and top of the screen and the beginning of the file.

Moving the Cursor

When you use a command that moves the cursor forward past the end (right) of a line, the cursor generally moves to the beginning (left) of the next line. When you move it backward past the beginning of a line, the cursor generally moves to the end of the previous line. Sometimes a line in the Work buffer may be too long to appear as a single line on the screen. In such a case vim wraps the current line onto the next line. You can move the cursor through the text by any Unit of Measure (that is, character, word, line, sentence, paragraph, or screen). If you precede a cursor-movement command with a number, called a Repeat Factor, the cursor moves that number of units through the text.

Moving the Cursor by Characters



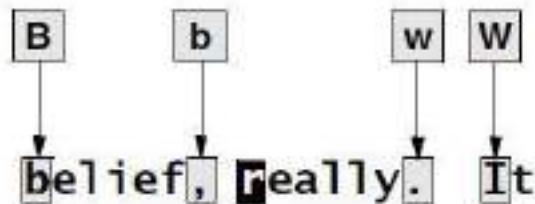
Key	Results
Space bar, l and Right Arrow Key	Forward
h and Left Arrow Key	Backward

For example, the command 7 SPACE or 7l moves the cursor seven characters to the right.

Moving the Cursor to a Specific Character

You can move the cursor to the next occurrence of a specified character on the current line by using the Find command. For example, the following command moves the cursor from its current position to the next occurrence of the character a, if one appears on the same line: fa. You can also find the previous occurrence by using a capital F. The following command moves the cursor to the position of the closest previous a in the current line: Fa. A semicolon (;) repeats the last Find command.

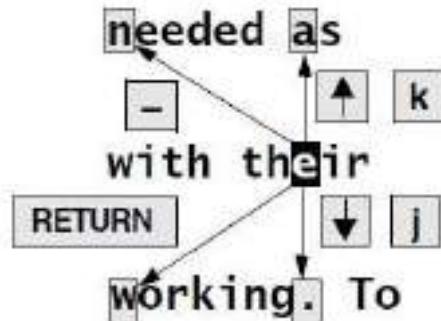
Moving the Cursor by Words



Key	Results
W and w	Moves the cursor forward
B and b	Moves the cursor backwards
E and e	Moves the cursor to the end of the next word

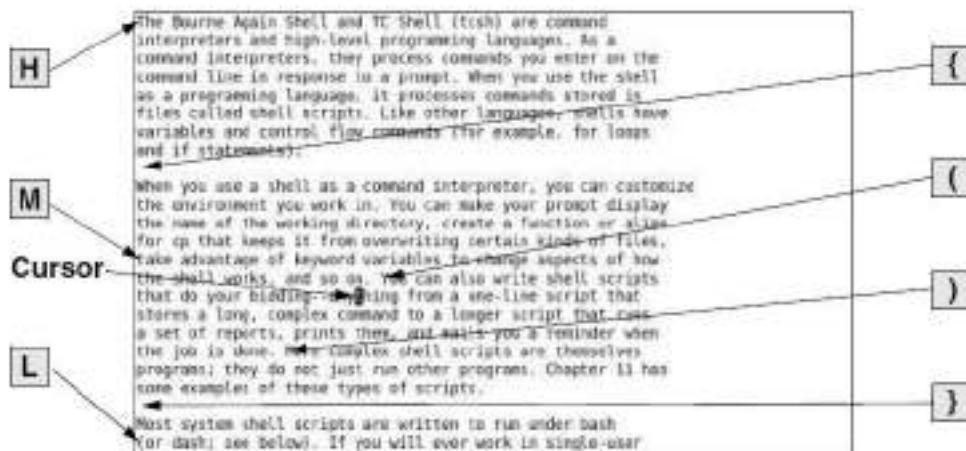
Groups of punctuation count as words. The command 15w moves the cursor to the first character of the fifteenth subsequent word. The W key is like the w key but moves the cursor by blank-delimited words, including punctuation, as it skips forward. The B key moves the cursor backward by blank-delimited words. E moves it to the end of the next blank-delimited word.

Moving the Cursor by Lines



Key	Results
RETURN	To the beginning of next line.
j/ DOWN Arrow	Down one line to the character just below the current character.
k and UP Arrow	Up one line to the character just above the current character.
-	To the end of the next line.
)	Forward to the beginning of the next sentence.
(Forward to the beginning of the next paragraph.
}	Backward to the beginning of current sentence.
{	Backward to the beginning of current paragraph.

Moving the Cursor Within the Screen



Key	Result
H	To the left end of the top line of the screen
M	To the middle line
L	To the bottom line

Viewing Different Parts of the Work Buffer

The screen displays a portion of the text that is in the Work buffer. You can display the text preceding or following the text on the screen by scrolling the display. You can also display a portion of the Work buffer based on a line number. Press CONTROL-D to scroll the screen down (forward) through the file so that vim displays half a screen of new text. Use CONTROL-U to scroll the screen up (backward) by the same amount. If you precede either of these commands with a number, vim scrolls that number of lines each time you press CONTROL-D or CONTROL-U for the rest of the session (unless you again change the number of lines to scroll). The CONTROL-F (forward) and CONTROL-B (backward) keys display almost a whole screen of new text, leaving a couple of lines from the previous screen for continuity. On many keyboards you can use the PAGE DOWN and PAGE UP keys in place of CONTROL-F and CONTROL-B, respectively.

Deleting and Changing of Text

- Undoing changes
- Deleting characters
- Deleting text
- Changing text
- Replacing text
- Changing case

Undoing Changes

The u command (Undo) restores text that you just deleted or changed by mistake. A single Undo command restores only the most recently deleted text. With the compatible parameter set, vim can

Linux and Shell Scripting

undo only the most recent change. The U command restores the last line you changed to the way it was before you started changing it, even after several changes.

Deleting Characters

The x command deletes the current character. You can precede the x command by a Repeat Factor to delete several characters on the current line, starting with the current character. The X command deletes the character to the left of the cursor.

Deleting Text

The d (Delete) command removes text from the Work buffer. The amount of text that d removes depends on the Repeat Factor and the Unit of Measure. After the text is deleted, vim is still in Command mode.

Command	Result
dl	Deletes current character (same as the x command)
d0	Deletes from the beginning of line
d^	Deletes from first character of line
dw	Deletes to end of word
d3w	Deletes to end of third word
db	Deletes from beginning of word
dw	Deletes to end of blank delimited word
dB	Deletes to beginning of blank delimited word
d7B	Deletes from seventh previous beginning of blank delimited word
d)	Deletes to end of sentence
d4)	Deletes to end of fourth sentence
d{	Deletes from beginning of sentence
d}	Deletes to end of paragraph
d{	Deletes from beginning of paragraph
d7{	Deletes from 7 th paragraph preceding from beginning of paragraph
d/text	Deletes upto next occurrence of word text
dfc	Deletes on current line upto and including next occurrence of character c
dtc	Deletes on current line upto next occurrence of c
D	Deletes to end of line
D\$	Deletes to end of line

Unit 07: The Shell and Popular Editors

dd	Deletes current line
5dd	Deletes five lines starting with current line
dL	Deletes through last line on screen
dH	Deletes from first line on screen
dG	Deletes through end of work buffer
D1G	Deletes from beginning of work buffer

Changing Text

The c (Change) command replaces existing text with new text. The new text does not have to occupy the same space as the existing text. You can change a word to several words, a line to several lines, or a paragraph to a single character. The C command replaces the text from the cursor position to the end of the line. The c command deletes the amount of text specified by the Repeat Factor and the Unit of Measure and puts vim in Input mode. When you finish entering the new text and press ESCAPE, the old word, line, sentence, or paragraph is changed to the new one. Pressing ESCAPE without entering new text deletes the specified text (that is, it replaces the specified text with nothing).

Command	Result
cl	Changes current character
cw	Changes to end of word
c3w	Changes to end of third word
cb	Changes from beginning of word
cW	Changes from end of blank delimited word
cB	Changes from beginning of blank delimited word
C7B	Changes from beginning of seventh previous blank delimited word
C\$	Changes to end of line
c0	Changes from beginning of line
c)	Changes to end of sentence
c4)	Changes to end of fourth sentence
c(Changes from beginning of sentence
c}	Changes to end of paragraph
c{	Changes from beginning of paragraph
c7{	Changes from beginning of seventh preceding paragraph
ctc	Changes of current line upto next occurrence of c
C	Changes to end of line
cc	Changes current line

Replacing Text

The s and S (Substitute) commands also replace existing text with new text. The s command deletes the current character and puts vim into Input mode. It has the effect of replacing the current character with whatever you type until you press ESCAPE. The S command does the same thing as the cc command: It changes the current line. The s command replaces characters only on the current line. If you specify a Repeat Factor before an s command and this action would replace more characters than are present on the current line, s changes characters only to the end of the line (same as C).

Command	Result
s	Substitute one or more characters for current character
S	Substitute one or more characters for current line
5s	Substitute one or more characters for five characters, starting with current character

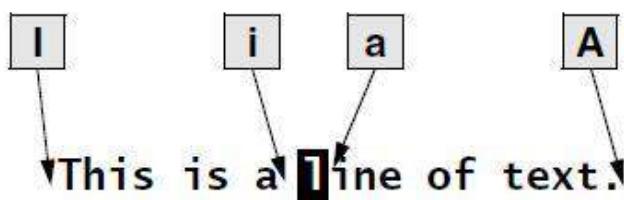
Changing Case

The tilde (~) character changes the case of the current character from uppercase to lowercase, or vice versa. You can precede the tilde with a number to specify the number of characters you want the command to affect. For example, the command 5~ transposes the next five characters starting with the character under the cursor but will not transpose characters past the end of the current line.

7.13 Input Mode

The Insert, Append, Open, Change, and Replace commands put vim in Input mode. While vim is in this mode, you can put new text into the Work buffer. To return vim to Command mode when you finish entering text, press the ESCAPE key.

Inserting Text



Key	Results
i	Put vim in the insert mode and places the text before the current character.
I	Places the text at the beginning of the current line.
a	Places the text after the current character.
A	Places the text after the current line.
o	Opens a new line below the current line.
O	Opens a new line above the current line.

Replacing Text

The r and R (Replace) commands cause the new text you enter to overwrite (replace) existing text. The single character you enter following an r command overwrites the current character. After you enter that character, vim returns to Command mode—you do not need to press the ESCAPE key. The R command causes all subsequent characters to overwrite existing text until you press ESCAPE to return vim to Command mode.

Quoting Special Characters in Input Mode

While you are in Input mode, you can use the Quote command, CONTROL-V, to enter any character into the text, including characters that normally have special meaning to vim. Among these characters are CONTROL-L (or CONTROL-R), which redraws the screen; CONTROL-W, which backs the cursor up a word to the left; CONTROL-M, which enters a NEWLINE; and ESCAPE, which ends Input mode. To insert one of these characters into the text, type CONTROL-V followed by the character. CONTROL-V quotes the single character that follows it.

7.14 Emacs

Emacs is one of the oldest and most versatile text editors available for Linux and UNIX-based systems. It's been around for a long time (more than twenty years for GNU emacs) and is well known for its powerful and rich editing features. The emacs editor, which is coded in C, contains a complete Lisp interpreter and fully supports the X Window System and mouse interaction. Version 22 has significant internationalization upgrades: an extended UTF-8 internal character set four times bigger than Unicode, along with fonts and keyboard input methods for more than 30 languages. Also, the user interface is moving in the direction of a WYSIWYG (what you see is what you get) word processor, which makes it easier for beginners to use the editor. You never need to switch emacs between Input and Command modes, emacs is a modeless editor.

emacs Versus vim

- 1) Like vim, emacs is a display editor: It displays on the screen the text you are editing and changes the display as you type each command or insert new text.
- 2) Unlike vim, emacs does not require you to keep track of whether you are in Command mode or Insert mode: Commands always use CONTROL or other special keys.
- 3) The emacs editor inserts ordinary characters into the text you are editing (as opposed to using ordinary characters as commands), another trait of modeless editing. For many people this approach is convenient and natural.
- 4) As with vim, you use emacs to edit a file in a work area, or buffer, and have the option of writing this buffer back to the file on the disk when you are finished. With emacs, however, you can have many work buffers and switch among them without having to write the buffer out and read it back in.
- 5) Like vim, emacs has a rich, extensive command set for moving about in the buffer and altering text. This command set is not "cast in concrete"—you can change or customize commands at any time.
- 6) Finally, and very unlike vim, emacs allows you to use Lisp to write new commands or override old ones. This feature is called online extensibility,

Getting Started with emacs

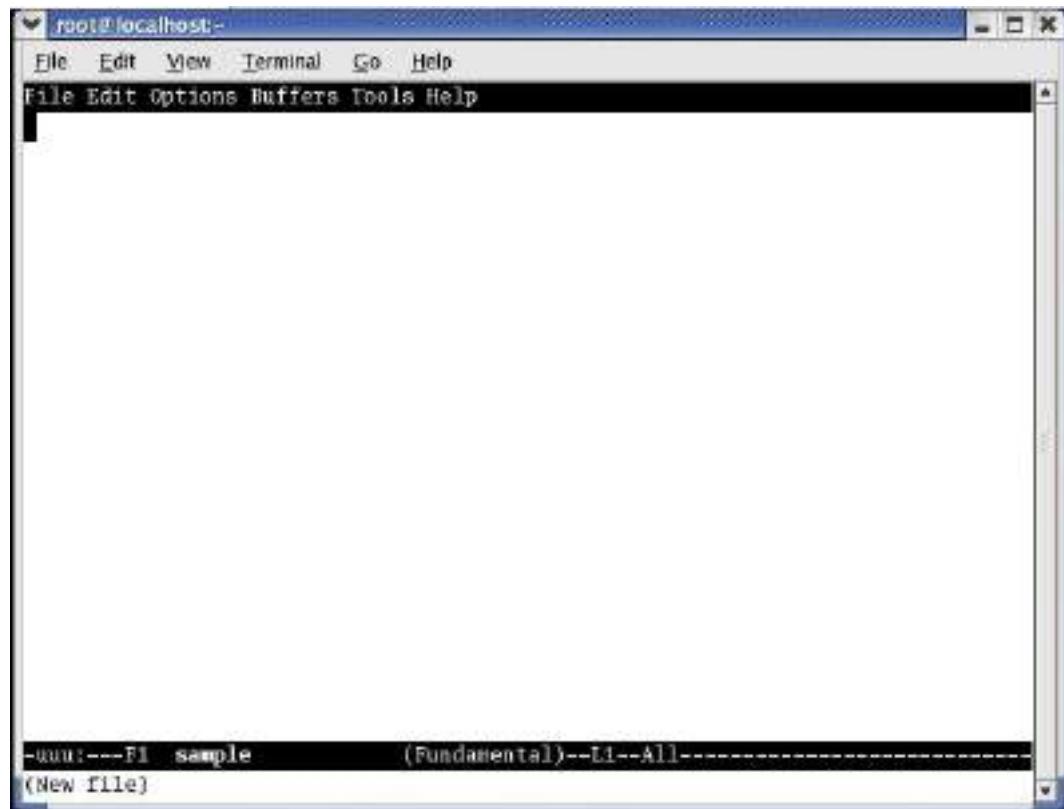
To edit a file named sample using emacs as a text-based editor, enter the following command:\$ emacs -nw -q sample.

- **-nw option:** It must be the first option on the emacs command line, tells emacs not to use its X interface (GUI).

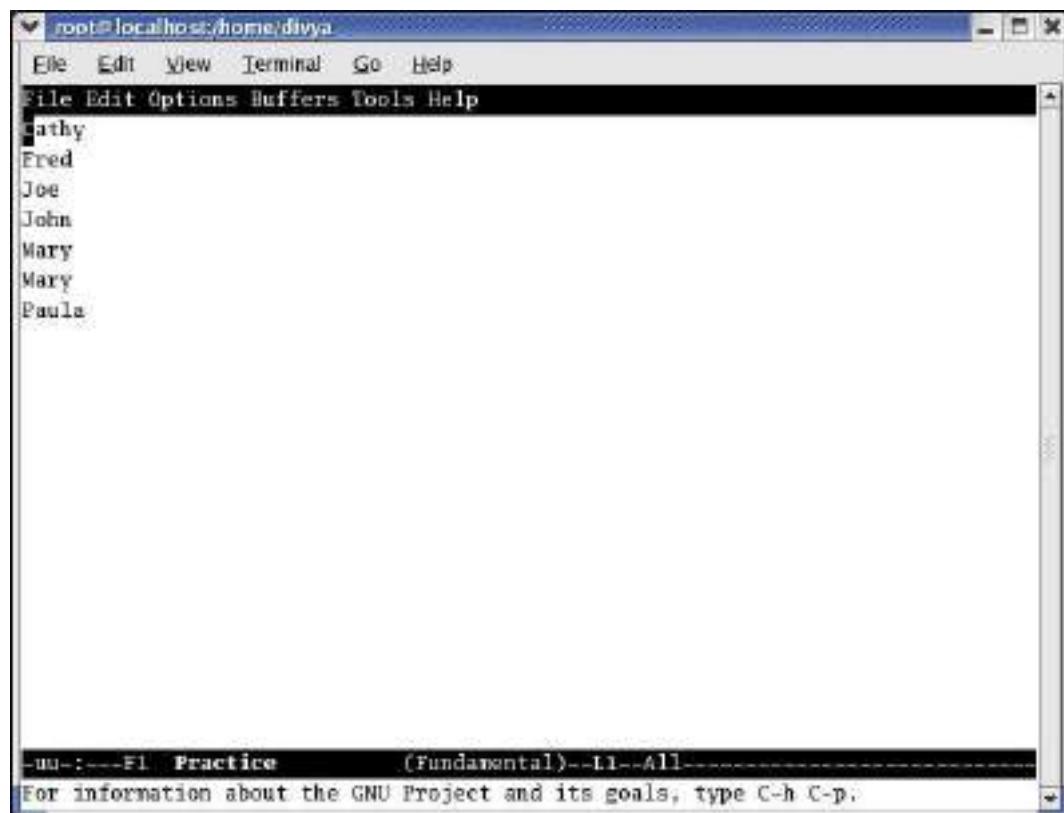
Linux and Shell Scripting

- **-q option:** It tells emacs not to read the `~/.emacs` startup file. Not reading this file guarantees that emacs will behave in a standard manner and can be useful for beginners or for other users who want to bypass a `.emacs` file.

The command `$ emacs -nw -q sample` starts emacs, reads the file named `sample` into a buffer, and displays its contents on the screen or window. If no file has this name, emacs displays a blank screen with (New File) at the bottom.

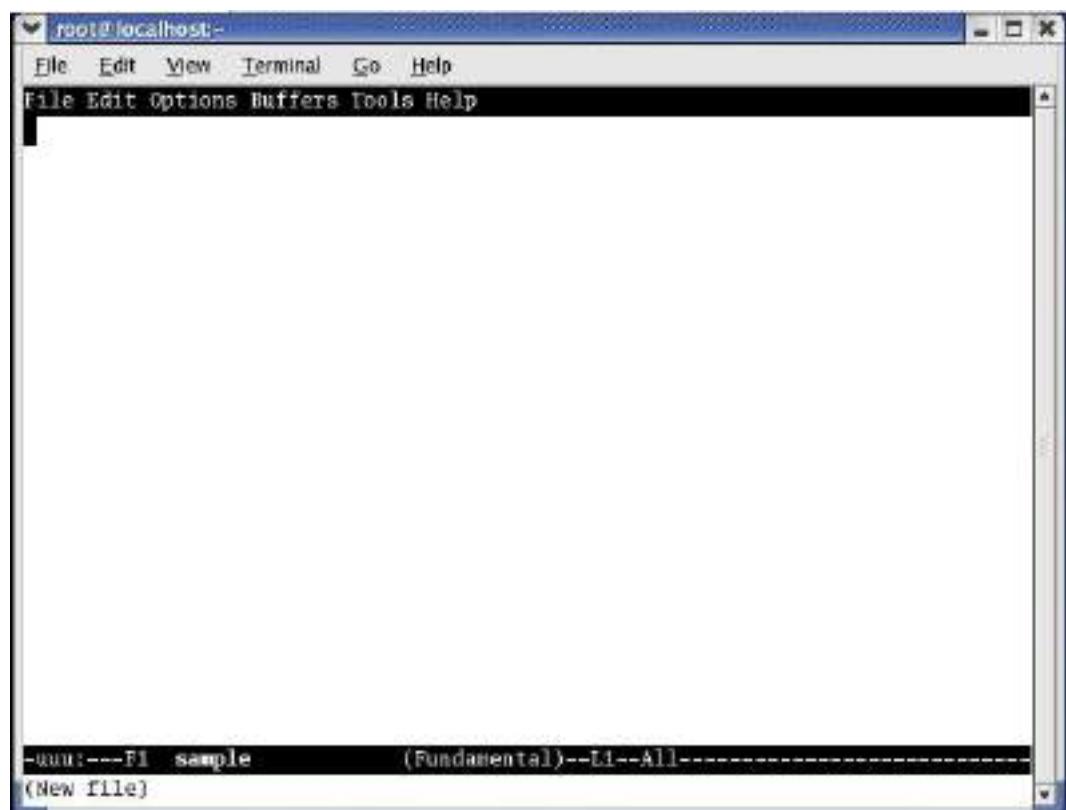
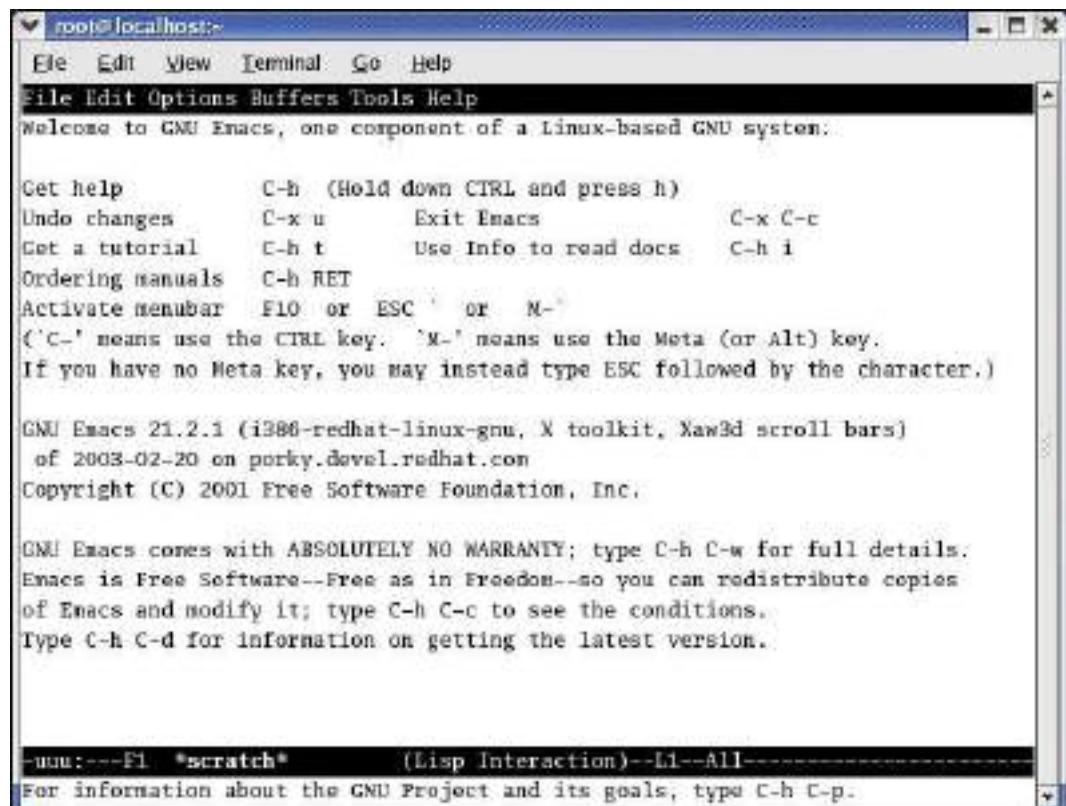


If the file exists, emacs displays the file and a different message.



Unit 07: The Shell and Popular Editors

If you start emacs without naming a file on the command line, it displays a welcome screen that includes usage information and a list of basic commands.



Initially, emacs displays a single window. At the top of the window is a reverse-video menubar that you can access using a mouse or keyboard. From the keyboard, F10, META-` (back tick), or META-x tmm-menubar RETURN displays the Menubar Completion List window. At the bottom of the emacs window is a reverse-video titlebar called the Mode Line. At a minimum, the Mode Line

Linux and Shell Scripting

shows which buffer the window is viewing, whether the buffer has been changed, which major and minor modes are in effect, and how far down the buffer the window is positioned. When multiple windows appear on the screen, one Mode Line appears in each window. At the bottom of the screen, emacs leaves a single line open. This Echo Area and Minibuffer line (they coexist on one line) is used for messages and special one-line commands. If you make an error while you are typing in the Minibuffer, emacs displays the error message in the Echo Area. The error message overwrites the command you were typing, but emacs restores the command in a few seconds. A cursor is either in the window or in the Minibuffer. All input and nearly all editing take place at the cursor. As you type ordinary characters, emacs inserts them at the cursor position. If characters are under the cursor or to its right, they are pushed to the right as you type, so no characters are lost.

Exiting

The command to exit from emacs is CONTROL-X CONTROL-C. If you want to cancel a half-typed command or stop a running command before it is done, press CONTROL-G. The emacs editor displays Quit in the Echo Area and waits for another command.

Inserting Text

Typing an ordinary (printing) character pushes the cursor and any characters to the right of the cursor one position to the right and inserts the new character in the space opened by moving the characters.

Deleting Characters

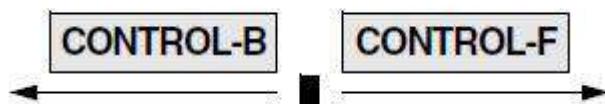
Depending on the keyboard and the emacs startup file, different keys may delete characters in different ways.

- CONTROL-D typically deletes the character under the cursor, as do DELETE and DEL.
- BACKSPACE typically deletes the character to the left of the cursor.

Moving the Cursor

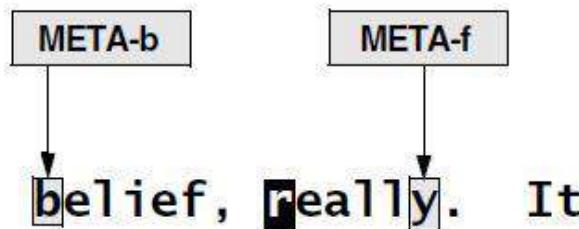
You can position the cursor over any character in the emacs window and move the window, so it displays any portion of the buffer. You can move the cursor forward or backward through the text various textual units—for example, characters, words, sentences, lines, and paragraphs.

Moving the Cursor by Characters



Key	Results
Control - B / Left Arrow Key	Moves the cursor backward one character.
Control - F / Right Arrow Key	Moves the cursor forward one character.

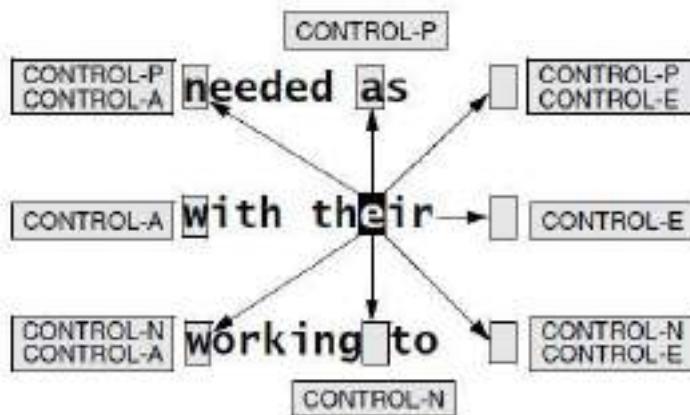
Moving the Cursor by Words



META-f: Moves the cursor forward one word. To invoke this command, hold down the META or ALT key while you press f. If the keyboard you are using does not have either of these keys, press ESCAPE, release it, and then press f.

META-b: Moves the cursor backward one word, leaving the cursor on the first letter of the word it started on.

Moving the Cursor by Lines



Key	Result
CONTROL-A/CONTROL-P	Moves the cursor to the beginning of the line it is on
CONTROL-E/CONTROL-P	Moves it to the end
UP ARROW key or CONTROL-P	Moves the cursor up one line to the position directly above where the cursor started
DOWN ARROW key or CONTROL-N	Moves the cursor down one line to the position directly above where the cursor started

Moving the Cursor by Sentences, Paragraphs, and Window Position



Key	Results
META-a	It moves the cursor to the beginning of the sentence the cursor is on
META-e	It moves the cursor to the end of the sentence the cursor is on.
META-{	It moves the cursor to the beginning of the paragraph the cursor is on.
META-}	It moves it to the end of the paragraph the cursor is on.
META-r	It moves the cursor to the beginning of the middle line of the window.
CONTROL-U META-r	It moves the cursor to the beginning of the top line (line zero) in the window.
CONTROL-U- (minus sign)	The command moves the cursor to the beginning of the last line of the window

Editing at the Cursor Position

Entering text requires no commands once you position the cursor in the window at the location you want to enter text. When you type text, emacs displays that text at the position of the cursor. Any text under or to the right of the cursor is pushed to the right. Pressing BACKSPACE removes characters to the left of the cursor. The cursor and the remainder of the text on this line both move to the left each time you press BACKSPACE. To join a line with the line above it, position the cursor on the first character of the second line and press BACKSPACE. Press CONTROL-D to delete the character under the cursor.

Saving and Retrieving the Buffer

No matter what changes you make to a buffer during an emacs session, the associated file does not change until you save the buffer. If you leave emacs without saving the buffer (emacs allows you to do so if you are persistent), the file is not changed and emacs discards the work you did during the session. The command CONTROL-X CONTROL-S saves the current buffer in its associated file. The emacs editor confirms a successful save by displaying an appropriate message in the Echo Area.

The emacs GUI

Full mouse support was introduced in version 19. The emacs editor is still evolving toward full internationalization, accessibility, and a simplified user experience that appeals to the widest possible audience. New features include a colorful menubar and toolbar, mixed text and graphics, tooltips, drag and drop editing, and new GUIs for browsing directories and selecting fonts.

Basic Editing Commands

Command functions in emacs usually involve two or three keys. The most common is the Ctrl key, followed by the Alt or Esc key. In emacs literature, Ctrl is shown in short form as "C". So if you see something like C-x C-c, it means "press the Ctrl key and x together, then press Ctrl and c". Similarly, if you see C-h t, it means "press Ctrl and h together, then release both keys and press t". Alt and Esc keys are referred to as "meta" key in emacs lingo. So if you see a notation like M-x, it means "press Alt/Esc/Option/Edit key and x together". The Enter key is shown as RET (short for "Return") and The Esc key is often shown as E.

Marking Text Regions

To mark a text region (like selecting text in popular word processors), follow these steps:

Unit 07: The Shell and Popular Editors

-
- 1) Move the cursor to the position where you would like the selection to start Press **C-Space** (Ctrl + Space Bar) or **C-@** to set a mark. The mini buffer will show a status message of Mark set.
 - 2) Move the cursor to the position where you want the region to end.
 - 3) The text will be highlighted up to the point where your cursor is now located.
 - 4) If you want to “un-mark” the highlighted text, press **C-Space** or **C-@** twice The mini buffer will show a status message of Mark deactivated.

Cut, Copy & Paste

Once you have the text region marked, you can copy or cut the text and paste it elsewhere. For copying the text, press **E-w**. For cutting the text, press **C-w**. Move your cursor to the position where the text needs to be pasted. Press **C-y** (y stands for “yank” - you are yanking the text from one position to another). The contents will be pasted here.

Deleting Text

For deleting, Backspace and Delete keys work just the way you would expect them to work. For deleting a whole word, move the cursor at the beginning of a word and press **M-d**. For deleting multiple words, hold the meta key down and keep pressing d. Words will start deleting one by one. For deleting a whole line, position the cursor where you want it to be and press **C-k**. This would delete the text right up to the end of the line on screen. For deleting a sentence, press **M-k**

Undo & Redo

Undoing the last operation is simple. Press **C-x u**. You can keep repeating this to go backwards. Another key combination is **C-/** (Ctrl + /) or **C-_** (Ctrl + _). For redoing your last undo, press **C-g**, followed by **C-_** (that's Ctrl + Shift+ Underscore). Another way to do the same thing would be to press **C-x C-u** again (Undoing the Undo).

Search & Replace

There are two search directions in emacs: forward and backward. In forward search, the word you specify will be searched forward from the current cursor position. For backward search, it's the other way round. Press **C-s** for forward search and press **C-r** for backward search.

For replacing text, follow these steps:

1. Press **M-%** (that's Alt + Shift + %). The mini buffer shows the prompt for the text to be searched (Query replace:).
2. Type the text and press Enter. The mini buffer displays a message like (Query replace <search_word> with:). Type the replacing text and press Enter.
3. For each match emacs finds, it will ask whether you would like to make a replacement (Query replacing <search_word> with <replace_word>: (C-h for help)). You can take any of the following actions:Press y to replace the current match found/ press n to skip to the next match/ press q to exit without any replacements (basically escaping)/ press ! to do a global replace without any prompts. (emacs will show a message like replaced n occurrences)

Left, Right and Centre Alignment

For justifying a selected text region, follow these steps:

1. Create a text region to highlight the text you wish to justify
2. Press **M-x**. The mini buffer will await a response
3. Start typing **set-justification-** and press Tab.
4. You will be given completion options like **set-justification-center**, **set-justification-left**, **set-justification-right**, **set-justification-none** and **set-justification-full**

Linux and Shell Scripting

5. Complete the justification command (for example set-justification-right) and press Enter.
6. The selected text will be justified.

Converting Case

Command	Result
M-c (c for Capitalize)	Capitalizing a word after the cursor
M-l (l for Lower case)	Converting a word to lower case
M-u (u for Upper case)	Converting a word to upper case
Block select, then C-x C-u	Converting a paragraph to upper case
Block select, then C-x C-l	Converting a paragraph to lower case

Summary

- Kernel is the innermost part of an operating system; a shell is the outermost one.
- The line that contains the command, including any arguments, is called the command line.
- By default, the shell directs standard output from a command to the screen.
- In addition to standard input and standard output, a running program normally has a place to send error messages: standard error.
- The redirect output symbol (>) instructs the shell to redirect the output of a command to the specified file instead of to the screen.
- The redirect input symbol (<) instructs the shell to redirect a command's input to come from the specified file instead of from the keyboard.
- A filter is a command that processes an input stream of data to produce an output stream of data.
- When you run a command in the foreground, the shell waits for it to finish before displaying another prompt and allowing you to continue. When you run a command in the background, you do not have to wait for the command to finish before running another command.
- You can suspend a foreground job (stop it from running without aborting the job) by pressing the suspend key, usually CONTROL-Z.
- The vim editor is not a text formatting program. It is a sophisticated text editor meant to be used to write code (C, HTML, Java, etc.), short notes, and input to a text formatting system, such as groff or troff.
- If you call vim without specifying a filename on the command line, vim assumes that you are a novice and tells you how to get started.
- There are three modes in vim: command mode, input mode and last line mode.
- To put vim in Input mode, press the i (insert before cursor) key or the a (append after cursor) key.
- In vim to correct text, use dd, dw, or x to remove the incorrect text. Then use i, a, o, or O to insert the correct text.
- There are various features of vim: online help, modes of operation, work buffer, emergency procedures, etc.

Unit 07: The Shell and Popular Editors

- The emacs editor, which is coded in C, contains a complete Lisp interpreter, and fully supports the X Window System and mouse interaction. You never need to switch emacs between Input and Command modes, emacs is a modeless editor.

Keywords

- **Kernel:** Kernel is at the nucleus of a computer. It makes the communication between the hardware and software possible.
- **Shell:** A shell in a Linux operating system takes input from you in the form of commands, processes it, and then gives an output.
- **Token:** On the command line each sequence of nonblank characters is called a token or word.
- **Standard output:** It is a place a program can send information, such as text.
- **Standard input:** It is a place that a program gets information from.
- **Redirection:** The term redirection encompasses the various ways you can cause the shell to alter where standard input of a command comes from and where standard output goes to.
- **noclobber:** The shell provides the noclobber feature that prevents overwriting a file using redirection.
- **Bit bucket:** The /dev/null device is a data sink, commonly referred to as a bit bucket.
- **Pipe:** The shell uses a pipe to connect standard output of one command to standard input of another command.
- **tee utility:** The tee utility copies its standard input both to a file and to standard output.
- **Job:** A job is a series of one or more commands that can be connected by pipes. You can have only one foreground job in a window or on a screen, but you can have many background jobs.
- **? special character:** The question mark (?) is a special character that causes the shell to generate filenames. It matches any single character in the name of an existing file.
- *** special character:** The asterisk (*) performs a function similar to that of the question mark but matches any number of characters, including zero characters, in a filename.
- **Builtin:** A builtin is a utility (also called a command) that is built into a shell.

Self Assessment

1. Which of these is used for erasing a word?
 A. CTRL-H
 B. CTRL-W
 C. CTRL-U
 D. None of the above
2. Standard _____ is a place that a program gets information from.
 A. Output
 B. Input
 C. Error
 D. None of the above
3. Which symbol is used for appending the standard output to a file?

Linux and Shell Scripting

A. >>

B. <<

C. !!

D. <>

4. Which of these components of operating system is the outermost part?

A. Kernel

B. Shell

C. CPU

D. None of the above

5. The redirection input symbol is

A. >

B. <

C. !

D. None of the above

6. What is another name of pathname expansion?

A. Local-ling

B. Globing

C. Met-forcing

D. None of the above

7. A _____ within a bracket defines a range.

A. Hyphen

B. Underscore

C. Asterisk

D. None of the above

8. How many jobs we can run in foreground?

A. 0

B. 1

C. 2

D. 3

9. Which of these is a suspend key which can suspend a foreground job?

A. CTRL-Z

B. CTRL-D

C. CTRL-C

D. CTRL-A

10. Which of these symbols puts the vim in last line mode?

A. :

B. ;

C. "

D. ?

11. Which of these modes are available in vim?

A. Command mode

B. Input mode

C. Last line mode

D. All of the above mentioned

12. Which of these characters delete a character?

A. x

B. y

C. z

D. None of the above

13. Which of these keys can be used to move the cursor backward by one character?

A. h

B. i

C. j

D. k

14. For backward search, press

A. C-s

B. C-b

C. C-r

D. C-c

15. The shortcut for exiting from emacs is

A. CTRL-X CTRL-C

B. CTRL-X CTRL-X

C. CTRL-C CTRL-C

D. CTRL-C CTRL-X

Answers for Self Assessment

1. B 2. B 3. A 4. B 5. B

6. B 7. A 8. B 9. A 10. A

11. D 12. A 13. A 14. C 15. A

Review Questions:

1. What is an operating system? Describe its main components: shell and kernel in detail.

Linux and Shell Scripting

2. What is a command? What are the rules for writing a command? Explain the components of a command.
3. What are standard input and standard output? Explain redirection and noclobber.
4. Explain pipes and filters.
5. How can we run a command in background? Explain with commands. How to move a job from foreground to background?
6. What is filename generation? Which special characters are used in this? Explain the difference of usage of each character.
7. What is a builtin? How can we list out bash and tschbuiltins?
8. What is vim? How can we use vim? Explain the starting, entering of text, moving of cursor, correction of text and exiting in vim with examples and commands.
9. What are modes in vim? Explain its usage.
10. Explain the various features of vim in detail.
11. What is command mode? How can we move the cursor in this?
12. How can we delete and change text in command mode? Explain with commands.
13. What is emacs? Explain its difference with vim.
14. What are basic editing commands in emacs? Explain.



Further Readings

<https://www.informit.com/articles/article.aspx?p=2854374&seqNum=5>

<https://unix.stackexchange.com/questions/986/what-are-the-pros-and-cons-of-vim-and-emacs>

<https://stackoverflow.com/questions/1430164/differences-between-emacs-and-vim>

Unit 08: The Bourne Again Shell and Tc Shell

CONTENTS

Objectives
Introduction
8.1 Startup Files
8.2 File Descriptors
8.3 Writing a Simple Shell Script
8.4 Job Control
8.5 Manipulating the Directory Stack
8.6 Shell Variables
8.7 Variable Attributes
8.8 Keyword Variables
8.9 Processes
8.10 History
8.11 Alias
8.12 Functions
8.13 Command Line Options
8.14 Shell Features
8.15 The TC Shell
Summary
Keywords
Self Assessment
Answers for Self Assessment
Review Questions
Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the basics of shell, parameters and variables of a shell
- Understand the special characters
- Understand about the process
- Understand the re-execution and editing of commands
- Understand the aliases and functions in Linux
- Know how to control bash, how to enter and leave the TC shell
- Understand the features common to Bourne Again and TC Shells

Introduction

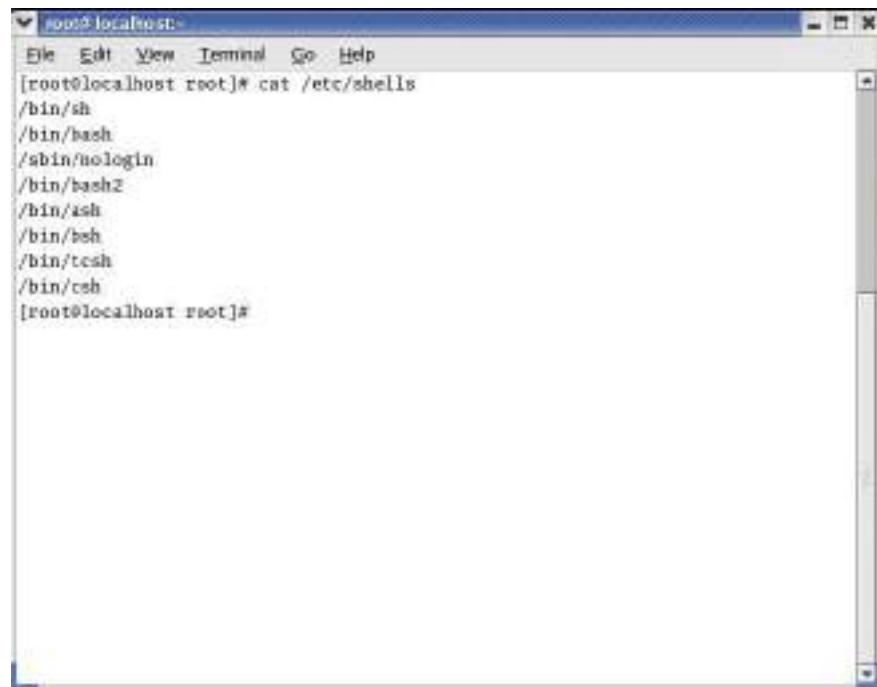
A Shell provides you with an interface to the Unix system. It gathers input from you and executes programs based on that input. When a program finish executing, it displays that program's output. Shell is an environment in which we can run our commands, programs, and shell scripts.

The Linux and Shell Scripting

There are different flavors of a shell, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions. There are two major types of shells –

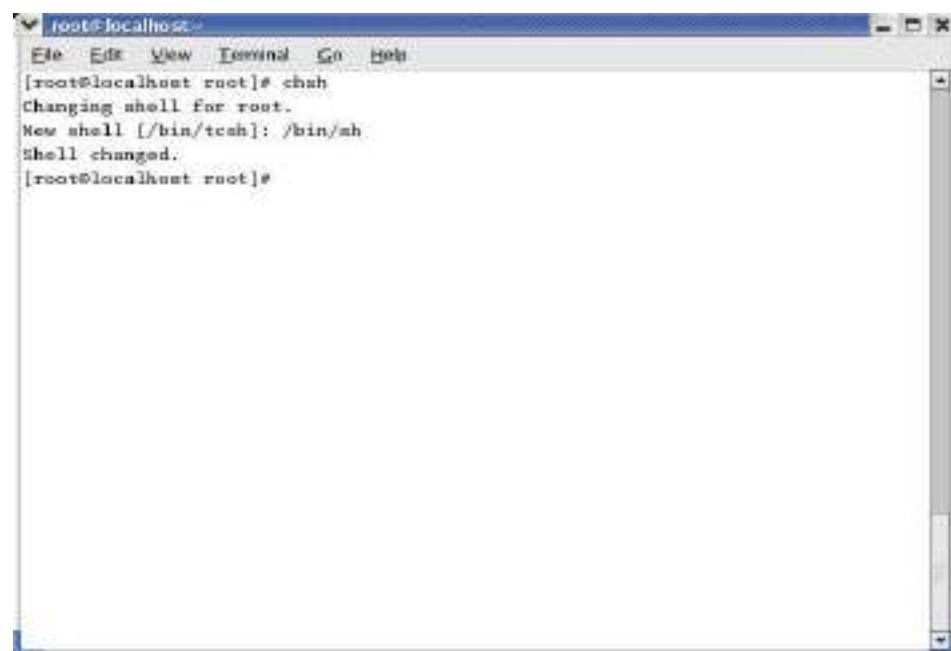
- Bourne Again shell
- TC shell

The Bourne Again Shell and TC Shell are command interpreters and high-level programming languages. As command interpreters, they process commands you enter on the command line in response to a prompt. When you use the shell as a programming language, it processes commands stored in files called shell scripts. Give the command **echo "\$SHELL"**. Under most Linux distributions, bash is the default shell. You can run any shell you like once you are logged in. When the command **cat /etc/shells** is given, it shows which shells are available to use in the system.

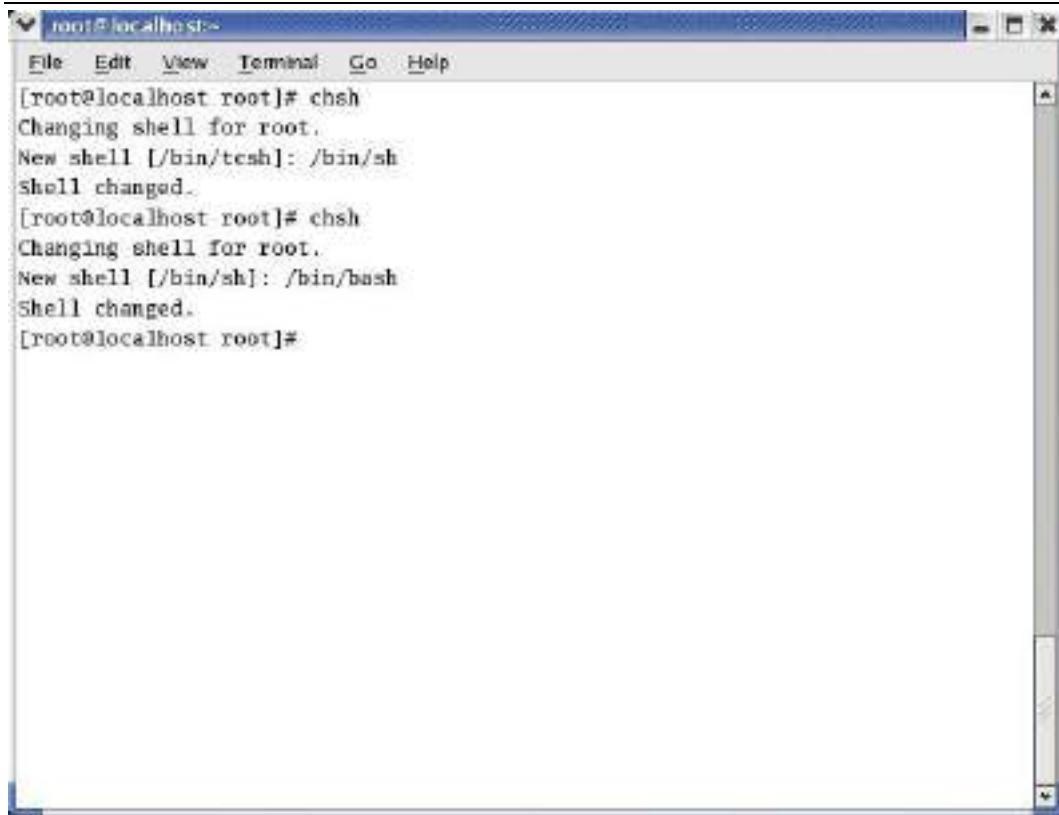


```
[root@localhost root]# cat /etc/shells
/bin/sh
/bin/bash
/sbin/nologin
/bin/bash2
/bin/ash
/bin/sh
/bin/tcsh
/bin/csh
[root@localhost root]#
```

Enter the name of the shell you want to use (bash, tcsh, or another shell) and press RETURN; the next prompt will be that of the new shell. Give an exit command to return to the previous shell.



```
[root@localhost root]# chsh
Changing shell for root.
New shell [/bin/tcsh]: /bin/sh
Shell changed.
[root@localhost root]#
```



The screenshot shows a terminal window titled "root@localhost". The menu bar includes "File", "Edit", "View", "Terminal", "Go", and "Help". The terminal content shows the following session:

```
[root@localhost root]# chsh
Changing shell for root.
New shell [/bin/tcsh]: /bin/sh
shell changed.
[root@localhost root]# chsh
Changing shell for root.
New shell [/bin/sh]: /bin/bash
Shell changed.
[root@localhost root]#
```

8.1 Startup Files

When a shell starts, it runs startup files to initialize itself. Which files the shell runs depends on whether it is a login shell, an interactive shell that is not a login shell, or a noninteractive shell. You must have read access to a startup file to execute the commands in it.

Login Shells

The shells that you start with the bash --login option. Login shells are, by their nature, interactive.

1) /etc/profile: The shell first executes the commands in /etc/profile. A user working with root privileges can set up this file to establish system-wide default characteristics for users running bash.

2) .bash_profile, .bash_login, .profile: Next the shell looks for ~/.bash_profile, ~/.bash_login, and ~/.profile (~/ is shorthand for your home directory), in that order, executing the commands in the first of these files it finds. You can put commands in one of these files to override the defaults set in /etc/profile.

3) .bash_logout: When you log out, bash executes commands in the ~/.bash_logout file. This file often holds commands that clean up after a session, such as those that remove temporary files.

Interactive Nonlogin Shells

The commands in these startup files are not executed by interactive, nonlogin shells. However, these shells inherit values from the login shell variables that are set by these startup files.

1) /etc/bashrc: Although not called by bash directly, many ~/.bashrc files call /etc/bashrc. This setup allows a user working with root privileges to establish systemwide default characteristics for nonlogin bash shells.

2) .bashrc: An interactive nonlogin shell executes commands in the ~/.bashrc file. Typically, a startup file for a login shell, such as .bash_profile, runs this file, so both login and nonlogin shells run the commands in .bashrc.

The Linux and Shell Scripting

However, the noninteractive shells inherit login shell variables that are set by these startup files. Noninteractive shells look for the environment variable BASH_ENV (or ENV if the shell is called as sh) and execute commands in the file named by this variable.

There are various commands that are used as symbols. The Bourne Again Shell uses these symbols as shown in the below table. A command can send error messages to standard error to keep them from getting mixed up with the information it sends to standard output.

Symbol	Command
()	Subshell
\$()	Command Substitution
(())	Arithmetic evaluation
\$(())	Arithmetic expression
[]	The test command
[[]]	Conditional expression; similar to [] but adds string comparisons

8.2 File Descriptors

A file descriptor is the place a program sends its output to and gets its input from. When you execute a program, Linux opens three file descriptors for the program:

- 0 (standard input), (0<)
- 1 (standard output), (1>)
- and 2 (standard error), (2>)

Opening a File Descriptor

The Bourne Again Shell opens files using the exec built in as follows:

- exec n>outfile: The line opens outfile for output and holds it open, associating it with file descriptor n.
- exec m<infile: The line opens infile for input and holds it open, associating it with file descriptor m.

Duplicating a file descriptor

The <& token duplicates an input file descriptor; >& duplicates an output file descriptor.

Redirection Operators

There are various redirection operators which can be used. These are shown in the table below.

Operator	Meaning
< filename	Redirects standard input from filename
>filename	Redirects standard output to filename unless filename exists and noclobber is set. If noclobber is not set, this redirection creates filename if it does not exist and overwrites it if it does exist.
> filename	Redirects standard output to filename, even if the file exists and noclobber is set.
>> filename	Redirects and appends standard output to filename unless filename exists and noclobber is set. If noclobber is not set, this redirection creates filename if it does not exist.
&> filename	Redirects standard output and standard error to filename.
<&m	Duplicates standard input from file descriptor m
[n] >&m	Duplicates standard output or file descriptor n if specified from file descriptor m
[n] <&-	Closes standard input or file descriptor n if specified
[n] >&-	Closes standard output or file descriptor n if specified.

8.3 Writing a Simple Shell Script

A shell script is a file that holds commands that the shell can execute. The commands in a shell script can be any commands you can enter in response to a shell prompt. For example, a command in a shell script might run a Linux utility, a compiled program, or another shell script. Like the commands you give on the command line, a command in a shell script can use ambiguous file references and can have its input or output redirected from or to a file or sent through a pipe. You can also use pipes and redirection with the input and output of the script itself.

chmod: Makes a File Executable

To execute a shell script by giving its name as a command, you must have permission to read and execute the file that contains the script. Read permission enables you to read the file that holds the script. Execute permission tells the shell and the system that the owner, group, and/or public has permission to execute the file; it implies that the content of the file is executable. When you create a shell script using an editor, the file does not typically have its execute permission set. The following example shows a file named whoson that contains a shell script:

```
$ cat whoson
date
echo "Users Currently Logged In"
```

The Linux and Shell Scripting

who

\$./whoson

```
S ls -l whoson
-rw-r-- 1 max group 40 May 24 11:30 whoson

S chmod u+x whoson
S ls -l whoson
-rwxr-- 1 max group 40 May 24 11:30 whoson

S ./whoson
Fri May 22 11:40:49 PDT 2009
Users Currently Logged In
zach pts/7 May 21 18:17
hls pts/1 May 22 09:59
sam pts/12 May 22 06:29 (bravo.example.com)
max pts/4 May 22 09:08
```

#! Specifies a Shell

You can put a special sequence of characters on the first line of a shell script to tell the operating system which shell (or other program) should execute the file. Because the operating system checks the initial characters of a program before attempting to execute it using exec, these characters save the system from making an unsuccessful attempt.

Begins a Comment

Comments make shell scripts and all code easier to read and maintain by you and others. The comment syntax is common to both the Bourne Again and the TC Shells.

Executing a Shell Script

A command on the command line causes the shell to fork a new process, creating a duplicate of the shell process (a subshell). The new process attempts to exec (execute) the command. Like fork, the exec routine is executed by the operating system (a system call).

Separating and Grouping Commands

Whether you give the shell commands interactively or write a shell script, you must separate commands from one another.

8.4 Job Control

A job is a command pipeline. For example: you run a simple job whenever you give the shell a command. If you type date on the command line and press RETURN, you have run a job. You can also create several jobs with multiple commands on a single command line. For example:

```
$ find . -print | sort | lpr& grep -l max /tmp/* >maxfiles&
[1] 18839
[2] 18876
```

jobs: Lists Jobs

The jobs built in lists all background jobs. Following, the sleep command runs in the background and creates a background job that jobs reports on:

```
$ sleep 60 &
[1] 7809
$ jobs
[1] + Running sleep 60 &
```

fg: Brings a Job to the Foreground

The shell assigns a job number to each command you run in the background. For each job run in the background, the shell lists the job number and PID number immediately, just before it issues a prompt:

```
$ xclock&
[1] 1246
$ date &
[2] 1247
$ Tue Dec 2 11:44:40 PST 2008
[2]+ Done date
$ find /usr -name ace -print >findout&
[2] 1269
$ jobs
[1]- Running xclock&
[2]+ Running find /usr -name ace -print >findout&
```

To move a background job to the foreground, use the fgbuiltin followed by the job number. Alternatively, you can give a percent sign (%) followed by the job number as a command. Either of the following commands moves job 2 to the foreground. When you move a job to the foreground, the shell displays the command it is now executing in the foreground.

```
$ fg 2
find /usr -name ace -print >findout
or
$ %2
find /usr -name ace -print >findout
```

Suspending a Job

Pressing the suspend key (usually CONTROL-Z) immediately suspends (temporarily stops) the job in the foreground and displays a message that includes the word Stopped.

CONTROL-Z

```
[2]+ Stopped find /usr -name ace -print >findout
```

bg: Sends a Job to the Background

To move the foreground job to the background, you must first suspend the job using CONTROL-Z. You can then use the bg built in to resume execution of the job in the background.

```
$ bg
```

The Linux and Shell Scripting

[2]+ find /usr -name ace -print >findout&

8.5 Manipulating the Directory Stack

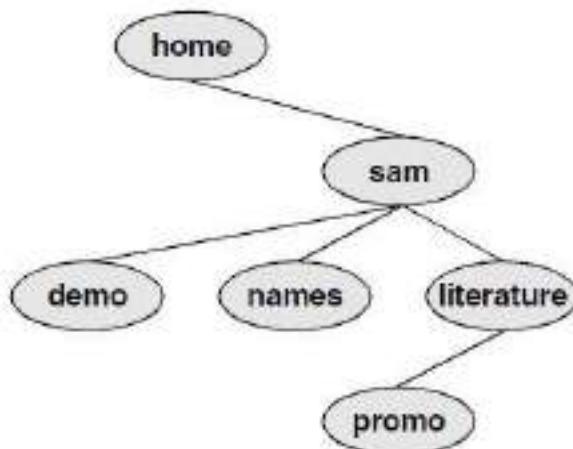
The Bourne Again Shell allows you to store a list of directories you are working with, enabling you to move easily among them. This list is referred to as a stack. It implements LIFO rule.

dirs: Displays the Stack

The dirs. Built in displays the contents of the directory stack. If you call dirs when the directory stack is empty, it displays the name of the working directory: \$ dirs

~/literature

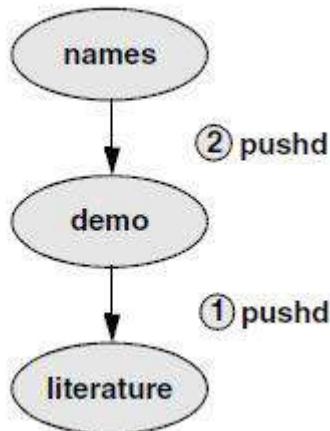
The dirsbuiltin uses a tilde (~) to represent the name of the home directory.



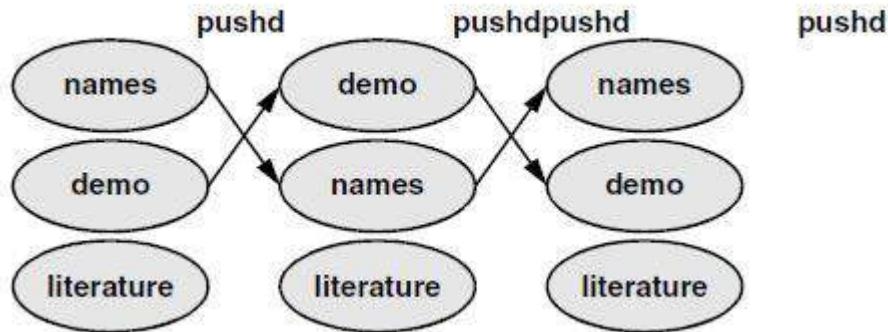
pushd: Pushes a Directory on the Stack

When you supply the pushd (push directory) builtin with one argument, it pushes the directory specified by the argument on the stack, changes directories to the specified directory, and displays the stack.

```
$ pushd ./demo  
~/demo ~/literature  
$ pwd  
/home/sam/demo  
$ pushd ./names  
~/names ~/demo ~/literature  
$ pwd  
/home/sam/names
```



When you use `pushd` without an argument, it swaps the top two directories on the stack, makes the new top directory (which was the second directory) the new working directory, and displays the stack



Using `pushd` in this way, you can easily move back and forth between two directories. You can also use `cd -` to change to the previous directory, whether you have explicitly created a directory stack. To access another directory in the stack, call `pushd` with a numeric argument preceded by a plus sign. The directories in the stack are numbered starting with the top directory, which is number 0.

popd: Pops a Directory Off the Stack

To remove a directory from the stack, use the `popd` (pop directory) builtin.

\$ dirs

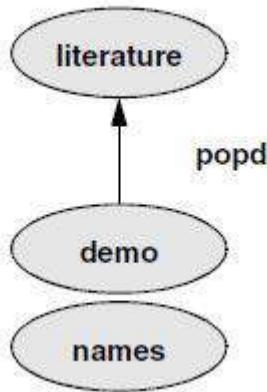
`~/literature ~/demo ~/names`

\$ popd

`~/demo ~/names`

\$ pwd

`/home/sam/demo`



To remove a directory other than the top one from the stack, use `popd` with a numeric argument preceded by a plus sign.

```
$ dirs
~/literature ~/demo ~/names
$ popd +1
~/literature ~/names
```

8.6 Shell Variables

Within a shell, a shell parameter is associated with a value that is accessible to the user. There are several kinds of shell parameters. The parameters whose names consist of letters, digits, and underscores are often referred to as shell variables, or simply variables. A variable name must start with a letter or underscore, not with a number. Thus `A76`, `MY_CAT`, and `_X_` are valid variable names, whereas `69TH_STREET` (starts with a digit) and `MY-NAME` (contains a hyphen) are not.

Keyword variables

Keyword shell variables (or simply keyword variables) have special meaning to the shell and usually have short, mnemonic names. When you start a shell (by logging in, for example), the shell inherits several keyword variables from the environment. Among these variables are `HOME`, which identifies your home directory, and `PATH`, which determines which directories the shell searches and in what order to locate commands that you give the shell. The shell creates and initializes (with default values) other keyword variables when you start it. Still other variables do not exist until you set them. You can change the values of most keyword shell variables. It is usually not necessary to change the values of keyword variables initialized in the `/etc/profile` or `/etc/csh.cshrc` systemwide startup files. If you need to change the value of a bash keyword variable, do so in one of your startup files. Just as you can make user-created variables global, so you can make keyword variables global—a task usually done automatically in startup files. You can also make a keyword variable read only.

Positional and special parameters

The names of positional and special parameters do not resemble variable names. Most of these parameters have one-character names (for example, `1`, `?`, and `#`) and are referenced (as are all variables) by preceding the name with a dollar sign (`$1`, `$?`, and `$#`). The values of these parameters reflect different aspects of your ongoing interaction with the shell. Whenever you give a command, each argument on the command line becomes the value of a positional parameter. Positional parameters enable you to access command-line arguments, a capability that you will often require when you write shell scripts. The `set` builtin enables you to assign values to positional parameters. Other frequently needed shell script values, such as the name of the last command executed, the number of command-line arguments, and the status of the most recently executed command, are available as special parameters. You cannot assign values to special parameters.

```
$ person=max
$ echo person
person
$ echo $person
max
```

User-created variables

Shell variables that you name and assign values to are user-created variables. You can change the values of user-created variables at any time, or you can make them read only so that their values cannot be changed.

User created global variables

You can also make user-created variables global. A global variable (also called an environment variable) is available to all shells and other programs you fork from the original shell. One naming convention is to use only uppercase letters for global variables and to use mixed-case or lowercase letters for other variables.

Assigning values

To assign a value to a variable in the Bourne Again Shell, use the following syntax: **VARIABLE=value**. There can be no whitespace on either side of the equal sign (=). An example assignment follows: **\$ myvar=abc**. Under the TC Shell the assignment must be preceded by the word set and the SPACES on either side of the equal sign are optional: **\$ set myvar = abc**. The Bourne Again Shell permits you to put variable assignments on a command line. This type of assignment creates a variable that is local to the command shell—that is, the variable is accessible only from the program the command runs.

```
$ cat my_script
echo $TEMPDIR
$ TEMPDIR=/home/sam/temp ./my_script
/home/sam/temp
$ echo $TEMPDIR
$
```

The my_script shell script displays the value of TEMPDIR. The following command runs my_script with TEMPDIR set to /home/sam/temp. The echo builtin shows that the interactive shell has no value for TEMPDIR after running my_script. If TEMPDIR had been set in the interactive shell, running my_script in this manner would have had no effect on its value.

Parameter substitution

Because the echo builtin copies its arguments to standard output, you can use it to display the values of variables.

```
$ person=max
$ echo person
person
$ echo $person
max
```

The Linux and Shell Scripting

The second line of the example shows that person does not represent max. Instead, the string person is echoed as person. The shell substitutes the value of a variable only when you precede the name of the variable with a dollar sign (\$). Because of the leading \$, the shell recognizes that \$person is the name of a variable, substitutes the value of the variable, and passes that value to echo.

Quoting the \$

You can prevent the shell from substituting the value of a variable by quoting the leading \$. Double quotation marks do not prevent the substitution; single quotation marks or a backslash (\) do.

```
$ echo $person
max
$ echo "$person"
max
$ echo '$person'
$person
$ echo \$person
$person
```

Spaces

To assign a value that contains SPACES or TABs to a variable, use double quotation marks around the value.

```
$ person="max and zach"
$ echo $person
max and zach
$ person=max and zach
bash: and: command not found
```

If you do not quote the variable, the shell collapses each string of blank characters into a single SPACE before passing the variable to the utility:

```
$ person="max and zach"
$ echo $person
max and zach
$ echo "$person"
max and zach
```

Pathname expansion in assignments

When you execute a command with a variable as an argument, the shell replaces the name of the variable with the value of the variable and passes that value to the program being executed. If the value of the variable contains a special character, such as * or ?, the shell may expand that variable.

```
$ memo=max*
$ echo "$memo"
max*
```

The first line assigns the string max* to the variable memo. The Bourne Again Shell does not expand the string because bash does not perform pathname expansion when it assigns a value to a variable. All shells process a command line in a specific order. Within this order bash (but not tcsh)

Unit 08: The Bourne Again Shell and TC Shell

expands variables before it interprets commands. The echo command line, the double quotation marks quote the asterisk (*) in the expanded value of \$memo and prevent bash from performing pathname expansion on the expanded memo variable before passing its value to the echo command.

```
$ ls
max.report
max.summary
$ echo $memo
max.reportmax.summary
```

All shells interpret special characters as special when you reference a variable that contains an unquoted special character. The shell expands the value of the memo variable because it is not quoted. Here the shell expands the **\$memo variable to max***, expands **max*** to **max.report** and **max.summary**, and passes these two values to echo.

User-Created Variables - unset: Removes a Variable

Unless you remove a variable, it exists as long as the shell in which it was created exists. To remove the value of a variable but not the variable itself, assign a null value to the variable (use set person = in tcsh):

```
$ person=
$ echo $person
$
```

You can remove a variable using the unset builtin. The following command removes the variable person:

```
$ unset person
```

8.7 Variable Attributes**readonly: Makes the Value of a Variable Permanent**

You can use the readonlybuiltin (not in tcsh) to ensure that the value of a variable cannot be changed.

```
$ person=zach
$ echo $person
zach
$ readonly person
$ person=helen
bash: person: readonly variable
```

If you use the readonlybuiltin without an argument, it displays a list of all readonly shell variables. This list includes keyword variables that are automatically set as readonly as well as keyword or user-created variables that you have declared as readonly.

declare and typeset: Assign Attributes to Variables

The declare and typeset builtins (two names for the same command, neither of which is available in tcsh) set attributes and values for shell variables.

The Linux and Shell Scripting

Attribute	Meaning
-a	Declares a variable as an array
-f	Declares a variable to be a function name
-I	Declares a variable to be of type integer
-r	Makes a variable readonly
-x	Exports a variable, makes it global

\$ declare person1=max: The first line declares person1 and assigns it a value of max. This command has the same effect with or without the word declare.

\$ declare -r person2=zach: The readonly and export builtins are synonyms for the commands declare -r and declare -x, respectively.

\$ declare -rx person3=helen

\$ declare -x person4: You can declare a variable without assigning a value to it, as the preceding declaration of the variable person4 illustrates. This declaration makes person4 available to all subshells (i.e., makes it global). Until an assignment is made to the variable, it has a null value. You can list the options to declare separately in any order. The following is equivalent to the preceding declaration of person3:\$ declare -x -r person3=helen. Use the + character in place of - when you want to remove an attribute from a variable. You cannot remove the readonly attribute. After the following command is given, the variable person3 is no longer exported but it is still readonly.\$ declare +x person3. You can use typeset instead of declare.

Listing variable attributes

Without any arguments or options, declare lists all shell variables. The same list is output when you run set without any arguments.

```
$ declare -r
declare -ar BASH_VERSINFO='([0]="3" [1]="2" [2]="39" [3]="1" ... )'
declare -ir EUID="500"
declare -ir PPID="936"
declare -r SHELLOPTS="braceexpand:emacs:hashall:histexpand:history:..."
declare -ir UID="500"
declare -r person2="zach"
declare -rx person3="helen"
```

If you use a declare builtin with options but no variable names as arguments, the command lists all shell variables that have the indicated attributes set. For example, the command declare -r displays a list of all read only shell variables. This list is the same as that produced by the read only command without any arguments.

Integer

By default the values of variables are stored as strings. When you perform arithmetic on a string variable, the shell converts the variable into a number, manipulates it, and then converts it back to a string. A variable with the integer attribute is stored as an integer. Assign the integer attribute as follows:

```
$ declare -i COUNT
```

8.8 Keyword Variables

Keyword variables either are inherited or are declared and initialized by the shell when it starts. You can assign values to these variables from the command line or from a startup file. Typically, you want these variables to apply to all subshells you start as well as to your login shell. For those variables not automatically exported by the shell, you must use export or setenv to make them available to child shells.

HOME: Your Home Directory

By default, your home directory is the working directory when you log in. Your home directory is established when your account is set up; under Linux its name is stored in the **/etc/passwd** file.

```
$ pwd
/home/max/laptop
$ echo $HOME
/home/max
$ cd
$ pwd
/home/max
```

When you log in, the shell inherits the pathname of your home directory and assigns it to the variable HOME. When you give a cd command without an argument, cd makes the directory whose name is stored in HOME the working directory. This example shows the value of the HOME variable and the effect of the cd builtin. After you execute cd without an argument, the pathname of the working directory is the same as the value of HOME: your home directory.

Tilde

The shell uses the value of HOME to expand pathnames that use the shorthand tilde (~) notation to denote a user's home directory. It uses ls to list the files in Max's laptop directory, which is a subdirectory of his home directory:

```
$ echo ~
/home/max
$ ls ~/laptop
tester count lineup
```

PATH: Where the Shell Looks for Programs

If the file with the pathname you specified does not exist, the shell reports command not found. If the file exists as specified but you do not have execute permission for it, or in the case of a shell script you do not have read and execute permission for it, the shell reports Permission denied. If you give a simple filename as a command, the shell searches through certain directories (your search path) for the program you want to execute. It looks in several directories for a file that has the same name as the command and that you have execute permission for (a compiled program) or read and execute permission for (a shell script). The PATH shell variable controls this search. The

The Linux and Shell Scripting

system directories include /bin and /usr/bin and other directories appropriate to the local system. The PATH variable specifies the directories in the order the shell should search them. Each directory must be separated from the next by a colon. The following command sets PATH so that a search for an executable file starts with the **/usr/local/bin** directory. If it does not find the file in this directory, the shell looks next in **/bin**, and then in **/usr/bin**. If the search fails in those directories, the shell looks in the **~/bin** directory, a subdirectory of the user's home directory. Finally, the shell looks in the working directory. The exporting PATH makes its value accessible to subshells:
\$ export PATH=/usr/local/bin:/bin:/usr/bin:~/bin: A null value in the string indicates the working directory.

MAIL: Where Your Mail Is Kept

The MAIL variable (mail under tcsh) contains the pathname of the file that holds your mail (your mailbox, usually **/var/mail/name**, where name is your username). If MAIL is set and MAILPATH (next) is not set, the shell informs you when mail arrives in the file specified by MAIL. In a graphical environment you can unset MAIL so the shell does not display mail reminders in a terminal emulator window. The MAIL variable and other mail-related shell variables do not do anything unless you have a local mail server.

- **MAILPATH:** The MAILPATH variable (not available under tcsh) contains a list of filenames separated by colons. If this variable is set, the shell informs you when any one of the files is modified.
- **MAILCHECK:** This variable (not available under tcsh) specifies how often, in seconds, the shell checks for new mail. The default is 60 seconds. If you set this variable to zero, the shell checks before each prompt.

PS1: User Prompt (Primary)

The default Bourne Again Shell prompt is a dollar sign (\$). When you run bash with root privileges, bash typically displays a pound sign (#) prompt. The PS1 variable holds the prompt string that the shell uses to let you know that it is waiting for a command. When you change the value of PS1 or prompt, you change the appearance of your prompt. You can customize the prompt displayed by PS1. For example, the assignment

```
$ PS1="[\u@\h \W \!]$ "
```

displays the following prompt:

```
[user@host directory event]$
```

where user is the username, host is the hostname up to the first period, directory is the basename of the working directory, and event is the event number of the current command.

Symbol	Display in Prompt
\\$	# is the user is running with root privileges; otherwise, \$
\w	Pathname of the working directory
\W	Basename of the working directory
\!	Current event (history) number

Unit 08: The Bourne Again Shell and TC Shell

\d	Date in Weekday Month Date format
\h	Machine hostname, without the domain
\H	Full machine hostname, including the domain
\u	Username of the current user
\@	Current time of day in 12-hour, AM/PM format
\T	Current time of day in 12-hour HH:MM:SS format
\A	Current time of day in 24-hour HH:MM format
\t	Current time of day in 24-hour HH:MM:SS format

PS2: User Prompt (Secondary)

The PS2 variable holds the secondary prompt. On the first line, an unclosed quoted string follows echo. The shell assumes the command is not finished and, on the second line, gives the default secondary prompt (>). This prompt indicates the shell is waiting for the user to continue the command line. The shell waits until it receives the quotation mark that closes the string. Only then does it execute the command:

```
$ echo "demonstration of prompt string
>2"
demonstration of prompt string
2
$ PS2="secondary prompt: "
$ echo "this demonstrates
secondary prompt: prompt string 2"
this demonstrates
prompt string 2
```

The second command changes the secondary prompt to secondary prompt: followed by a SPACE. A multiline echo demonstrates the new prompt.

PS3: Menu Prompt

The PS3 variable holds the menu prompt for the select control structure.

PS4: Debugging Prompt

The PS4 variable holds the bash debugging symbol.

Keyword Variables

Variable	Value
BASH_ENV	The pathname of the startup file for noninteractive shells
CDPATH	The cd search path
COLUMNS	The width of the display used by select
FCEDIT	The name of the editor that fc uses by default
HISTFILE	The pathname of the file that holds the history list
HISTFILESIZE	The maximum number of entries saved in HISTFILE
HISTSIZE	The maximum number of entries saved in the history list
HOME	The pathname of the user's home directory; used as the default argument for cd and in tilde expansion
IFS	Internal Field Separator; used for word splitting
INPUTRC	The pathname of the Readline startup file
LANG	The locale category when that category is not specifically set with an LC_* variable
LC_*	A group of variables that specify locale categories including LC_COLLATE , LC_CTYPE , LC_MESSAGES , and LC_NUMERIC ; use the locale builtin to display a complete list with values
LINES	The height of the display used by select
MAIL	The pathname of the file that holds a user's mail
MAILCHECK	How often, in seconds, bash checks for mail
MAILPATH	A colon-separated list of file pathnames that bash checks for mail in
PATH	A colon-separated list of directory pathnames that bash looks for commands

Unit 08: The Bourne Again Shell and TC Shell

PROMPT_COMMAND	A command that bash executes just before it displays the primary prompt
PS1	Prompt String 1; the primary prompt
PS2	Prompt String 2; the secondary prompt
PS3	The prompt issued by select
PS4	The bash debugging symbol
REPLY	Holds the line that read accepts also used by select

Special Characters

Characters	Use
NEWLINE	Initiates the execution of a command page
;	Separates commands
()	Groups commands for execution by a subshell or identifies a function
(())	Expands an arithmetic expression
&	Executes a command in the background
	Pipe
>	Redirects standard output
>>	Appends standard output
<	Redirects standard input
<<	Here document
*	Any string of zero or more characters in an ambiguous file reference
?	Any single character in an ambiguous file reference
\	Quotes the following character

The Linux and Shell Scripting

'	Quotes a string, preventing all substitution
"	Quotes a string, allowing only variable and command substitution
'...'	Performs command substitution
[]	Character class in an ambiguous file reference
\$	Reference a variable
.	Executes a command
#	Begins a comment
{ }	Surrounds the contents of a function
:	Returns true
&&	Boolean AND
	Boolean OR
!	Boolean NOT
\$()	Performs command substitution
[]	Evaluates an arithmetic expression

8.9 Processes

A process is the execution of a command by the Linux kernel. The shell that starts when you log in is a command, or a process, like any other. When you give the name of a Linux utility on the command line, you initiate a process. When you run a shell script, another shell process is started and additional processes are created for each command in the script. Depending on how you invoke the shell script, the script is run either by the current shell or, more typically, by a subshell (child) of the current shell. A process is not started when you run a shell builtin, such as cd.

Process Structure

Like the file structure, the process structure is hierarchical, with parents, children, and even a root. A parent process forks a child process, which in turn can fork other processes. The term fork indicates that, as with a fork in the road, one process turns into two. Initially the two forks are identical except that one is identified as the parent and one as the child. You can also use the term spawn; the words are interchangeable. The operating system routine, or system call, that creates a new process is named fork(). When Linux begins execution when a system is started, it starts init, a single process called a spontaneous process, with PID number 1. This process holds the same position in the process structure as the root directory does in the file structure: It is the ancestor of all processes the system and users work with. When a command-line system is in multiuser mode,

Unit 08: The Bourne Again Shell and TC Shell

init runs getty or mingetty processes, which display login: prompts on terminals. When a user responds to the prompt and presses RETURN, getty hands control over to a utility named login, which checks the username and password combination. After the user logs in, the login process becomes the user's shell process.

Process Identification

Linux assigns a unique PID number at the inception of each process. As long as a process exists, it keeps the same PID number. During one session the same process is always executing the login shell. When you fork a new process—for example, when you use an editor—the PID number of the new (child) process is different from that of its parent process. When you return to the login shell, it is still being executed by the same process and has the same PID number as when you logged in.

- `$ sleep 10 &`

[1] 22789

- `$ ps -f`

UID PID PPID C STIME TTY TIME CMD

max 21341 21340 0 10:42 pts/16 00:00:00 bash

max 22789 21341 0 17:30 pts/16 00:00:00 sleep 10

max 22790 21341 0 17:30 pts/16 00:00:00 ps -f

The following example shows that the process running the shell forked (is the parent of) the process running ps. When you call it with the -f option, ps displays a full listing of information about each process. The line of the ps display with bash in the CMD column refers to the process running the shell. The column headed by PID identifies the PID number. The column headed PPID identifies the PID number of the parent of the process. From the PID and PPID columns you can see that the process running the shell (PID 21341) is the parent of the process running sleep (PID 22789). The parent PID number of sleep is the same as the PID number of the shell (21341). A second pair of sleep and ps -f commands shows that the shell is still being run by the same process but that it forked another process to run sleep:

- `$ sleep 10 &`

[1] 22791

- `$ ps -f`

UID PID PPID C STIME TTY TIME CMD

max 21341 21340 0 10:42 pts/16 00:00:00 bash

max 22791 21341 0 17:31 pts/16 00:00:00 sleep 10

max 22792 21341 0 17:31 pts/16 00:00:00 ps -f

You can also use pstree (or ps --forest, with or without the -e option) to see the parent-child relationship of processes

```
$ pstree -p
init(1)---acpid(1395)
|-atd(1758)
|-crond(1702)

+-kdeinit(2223)---firefox(8914)---run-mozilla.sh(8920)---firefox-bin(8925)
|   |-gaim(2306)
|   |-gqviev(14062)
|   |-kdeinit(2228)
|   |-kdeinit(2294)
|   |-kdeinit(2314)-+bash(2329)---ssh(2561)
|       |-bash(2339)
|           |-bash(15821)---bash(16778)
|   |-kdeinit(16448)
|   |-kdeinit(20888)
|   |-oclock(2317)
|       |-pam-panel-icon(2305)---pam_timestamp_c(2307)

+-login(1823)---bash(20986)---pstree(21028)
|   '-sleep(21026)

...
```

The line that starts with -kdeinit shows a graphical user running many processes, including firefox, gaim, and o'clock. The line that starts with -login shows a textual user running sleep in the background and running pstree in the foreground.

Executing a Command

When you give the shell a command, it usually forks [spawns using the fork() system call] a child process to execute the command. While the child process is executing the command, the parent process sleeps [implemented as the sleep() system call]. While a process is sleeping, it does not use any computer time; it remains inactive, waiting to wake up. When the child process finishes executing the command, it tells its parent of its success or failure via its exit status and then dies. The parent process (which is running the shell) wakes up and prompts for another command. When you run a process in the background by ending a command with an ampersand (&), the shell forks a child process without going to sleep and without waiting for the child process to run to completion. The parent process, which is executing the shell, reports the job number and PID number of the child process and prompts for another command. The child process runs in the background, independent of its parent. Although the shell forks a process to run most of the commands you give it, some commands are built into the shell. The shell does not need to fork a process to run builtins.

Variables Within a given process, such as your login shell or a subshell, you can declare, initialize, read, and change variables. By default, however, a variable is local to a process. When a process forks a child process, the parent does not pass the value of a variable to the child. You can make the value of a variable available to child processes (global) by using the export builtin under bash or the setenv builtin under tcsh.

8.10 History

The history mechanism, a feature adapted from the C Shell, maintains a list of recently issued command lines, also called events. There are various features of this:

- It provides a quick way to reexecute any of the events in the list.
- Execute variations of previous commands and reuses arguments from them.
- Replicates complicated commands and arguments that you used earlier in this login session or in a previous one.
- Enter a series of commands that differ from one another in minor ways.
- Serves as a record of what you have done.
- Keeps a record of a procedure that involved a series of commands.

Variables That Control History

The TC Shell's history mechanism is similar to bash's but uses different variables and has other differences.

Variable	Default	Function
HISTSIZE	500 events	Maximum number of events saved during a session
HISTFILE	~/.bash_history	Location of the history file
HISTFILESIZE	500 events	Maximum number of events saved between session

- **HISTSIZE:** The value of the HISTSIZE variable determines the number of events preserved in the history list during a session. A value in the range of 100 to 1,000 is normal.
- **HISTFILE:** When you exit from the shell, the most recently executed commands are saved in the file whose name is stored in the HISTFILE variable (the default is `~/.bash_history`). The next time you start the shell, this file initializes the history list.
- **HISTFILESIZE:** The value of the HISTFILESIZE variable determines the number of lines of history saved in HISTFILE.

The Bourne Again Shell assigns a sequential event number to each command line. You can display this event number as part of the bash prompt by including `\!` in PS1. Give the following command manually, or place it in `~/.bash_profile` to affect future sessions, to establish a history list of the 100 most recent events: `$ HISTSIZE=100`. The following command causes bash to save the 100 most recent events across login sessions: `$ HISTFILESIZE=100`. After you set HISTFILESIZE, you can log out and log in again, and the 100 most recent events from the previous login session will appear in your history list. Give the command `history` to display the events in the history list. This list is ordered so that the oldest events appear at the top. A tcsh history list includes the time the command was executed. The following history list includes a command to modify the bash prompt so it displays the history event number. The last event in the history list is the history command that displayed the list.

```

32 $ history | tail
23 PS1="!\ bash$ "
24 ls -l
25 cat temp
26 rm temp
27 vim memo
28 lpr memo
29 vim memo
30 lpr memo
31 rm memo
32 history | tail

```

As you run commands and your history list becomes longer, it may run off the top of the screen when you use the `history` builtin. Pipe the output of `history` through `less` to browse through it, or give the command `history 10` or `history | tail` to look at the ten most recent commands.

History alias

Creating the following aliases makes working with history easier. The first allows you to give the command h to display the ten most recent events. The second alias causes the command hg string to display all events in the history list that contain string. Put these aliases in your `~/.bashrc` file to make them available each time you log in.

- `$ alias 'h=history | tail'`
- `$ alias 'hg=history | grep'`

Reexecuting and Editing Commands

You can reexecute any event in the history list. This feature can save you time, effort, and aggravation. Not having to reenter long command lines allows you to reexecute events more easily, quickly, and accurately than you could if you had to retype the command line in its entirety. You can recall, modify, and reexecute previously executed events in three ways:

- `fc builtin,`
- `exclamation point commands ,`
- `Readline Library`

fc: Displays, Edits, and Reexecutes Commands

The fc (fix command) builtin (not in tcsh) enables you to display the history list and to edit and reexecute previous commands. It provides many of the same capabilities as the command-line editors. When you call fc with the -l option, it displays commands from the history list. Without any arguments, fc -l lists the 16 most recent commands in a numbered list, with the oldest appearing first:

```
$ fc -l
1024 cd
1025 view calendar
1026 vim letter.adams01
1027 aspell -c letter.adams01
1028 vim letter.adams01
1029 lpr letter.adams01
1030 cd ../memos
1031 ls
1032 rm *0405
1033 fc -l
1034 cd
1035 whereisaspell
1036 man aspell
1037 cd /usr/share/doc/*aspell*
1038 pwd
1039 ls
1040man-htm
```

The fc builtin can take zero, one, or two arguments with the -l option. The arguments specify the part of the history list to be displayed: `fc -l [first [last]]`. The fc builtin lists commands beginning with the most recent event that matches first. The argument can be an event number, the first few characters of the command line, or a negative number, which is taken to be the nth previous command. Without last, fc displays events through the most recent. If you include last, fc displays

Unit 08: The Bourne Again Shell and TC Shell

commands from the most recent event that matches first through the most recent event that matches last. The next command displays the history list from event 1030 through event 1035:

\$ **fc -l 1030 1035**

1030 cd .. /memos

1031 ls

1032 rm *0405

1033 fc -l

1034 cd

1035 whereisaspell

This command lists the most recent event that begins with view through the most recent command line that begins with whereis: \$ **fc -l view whereis**

1025 view calendar

1026 vim letter.adams01

1027 aspell -c letter.adams01

1028 vim letter.adams01

1029 lpr letter.adams01

1030 cd .. /memos

1031 ls

1032 rm *0405

1033 fc -l

1034 cd

1035 whereisaspell

To list a single command from the history list, use the same identifier for the first and second arguments. The following command lists event 1027: \$ **fc -l 1027 1027**

1027 ell -c letter.adams01

You can use fc to edit and reexecute previous commands: **fc [-e editor] [first [last]]**. When you call fc with the -e option followed by the name of an editor, fc calls the editor with event(s) in the Work buffer. By default, fc invokes the nano editor. Without first and last, it defaults to the most recent command: \$ **fc -e vi**. The fc builtin uses the stand-alone vim editor. If you set the FCEDIT variable, you do not need to use the -e option to specify an editor on the command line. Because the value of FCEDIT has been changed to /usr/bin/emacs and fc has no arguments, the following command edits the most recent command using the emacs editor:

\$ export FCEDIT=/usr/bin/emacs

\$ **fc**

If you call it with a single argument, fc invokes the editor on the specified command. This command starts the editor with event 1029 in the Work buffer. When you exit from the editor, the shell executes the command: \$ **fc 1029**. You can identify commands with numbers or by specifying the first few characters of the command name. This command calls the editor to work on events from the most recent event that begins with the letters vim through event 1030: \$ **fc vim 1030**. You can reexecute previous commands without using an editor. If you call fc with the -s option, it skips the editing phase and reexecutes the command. This command reexecutes event 1029: \$ **fc -s 1029**

lpr letter.adams01

This command reexecutes the previous command:

\$ **fc -s**

When you reexecute a command, you can tell fc to substitute one string for another. This command substitutes the string john for the string adams in event 1029 and executes the modified event:

\$ **fc -s adams=john 1029**

lpr letter.john01

Using an Exclamation Point (!) to Reference Events

The C Shell history mechanism uses an exclamation point to reference events. For example, the **!! command** reexecutes the previous event, and the shell replaces the \$ token with the last word on the previous command line. You can reference an event by using its absolute event number, its relative event number, or the text it contains. All references to events, called event designators, begin with an exclamation point (!). One or more characters follow the exclamation point to specify an event. You can put history events anywhere on a command line. To escape an exclamation point so that the shell interprets it literally instead of as the start of a history event, precede the exclamation point with a backslash (\) or enclose it within single quotation marks.

Event Designators

Designator	Meaning
!	Starts a history event unless followed immediately by SPACE, NEWLINE, = or (.
!!	The previous command
!n	Command number n in the history list
!-n	The nth preceding command
!string	The most recent command line that started with string
!?string[?]	The most recent command that contained string. The last ? Is optional
!#	The current command
!{event}	The event is an event designator. The braces isolate event from the surrounding text.

An event designator specifies a command in the history list.

You can reexecute the previous event by giving a !! command. In the following example, event 45 reexecutes event 44:

44 \$ ls -l text

-rw-rw-r-- 1 max group 45 Apr 30 14:53 text

45 \$!!

ls -l text

-rw-rw-r-- 1 max group 45 Apr 30 14:53 text

A number following an exclamation point refers to an event. If that event is in the history list, the shell executes it. Otherwise, the shell displays an error message. A negative number following an exclamation point references an event relative to the current event. For example, the command !-3 refers to the third preceding event. After you issue a command, the relative event number of a given event changes (event -3 becomes event -4). Both of the following commands reexecute event 44:

51 \$!44

ls -l text

-rw-rw-r-- 1 max group 45 Apr 30 14:53 text

52 \$!-8

ls -l text

-rw-rw-r-- 1 max group 45 Apr 30 14:53 text

When a string of text follows an exclamation point, the shell searches for and executes the most recent event that began with that string. If you enclose the string within question marks, the shell executes the most recent event that contained that string. The final question mark is optional if a RETURN would immediately follow it.

The Readline Library

Command-line editing under the Bourne Again Shell is implemented through the Readline Library, which is available to any application written in C. Any application that uses the Readline Library supports line editing that is consistent with that provided by bash. Programs that use the Readline Library, including bash, read `~/.inputrc` for key binding information and configuration settings. The **--noediting** command-line option turns off command-line editing in bash. You can choose one of two editing modes when using the Readline Library in bash: emacs or vi(m). Both modes provide many of the commands available in the standalone versions of the emacs and vim editors. You can also use the ARROW keys to move around. Up and down movements move you backward and forward through the history list. In addition, Readline provides several types of interactive word completion. The default mode is emacs; you can switch to vi mode with the following command: `$ set -o vi`. This command switches back to emacs mode: `$ set -o emacs`

Before you start, make sure the shell is in vi mode. When you enter bash commands while in vi editing mode, you are in Input mode. As you enter a command, if you discover an error before you press RETURN, you can press ESCAPE to switch to vim Command mode.

Unlike the vim editor, emacs is modeless. You need not switch between Command mode and Input mode because most emacs commands are control characters, allowing emacs to distinguish between input and commands. Like vim, the emacs command-line editor provides commands for moving the cursor on the command line and through the command history list and for modifying part or all of a command. However, in a few cases, the emacs command-line editor commands differ from those in the stand-alone emacs editor. In emacs you perform cursor movement by using both CONTROL and ESCAPE commands.

You can use the TAB key to complete words you are entering on the command line. This facility, called completion, works in both vi and emacs editing modes and is similar to the completion facility available in tcsh. Several types of completion are possible, and which one you use depends on which part of a command line you are typing when you press TAB.

Command Completion

If you are typing the name of a command (usually the first word on the command line), pressing TAB initiates command completion, in which bash looks for a command whose name starts with the part of the word you have typed. If no command starts with the characters you entered, bash beeps. If there is one such command, bash completes the command name. If there is more than one choice, bash does nothing in vi mode and beeps in emacs mode. Pressing TAB a second time causes bash to display a list of commands whose names start with the prefix you typed and allows you to continue typing the command name. In the following example, the user types bz and presses TAB. The shell beeps (the user is in emacs mode) to indicate that several commands start with the letters bz. The user enters another TAB to cause the shell to display a list of commands that start with bz followed by the command line as the user had entered it so far:

```
$ bz → TAB (beep) → TAB
bzcatbzdiff bzip2 bzless
bzcmpbzgrep bzip2recover bzmore
$ bz■
```

Next the user types c and presses TAB twice. The shell displays the two commands that start with bzc. The user types a followed by TAB. At this point the shell completes the command because only one command starts with bzca.

```
$ bzc → TAB (beep) → TAB
bzcatbzcmp
$ bzca → TAB → t ■
```

Pathname Completion

Pathname completion, which also uses TABs, allows you to type a portion of a pathname and have bash supply the rest. If the portion of the pathname you have typed is sufficient to determine a unique pathname, bash displays that pathname. If more than one pathname would match it, bash completes the pathname up to the point where there are choices so that you can type more. When you are entering a pathname, including a simple filename, and press TAB, the shell beeps (if the shell is in emacs mode—in vi mode there is no beep). It then extends the command line as far as it can.

```
$ cat films/dar → TAB (beep) cat films/dark_
```

In the films directory every file that starts with dar has k_ as the next characters, so bash cannot extend the line further without making a choice among files. The shell leaves the cursor just past the _ character. At this point you can continue typing the pathname or press TAB twice. In the latter case bash beeps, displays your choices, redisplays the command line, and again leaves the cursor just after the _ character.

```
$ cat films/dark_ → TAB (beep) → TAB
dark_passagedark_victory
$ cat films/dark_
```

When you add enough information to distinguish between the two possible files and press TAB, bash displays the unique pathname. If you enter p followed by TAB after the _ character, the shell completes the command line: \$ cat films/dark_p → TAB → assage. Because there is no further ambiguity, the shell appends a SPACE so you can finish typing the command line or just press RETURN to execute the command. If the complete pathname is that of a directory, bash appends a slash (/) in place of a SPACE.

Variable Completion

When you are typing a variable name, pressing TAB results in variable completion, wherein bash attempts to complete the name of the variable. In case of an ambiguity, pressing TAB twice displays a list of choices:

```
$ echo $HO → TAB → TAB
```

Unit 08: The Bourne Again Shell and TC Shell

\$HOME \$HOSTNAME \$HOSTTYPE

\$ echo \$HOM → TAB → E

.inputrc: Configuring the Readline Library

The Bourne Again Shell and other programs that use the Readline Library read the file specified by the INPUTRC environment variable to obtain initialization information. If INPUTRC is not set, these programs read the `~/inputrc` file. They ignore lines of `.inputrc` that are blank or that start with a pound sign (#). You can set variables in `.inputrc` to control the behavior of the Readline Library using the following syntax: `set variable value`.

Readline variables

Variable	Effect
editing-mode	Set to vi to start Readline in vi mode. Set to emacs to start Readline in emacs mode (the default). Similar to the <code>set -o vi</code> and <code>set -o emacs</code> shell commands
horizontal-scroll-mode	Set to on to cause long lines to extend off the right edge of the display area. Moving the cursor to the right when it is at the right edge of the display area shifts the line to the left so you can see more of the line. You can shift the line back by moving the cursor back past the left edge. The default value is off, which causes long lines to wrap onto multiple lines of the display.
mark-directories	Set to off to cause Readline not to place a slash (/) at the end of directory names it completes. The default value is on.
mark-modified-lines	Set to on to cause Readline to precede modified history lines with an asterisk. The default value is off.

8.11 Alias

An alias is a usually short name that the shell translates into another usually longer name or complex command. Aliases allow you to define new commands by substituting a string for the first token of a simple command. They are typically placed in the `~/bashrc` (bash) or `~/tcshrc` (tcsh) startup files so that they are available to interactive subshells. Under bash the syntax of the alias builtin is: `alias [name]=[value]`. Under tcsh the syntax is: `alias [name] [value]`.

In the bash syntax no SPACES are permitted around the equal sign. If value contains SPACES or TABs, you must enclose value within quotation marks. Unlike aliases under tcsh, a bash alias does not accept an argument from the command line in value. Use a bash function when you need to use an argument. An alias does not replace itself, which avoids the possibility of infinite recursion in handling an alias such as the following: `$ alias ls='ls -F'`. You can nest aliases. Aliases are disabled for noninteractive shells (that is, shell scripts). To see a list of the current aliases, give the command `alias`. To view the alias for a particular name, give the command `alias` followed by the name of the alias. You can use the `unalias` builtin to remove an alias. When you give an alias builtin command without any arguments, the shell displays a list of all defined aliases:

The Linux and Shell Scripting

```
$ alias
alias ll='ls -l'
alias l='ls -ltr'
alias ls='ls -F'
alias zap='rm -i'
```

Give an alias command to see which aliases are in effect. You can delete the aliases you do not want from the appropriate startup file.

Single Versus Double Quotation Marks in Aliases

The choice of single or double quotation marks is significant in the alias syntax when the alias includes variables. If you enclose value within double quotation marks, any variables that appear in value are expanded when the alias is created. If you enclose value within single quotation marks, variables are not expanded until the alias is used. The PWD keyword variable holds the pathname of the working directory (Shown in next slide). Max creates two aliases while he is working in his home directory. Because he uses double quotation marks when he creates the dirA alias, the shell substitutes the value of the working directory when he creates this alias. The alias dirA command displays the dirA alias and shows that the substitution has already taken place:

```
$ echo $PWD
/home/max
$ alias dirA="echo Working directory is $PWD"
$ alias dirA
alias dirA='echo Working directory is /home/max'
```

When Max creates the dirB alias, he uses single quotation marks, which prevent the shell from expanding the \$PWD variable. The alias dirB command shows that the dirB alias still holds the unexpanded \$PWD variable:

```
$ alias dirB='echo Working directory is $PWD'
$ alias dirB
alias dirB='echo Working directory is $PWD'
```

After creating the dirA and dirB aliases, Max uses cd to make cars his working directory and gives each of the aliases as commands. The alias he created using double quotation marks displays the name of the directory he created as the alias in as the working directory (which is wrong). In contrast, the dirB alias displays the proper name of the working directory.

```
$ cd cars
$ dirA
Working directory is /home/max
$ dirB
Working directory is /home/max/cars
```

Examples of Aliases

The following alias allows you to type r to repeat the previous command: \$ alias r='fc -s'. If you use the command ls -ltr frequently, you can create an alias that substitutes ls -ltr when you give the command l:

```
$ alias l='ls -ltr'
```

```
$1
total 41
-rw-r--r-- 1 max group 30015 Mar 1 2008 flute.ps
-rw-r----- 1 max group 3089 Feb 11 2009 XTerm.ad
-rw-r--r-- 1 max group 641 Apr 1 2009 fixtax.icn
-rw-r--r-- 1 max group 484 Apr 9 2009 maptax.icn
drwxrwxr-x 2 max group 1024 Aug 9 17:41 Tiger
drwxrwxr-x 2 max group 1024 Sep 10 11:32 testdir
-rwxr-xr-x 1 max group 485 Oct 21 08:03 floor
drwxrwxr-x 2 max group 1024 Oct 27 20:19 Test_Emacs
```

Another common use of aliases is to protect yourself from mistakes. The following example substitutes the interactive version of the rm utility when you give the command zap:

```
$ alias zap='rm -i'
$ zap f*
rm: remove 'fixtax.icn'? n
rm: remove 'flute.ps'? n
rm: remove 'floor'? n
```

The `-i` option causes rm to ask you to verify each file that would be deleted, thereby helping you avoid deleting the wrong file. You can also alias rm with the rm `-i` command: alias `rm='rm -i'`. This aliases cause the shell to substitute ls `-l` each time you give an ll command and ls `-F` each time you use ls:

```
$ alias ls='ls -F'
$ alias ll='ls -l'
```

```
$ ll
total 41
drwxrwxr-x 2 max group 1024 Oct 27 20:19 Test_Emacs/
drwxrwxr-x 2 max group 1024 Aug 9 17:41 Tiger/
-rw-r----- 1 max group 3089 Feb 11 2009 XTerm.ad
-rw-r--r-- 1 max group 641 Apr 1 2009 fixtax.icn
-rw-r--r-- 1 max group 30015 Mar 1 2008 flute.ps
-rwxr-xr-x 1 max group 485 Oct 21 08:03 floor*
-rw-r--r-- 1 max group 484 Apr 9 2009 maptax.icn
drwxrwxr-x 2 max group 1024 Sep 10 11:32 testdir/
```

The `-F` option causes ls to print a slash (/) at the end of directory names and an asterisk (*) at the end of the names of executable files. In this example, the string that replaces the alias ll (ls `-l`) itself contains an alias (ls). When it replaces an alias with its value, the shell looks at the first word of the replacement string to see whether it is an alias. In the preceding example, the replacement string contains the alias ls, so a second substitution occurs to produce the final command ls `-F -l`. When given a list of aliases without the =value or value field, the alias builtin responds by displaying the value of each defined alias. The alias builtin reports an error if an alias has not been defined:

The Linux and Shell Scripting

```
$ alias ll l lszapwx
alias ll='ls -l'
alias l='ls -ltr'
alias ls='ls -F'
alias zap='rm -i'
bash: alias: wx: not found
```

You can avoid alias substitution by preceding the aliased command with a backslash (\):

```
$ \ls
Test_Emacs XTerm.ad flute.ps maptax.icn
Tiger fixtax.icn floor testdir
```

Because the replacement of an alias name with the alias value does not change the rest of the command line, any arguments are still received by the command that gets executed:

```
$ ll f*
-rw-r--r-- 1 max group 641 Apr 1 2009 fixtax.icn
-rw-r--r-- 1 max group 30015 Mar 1 2008 flute.ps
-rwxr-xr-x 1 max group 485 Oct 21 08:03 floor*
```

You can remove an alias with the unalias builtin. When the zap alias is removed, it is no longer displayed with the alias builtin and its subsequent use results in an error message:

```
$ unalias zap
$ alias
alias ll='ls -l'
alias l='ls -ltr'
alias ls='ls -F'
$ zap maptax.icn
```

8.12 Functions

A bash shell function (tcsh does not have functions) is similar to a shell script in that it stores a series of commands for execution at a later time. However, because the shell stores a function in the computer's main memory (RAM) instead of in a file on the disk, the shell can access it more quickly than the shell can access a script. The shell also preprocesses (parses) a function so that it starts up more quickly than a script. Finally the shell executes a shell function in the same shell that called it. If you define too many functions, the overhead of starting a subshell can become unacceptable. You can declare a shell function in the `~/.bash_profile` startup file, in the script that uses it, or directly from the command line. You can remove functions with the `unset` builtin. The shell does not retain functions after you log out.

Removing variables and functions

If you have a shell variable and a function with the same name, using `unset` removes the shell variable. If you then use `unset` again with the same name, it removes the function. The syntax that declares a shell function is

```
[function] function-name ()
{
```

Unit 08: The Bourne Again Shell and TC Shell

*commands**}*

The word function is optional, function-name is the name you use to call the function, and commands comprise the list of commands the function executes when you call it. The commands can be anything you would include in a shell script, including calls to other functions. There are some features of functions:

- The opening brace ({}) can appear on the same line as the function name.
- Aliases and variables are expanded when a function is read, not when it is executed.
- You can use the break statement within a function to terminate its execution.
- Shell functions are useful as a shorthand as well as to define special commands.
- It creates a simple function that displays the date, a header, and a list of the people who are logged in on the system. This function runs the same commands as the whoson script.
- The greater than (>) signs are secondary shell prompts (PS2); do not enter them.

```
$ function whoson ()
> {
> date
> echo "Users Currently Logged On"
> who
> }
$ whoson
Sun Aug 9 15:44:58 PDT 2009
Users Currently Logged On
hls console Aug 8 08:59 (:0)
max pts/4 Aug 8 09:33 (0.0)
zach pts/7 Aug 8 09:23 (bravo.example.com)
```

Functions in startup files

If you want to have the whoson function always be available without having to enter it each time you log in, put its definition in `~/.bash_profile`. Then run `.bash_profile`, using the `.` (dot) command to put the changes into effect immediately:

```
$ cat ~/.bash_profile
export TERM=vt100
stty sane
whoson () {
date
echo "Users Currently Logged On"
who
}
$ . ~/.bash_profile
```

The Linux and Shell Scripting

You can specify arguments when you call a function. Within the function these arguments are available as positional parameters. The following example shows the arg1 function entered from the keyboard:

```
$ arg1 () {
> echo "$1"
>
$ arg1 first_arg
first_arg
```

8.13 Command Line Options

Two kinds of command-line options are available: short and long. Short options consist of a hyphen followed by a letter. Long options have two hyphens followed by multiple characters. Long options must appear before short options on a command line that calls bash.

Option	Explanation	Syntax
Help	Displays a usage message.	--help
No edit	Prevents users from using the Readline Library to edit command lines in an interactive shell.	--noediting
No profile	Prevents reading these startup files: <i>/etc/profile</i> , <i>~/.bash_profile</i> , <i>~/.bash_login</i> , and <i>~/.profile</i> .	--noprofile
No rc	Prevents reading the <i>~/.bashrc</i> startup file. This option is on by default if the shell is called as sh .	--norc
POSIX	Runs bash in POSIX mode.	--posix
Version	Displays bash version information and exits.	--version
Login	Causes bash to run as though it were a login shell.	-l (lowercase "l")
Shopt	Runs a shell with the opt shopt option . A -O (uppercase "O") sets the option; +O unsets it.	[±] 0 [opt]
End options of	On the command line, signals the end of options. Subsequent tokens are treated as arguments even if they begin with a hyphen (-).	--

8.14 Shell Features

You can control the behavior of the Bourne Again Shell by turning features on and off. Different features use different methods to turn features on and off. The `set` builtin controls one group of features, while the `shopt` builtin controls another group. You can also control many features from the command line you use to call bash.

set ±o: Turns Shell Features On and Off

The bash `set` builtin (there is a `set` builtin in tcsh, but it works differently), when used with the `-o` or `+o` option, enables, disables, and lists certain bash features. For example, the following command turns on the noclobber feature: `$ set -o noclobber`. You can turn this feature off (the default) by giving the command: `$ set +o noclobber`. The command `set -o` without an option lists each of the features controlled by `set`, followed by its state (on or off). The command `set +o` without an option lists the same features in a form you can use as input to the shell.

shopt: Turns Shell Features On and Off

The `shopt` (shell option) builtin (not available in tcsh) enables, disables, and lists certain bash features that control the behavior of the shell. For example, the following command causes bash to include filenames that begin with a period (.) when it expands ambiguous file references (the `-s` stands for `set`): `$ shopt -s dotglob`. You can turn this feature off (the default) by giving the following command (the `-u` stands for `unset`): `$ shopt -u dotglob`. The shell displays how a feature is set if you give the name of the feature as the only argument to `shopt`: `$ shopt dotglob`. The command `shopt` without any options or arguments lists the features controlled by `shopt` and their state. The command `shopt -s` without an argument lists the features controlled by `shopt` that are set or on. The command `shopt -u` lists the features that are unset or off.

bash features

Feature	Description	Syntax	Alternate Syntax
allexport	Automatically exports all variables and functions you create or modify after giving this command.	<code>set -o allexport</code>	<code>set -a</code>
braceexpand	Causes bash to perform brace expansion	<code>set braceexpand -o</code>	<code>set -B</code>
cdspell	Corrects minor spelling errors in directory names used as arguments to cd.	<code>shopt -s cdspell</code>	
cmdhist	Saves all lines of a multiline command in the same history entry, adding semicolons as needed.	<code>shopt -s cmdhist</code>	

dotglob	Causes shell special characters (wildcards) in an ambiguous file reference to match a leading period in a filename. By default, special characters do not match a leading period. You must always specify the filenames . and .. explicitly because no pattern ever matches them.	shopt -s dotglob	
emacs	Specifies emacs editing mode for command-line editing	set -o emacs	
errexit	Causes bash to exit when a simple command (not a control structure) fails.	set -o errexit	set -e
execfail	Causes a shell script to continue running when it cannot find the file that is given as an argument to exec. By default a script terminates when exec cannot find the file that is given as its argument.	shopt -s execfail	
expand_aliases	Causes aliases to be expanded (by default it is on for interactive shells and off for noninteractive shells).	shopt -s expand_aliases	
hashall	Causes bash to remember where commands it has found using PATH are located (default).	set -o hashall	set -h
histappend	Causes bash to append the history list to the file named by HISTFILE when the shell exits. By default bash overwrites this file.	shopt -s histappend	
histexpand	Turns on the history mechanism (which uses exclamation points by default). Turn this feature off to turn off history expansion.	set -o histexpand	set -H
history	Enables command history	set -o history	
huponexit	Specifies that bash send a SIGHUP signal to all jobs when an interactive login shell exits.	shopt -s huponexit	
ignoreeof	Specifies that bash must receive ten EOF characters before it exits. Useful on noisy dial-up lines.	set -o ignoreeof	

Unit 08: The Bourne Again Shell and TC Shell

monitor	Enables job control	set -o monitor	set -m
nocaseglob	Causes ambiguous file references to match filenames without regard to case (off by default).	shopt -s nocaseglob	
noclobber	Helps prevent overwriting files	set -o noclobber	set -C
noglob	Disables pathname expansion	set -o noglob	set -f
notify	With job control enabled, reports the termination status of background jobs immediately. The default behavior is to display the status just before the next prompt.	set -o notify	set -b
nounset	Displays an error and exits from a shell script when you use an unset variable in an interactive shell. The default is to display a null value for an unset variable.	set -o nounset	set -u
nullglob	Causes bash to expand ambiguous file References that do not match a filename to a null string. By default bash passes these file references without expanding them.	shopt -s nullglob	
posix	Runs bash in POSIX mode.	set -o posix	
verbose	Displays command lines as bash readsthem.	set -o verbose	set -v
vi	Specifies vi editing mode for commandline Editing	set -o vi	
xpg_echo	Causes the echo builtin to expand backslash escape sequences without the need for the -e option	shopt -s xpg_echo	
xtrace	Turns on shell debugging	set -o xtrace	set -x

8.15 The TC Shell

The TC Shell (tcsh) provides an interface between you and the Linux operating system. The TC Shell is an interactive command interpreter as well as a high-level programming language. You use only one shell at any given time. The TC Shell is an expanded version of the C Shell. The "T" in TC Shell comes from the TENEX and TOPS-20 operating systems. A number of features not found in

The Linux and Shell Scripting

csh are present in tcsh, including file and username completion, command-line editing, and spelling correction.

Assignment statement

The tcsh assignment statement has the following syntax:`set variable = value`. Having SPACES on either side of the equal sign, although illegal in bash, is allowed in tcsh. The default tcsh prompt is a greater than sign (`>`), but it is frequently set to a single \$ character followed by a SPACE.

Shell Scripts

The TC Shell can execute files containing tcsh commands. If the first character of a shell script is a pound sign (#) and the following character is not an exclamation point (!), the TC Shell executes the script under tcsh. If the first character is anything other than #, tcsh calls the sh link to dash or bash to execute the script. The tcsh echo builtin accepts either a -n option or a trailing \c to get rid of the RETURN that echo normally displays at the end of a line.

Checking shell

- **ps:** If you are not sure which shell you are using, use the ps utility to find out. It shows whether you are running tcsh, bash, sh (linked to bash), or possibly another shell.
- **finger:** The finger command followed by your username displays the name of your login shell, which is stored in the /etc/passwd file.

Entering the TC Shell

You can execute tcsh by giving the command `tcsh`. If you want to use tcsh as a matter of course, you can use the `chsh` (change shell) utility to change your login shell:

`bash $ chsh`

Changing shell for sam.

Password:

New shell [/bin/bash]: **/bin/tcsh**

Shell changed.

`bash $`

Leaving the tc shell

You can leave tcsh in several ways. The approach you choose depends on two factors: whether the shell variable `ignoreeof` is set and whether you are using the shell that you logged in on (your login shell) or another shell that you created after you logged in. If you are not sure how to exit from tcsh, press CONTROL-D on a line by itself with no leading SPACES, just as you would to terminate standard input to a program. You will either exit or receive instructions on how to exit. If you have not set `ignoreeof` and it has not been set for you in a startup file, you can exit from any shell by using CONTROL-D (the same procedure you use to exit from the Bourne Again Shell). When `ignoreeof` is set, CONTROL-D does not work. The `ignoreeof` variable causes the shell to display a message telling you how to exit. You can always exit from tcsh by giving an exit command. A logout command allows you to exit from your login shell only.

Startup Files

When you log in on the TC Shell, it automatically executes various startup files. You must have read access to a startup file to execute the commands in it. When you log in on the TC Shell, it automatically executes various startup files. You must have read access to a startup file to execute the commands in it.

1) /etc/csh.cshrc and /etc/csh.login:

The shell first executes the commands in /etc/csh.cshrc and /etc/csh.login. A user working with root privileges can set up these files to establish systemwide default characteristics for tcsh users. They contain systemwide configuration information, such as the default path, the location to check for mail, and so on.

2) .tcshrc and .cshrc:

Next the shell looks for ~/.tcshrc or, if it does not exist, ~/.cshrc. You can use these files to establish variables and parameters that are local to your shell. Each time you create a new shell, tcsh reinitializes these variables for the new shell.

3) .history

Login shells rebuild the history list from the contents of ~/.history. If the histfile variable exists, tcsh uses the file that histfile points to in place of .history.

4).login

Login shells read and execute the commands in ~/.login. This file contains commands that you want to execute once, at the beginning of each session.

5) /etc/csh.logout and .logout

The TC Shell runs the /etc/csh.logout and ~/.logout files, in that order, when you exit from a login shell.

Features Common to the Bourne Again and TC Shells

Most of the features common to both bash and tcsh are derived from the original C Shell:

- Command-line expansion (also called substitution)
- History
- Aliases
- Job control
- Filename substitution
- Directory stack manipulation
- Command substitution

Command-Line Expansion (Substitution)

The tcsh man page uses the term substitution instead of expansion; the latter is used by bash. The TC Shell scans each token for possible expansion in the following order:

The Linux and Shell Scripting

1. History substitution
2. Alias substitution
3. Variable substitution
4. Command substitution
5. Filename substitution
6. Directory stack substitution

1) History substitution:

The TC Shell assigns a sequential event number to each command line. You can display this event number as part of the tcsh prompt. As in bash, the `history` builtin displays the events in your history list. The list of events is ordered with the oldest events at the top. The last event in the history list is the `history` command that displayed the list. In the following history list, which is limited to ten lines by the argument of 10 to the `history` command, command 23 modifies the tcsh prompt to display the history event number. The time each command was executed appears to the right of the event number.

```
32 $ history 10
```

```
23 23:59 set prompt = "! $ "
24 23:59 ls -l
25 23:59 cat temp
26 0:00 rm temp
27 0:00 vim memo
28 0:00 lpr memo
29 0:00 vim memo
30 0:00 lpr memo
31 0:00 rm memo
32 0:00 history
```

The same event and word designators work in both shells. For example, `!!` refers to the previous event in tcsh, just as it does in bash. The command `!328` executes event number 328; `!?txt?` executes the most recent event containing the string `txt`.

- Few tcsh word modifiers not found in bash.

Modifier	Function
u	Converts the first lowercase letter into uppercase
l	Converts the first uppercase letter into lowercase
a	Applies the next modifier globally within a single word

Unit 08: The Bourne Again Shell and TC Shell

-
- Variables to control history

Variable	Default	Function
history	100 words	Maximum number of events saved during a session
histfile	<code>~/.history</code>	Location of the history file
savehist	not set	Maximum number of events saved between sessions

2) Aliases Substitution

The alias builtin has a slightly different syntax: alias name value. The following command creates an alias for ls: tcsh \$ alias ls "ls -lF". The tcsh alias allows you to substitute command-line arguments, whereas bash does not:

```
$ alias nam "echo Hello, \!^ is my name"
```

```
$ nam Sam
```

```
Hello, Sam is my name
```

The string `\!*` within an alias expands to all command-line arguments:\$ alias sortprint "sort \!* | lpr"

The next alias displays its second argument:\$ alias n2 "echo \!:2"

3) Job Control

Job control is similar in both bash and tcsh. You can move commands between the foreground and the background, suspend jobs temporarily, and get a list of the current jobs. The % character references a job when it is followed by a job number or a string prefix that uniquely identifies the job. You will see a minor difference when you run a multiple-process command line in the background from each shell. Whereas bash displays only the PID number of the last background process in each job, tcsh displays the numbers for all processes belonging to a job.

4) Filename Substitution

The TC Shell expands the characters *, ?, and [] in a pathname just as bash does. * matches any string of zero or more characters, ? matches any single character, [] defines a character class, which is used to match single characters appearing within a pair of brackets. The TC Shell expands command-line arguments that start with a tilde (~) into filenames in much the same way that bash does, with the ~ standing for the user's home directory or the home directory of the user whose name follows the tilde. The bash special expansions ~+ and ~- are not available in tcsh. Brace expansion is available in tcsh. Like tilde expansion, it is regarded as an aspect of filename substitution even though brace expansion can generate strings that are not the names of actual files.

5) Manipulating the Directory Stack

Directory stack manipulation in tcsh does not differ much from that in bash. The dirsbuiltin displays the contents of the stack, and the pushd and popdbuiltins push directories onto and pop directories off of the stack.

6) Command Substitution

The \${...} format for command substitution is not available in tcsh. In its place you must use the original '...' format. Otherwise, the implementation in bash and tcsh is identical.

Summary

- The Bourne Again Shell and TC Shell are command interpreters and high-level programming languages.
- Login shells are, by their nature, interactive.
- Pressing the suspend key (usually CONTROL-Z) immediately suspends (temporarily stops) the job in the foreground and displays a message that includes the word Stopped.
- Keyword shell variables have special meaning to the shell and usually have short, mnemonic names.
- Like the file structure, the process structure is hierarchical, with parents, children, and even a root. A parent process forks a child process, which in turn can fork other processes.
- You can declare a shell function in the `~/.bash_profile` startup file, in the script that uses it, or directly from the command line.
- You can remove functions with the `unset` builtin. The shell does not retain functions after you log out.
- The `shopt` (shell option) builtin (not available in tcsh) enables, disables, and lists certain bash features that control the behavior of the shell.
- The TC Shell is an interactive command interpreter as well as a high-level programming language.

Keywords

- **BASH_ENV:** Noninteractive shells look for the environment variable `BASH_ENV` (or `ENV` if the shell is called as `sh`) and execute commands in the file named by this variable.
- **File Descriptors:** A file descriptor is the place a program sends its output to and gets its input from.
- **Shell Script:** A shell script is a file that holds commands that the shell can execute. The commands in a shell script can be any commands you can enter in response to a shell prompt.
- **Positional Parameters:** Positional parameters enable you to access command-line arguments, a capability that you will often require when you write shell scripts.
- **MAIL:** The `MAIL` variable (`mail` under tcsh) contains the pathname of the file that holds your mail (your mailbox, usually `/var/mail/name`, where `name` is your username).
- **MAILPATH:** The `MAILPATH` variable (not available under tcsh) contains a list of filenames separated by colons. If this variable is set, the shell informs you when any one of the files is modified.
- **MAILCHECK:** This variable (not available under tcsh) specifies how often, in seconds, the shell checks for new mail. The default is 60 seconds. If you set this variable to zero, the shell checks before each prompt.
- **Process:** A process is the execution of a command by the Linux kernel.
- **Events:** The history mechanism, a feature adapted from the C Shell, maintains a list of recently issued command lines, also called events.

Unit 08: The Bourne Again Shell and TC Shell

- **HISTSIZE:** The value of the HISTSIZE variable determines the number of events preserved in the history list during a session. A value in the range of 100 to 1,000 is normal.
- **HISTFILE:** When you exit from the shell, the most recently executed commands are saved in the file whose name is stored in the HISTFILE variable (the default is `~/.bash_history`). The next time you start the shell, this file initializes the history list.
- **HISTFILESIZE:** The value of the HISTFILESIZE variable determines the number of lines of history saved in HISTFILE.
- **Alias:** An alias is a usually short name that the shell translates into another usually longer name or complex command.
- **ps:** If you are not sure which shell you are using, use the ps utility to find out. It shows whether you are running tcsh, bash, sh (linked to bash), or possibly another shell.
- **finger:** The finger command followed by your username displays the name of your login shell, which is stored in the `/etc/passwd` file.

Self Assessment

1. The shopt builtin _____ the features that control the bash.
 - A. Enables
 - B. Disables
 - C. Lists
 - D. All of the above mentioned

2. The short command line options consists of
 - A. Hyphen
 - B. A letter
 - C. Hyphen followed by a letter
 - D. None of the above

3. Using _____, we can recall, modify and re-execute previously executed events.
 - A. Fc builtin
 - B. Exclamation point commands
 - C. Readline libraries
 - D. All of the above mentioned

4. Event designators start with
 - A. &
 - B. !
 - C. @
 - D. #

5. Which key is used for pathname and command completion?
 - A. CTRL
 - B. TAB
 - C. RETURN

- D. SHIFT
6. Which of these variables gives the location of history file?
- A. HISTSIZE
 - B. HISTFILE
 - C. HISTFILESIZE
 - D. None of the above
7. Which builtin sets the attributes and values for shell variables?
- A. declare
 - B. typeset
 - C. Both declare and typeset
 - D. None of the above mentioned
8. PS4 is _____
- A. Primary prompt
 - B. Secondary prompt
 - C. Prompt issued by select
 - D. Bash debugging symbol
9. What is the naming convention for global variables?
- A. Only lowercase letters
 - B. Only uppercase letters
 - C. Mixed case letters
 - D. Only numbers
10. What is the correct syntax for assigning a value to a variable in Bourne Again Shell?
- A. VARIABLE=value
 - B. VARIABLE = value
 - C. VARIABLE= value
 - D. VARIABLE =value
11. Which keyword holds the pathname of the working directory?
- A. pwd
 - B. work
 - C. dir
 - D. key
12. You can specify arguments when you call a function. Within the function these arguments are available as _____
- A. Special parameters
 - B. Positional parameters
 - C. Uni parameters

Unit 08: The Bourne Again Shell and TC Shell

- D. None of the above
13. Which built in is used to make the value of a variable available to the child processes?
- A. echo
 - B. export
 - C. cat
 - D. avail
14. What is used to see the parent-child relationship?
- A. pstree
 - B. treeps
 - C. trpsee
 - D. eesptr
15. The directory stack implements _____ rule.
- A. FIFO
 - B. LIFO
 - C. RIRO
 - D. None of the above
16. To remove a directory from the stack, use the _____ builitin.
- A. popd
 - B. pushd
 - C. remd
 - D. None of the above
17. Which variable gives the maximum number of events saved between the session?
- A. history
 - B. histfile
 - C. savehist
 - D. None of the above
18. Which of these features are common in bash and tcsh?
- A. Aliases
 - B. Job control
 - C. Command substitution
 - D. All of the above mentioned

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. D | 2. C | 3. D | 4. B | 5. B |
| 6. B | 7. C | 8. D | 9. B | 10. A |
| 11. A | 12. B | 13. B | 14. A | 15. A |
| 16. A | 17. C | 18. D | | |

Review Questions

- 1) What are startup files? Explain in detail.
- 2) What are file descriptors? Explain.
- 3) How can we manipulate a directory stack? Explain the operations.
- 4) What are shell variables? Explain its types in details.
- 5) What are keyword variables? Explain with examples.
- 6) What is a process in Linux? Explain it.
- 7) What is history feature in Linux? Write its features. Which variables that control history?
- 8) How can we re-execute and edit commands? Write the different ways to do this.
- 9) What is an alias? Write its syntax. What is the use of single and double quotation marks in alias? Explain with example.
- 10) What are shell features? Write bash features.
- 11) What is TC shell? How can we enter and leave the TC shell? Write its startup files.
- 12) Write the features common to Bourne again shell and TC shell.



Further Readings

Mark G. Sobell, *A Practical Guide to Linux Commands, Editors, and Shell Programming*, Second Edition, Prentice Hall, Pearson Education, Inc.



Web Links

https://www.bottomupcs.com/file_descriptors.xhtml

<https://www.ibm.com/docs/en/aix/7.2?topic=concepts-shell-features>

Unit 09: Programming the Bourne Again Shell

CONTENTS

- Objectives
- Introduction
- 9.1 Control Structures
- 9.2 File Descriptor
- 9.3 Parameters and Variables
- 9.4 Builtin Commands
- 9.5 Expressions
- 9.6 Operators
- 9.7 Increment and Decrement
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to:

- Understand the control structures
- Understand the file descriptors
- Know the parameters and variables
- Understand the builtins
- Understand the expressions

Introduction

The programming languages are used for writing the programs. A simple program executes a sequence of statements without any jump or condition. In a programming language, we have various kinds of control structures which basically interrupts the flow of statements based upon conditions. The control flow commands alter the order of execution of commands within a shell script. It specifies the order in which computations are performed.

9.1 Control Structures

There are various control structures:

- if...then,
- for...in,
- while,
- until,
- break,

- Continue
- case statements

if...then

This control structure is used to express the decisions. The if...then control structure has the following syntax:

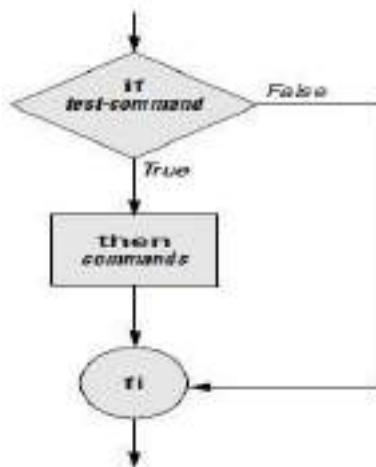
if
test-command

then

commands

fi

If the statement given in test_command turns out to be true, then the commands given must be executed. In this control structure, there is no command that needs to be printed when the test_command turns out to be false. The flow of if ... then control structure is shown below:



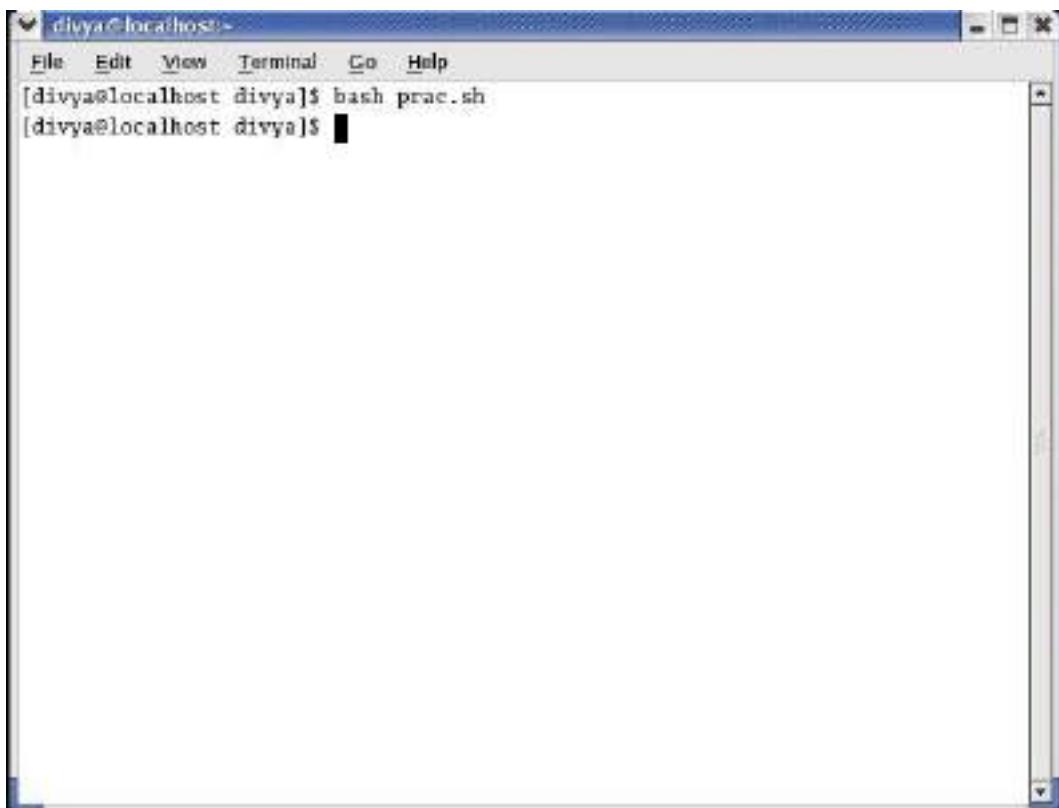
Here, one example is taken in which the value of a is 1 and b is 2 and the test condition is to check whether both values are equal or not. If the condition is true, then only the statement "Hi, LPU" will be printed. Otherwise, the control will exit. The control structure is ended using fi.

```

#!/bin/sh
a=1
b=2
if [ $a == $b ]
then
echo "Hi, LPU"
fi

```

The output of the program is " ". It will print nothing as the condition is false here. The control will just exit.

A screenshot of a terminal window titled "divya@localhost:~". The window has a menu bar with File, Edit, View, Terminal, Go, and Help. The main area shows the command "[divya@localhost divya]\$ bash prac.sh" followed by a blank line where the script's output would appear.

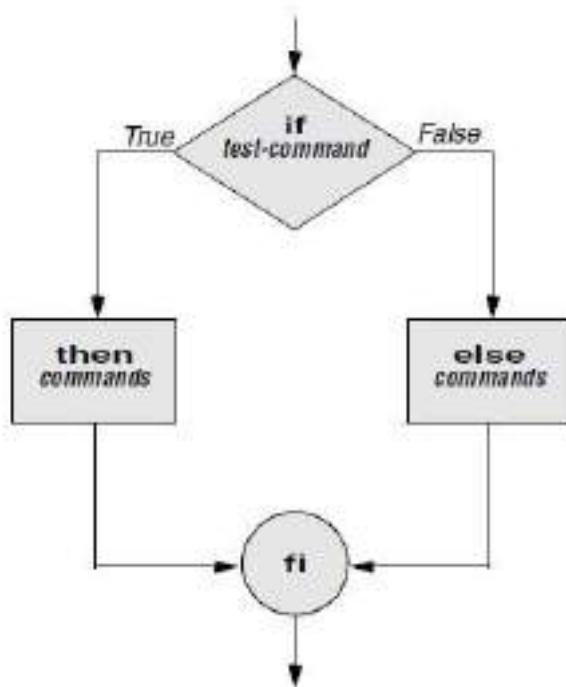
Task: Write a shell script to display square of a number

if...then...else

This control structure is also used to represent the decisions. Here the else part can be optional. The test-command will be evaluated. If this turns out to be true, then commands after that will be printed. If the test-command turns out to be false, then the statements given in else will be printed. The **if...then...else control structure** has the following syntax:

```
if test-command
then
commands
else
commands
fi
```

The flow of if... then... else is shown below:

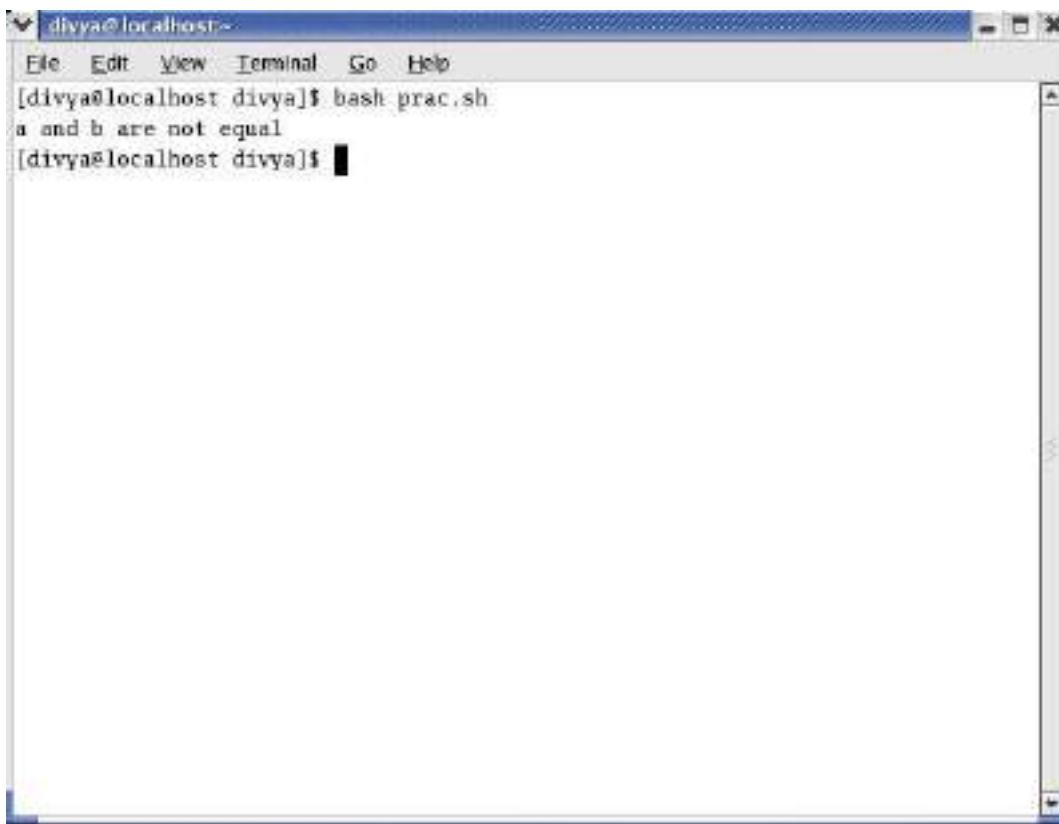


Here in this example, two variables are taken, i.e., a and b. If the values of a and b are equal, then the statement "a and b are equal" otherwise "a and b are not equal" is printed. The structure is ended with fi.

```

#!/bin/sh
a=1
b=2
if [ $a == $b ]
then
echo "a and b are equal"
else
echo "a and b are not equal"
fi
  
```

The output of the program is: The value of a is 1 and b is 2. It will print "a and b are not equal".



The screenshot shows a terminal window titled "divya@localhost:~". The menu bar includes "File", "Edit", "View", "Terminal", "Go", and "Help". The command entered is "bash prac.sh". The output displayed is "a and b are not equal".



Task: Write a shell script to check whether a person is eligible to vote or not.

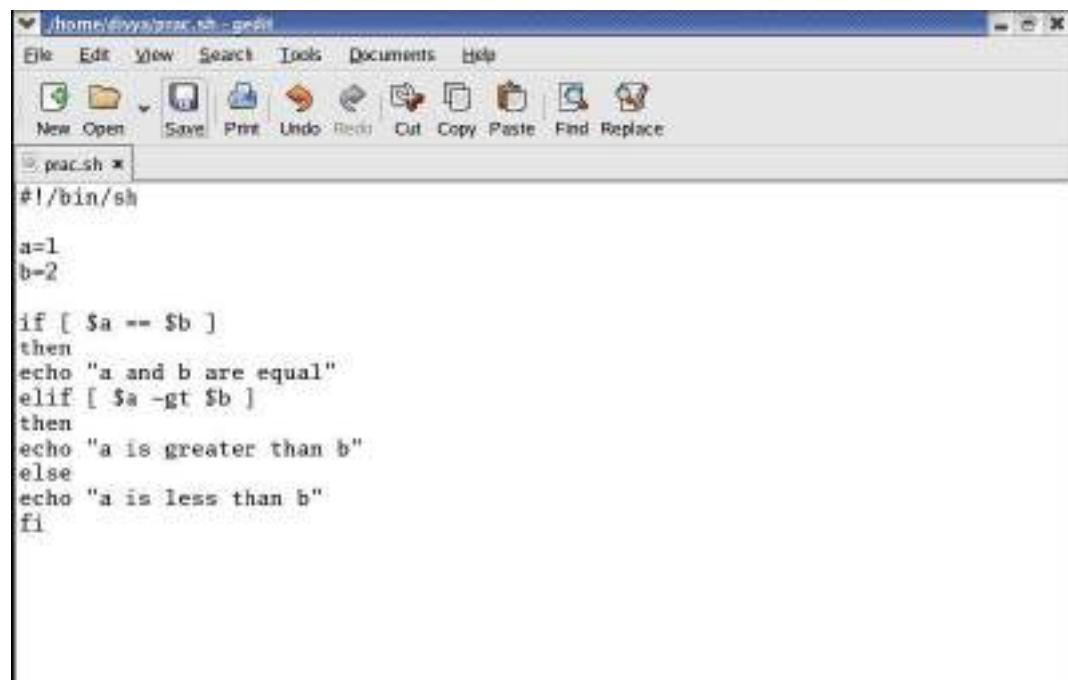
if...then...elif

This sequence of if statements is the most general way of writing a multi-way decision. The expressions are evaluated in order; if an expression is true, the statement associated with it is executed, and this terminates the whole chain. As always, the code for each statement is either a single statement, or a group of them in braces. The last else part handles the "none of the above" or default case where none of the other conditions is satisfied. The **if...then...elif control structure has the following syntax:**

```
if test-command  
then  
  commands  
elif test-command  
then  
  commands  
...  
else  
  commands  
fi
```

Linux and Shell Programming

The values of a and b are taken. The condition is to check the value of a and b. If both are equal, then it should print "a and b are equal", If not, it should check if a is greater than b, if yes, then it should print "a is greater than b". Otherwise "a is less than b".



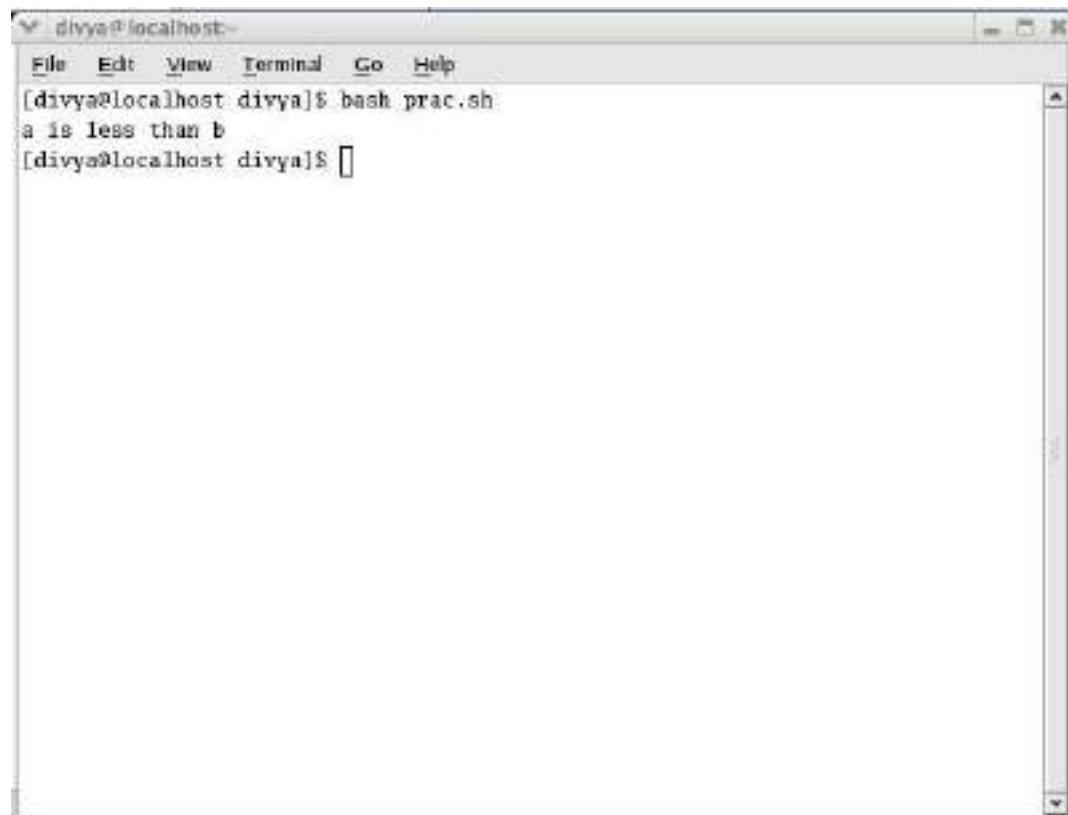
A screenshot of a text editor window titled 'prac.sh'. The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar contains icons for New, Open, Save, Print, Undo, Redo, Cut, Copy, Paste, Find, and Replace. The code in the editor is:

```
#!/bin/sh

a=1
b=2

if [ $a == $b ]
then
echo "a and b are equal"
elif [ $a -gt $b ]
then
echo "a is greater than b"
else
echo "a is less than b"
fi
```

The value of a is 1 and b is 2. So, the output of the program is "a is less than b".



A screenshot of a terminal window titled 'divya@localhost'. The menu bar includes File, Edit, View, Terminal, Go, and Help. The command entered is 'bash prac.sh'. The output is:

```
[divya@localhost divya]$ bash prac.sh
a is less than b
[divya@localhost divya]$
```

 Task: Write a shell script to display grades as per the following ranges of %age:

```
>= 80  'A+'
>=60  &&<80 'A'
>= 50  &&< 60 'B'
>= 40  &&<50 'C'
<40    'E'.
```

for...in

A loop is a sequence of instructions that is continually repeated until a certain condition is reached. It is a block of code that will repeat repeatedly. The for loop operates on lists of items. It repeats a set of commands for every item in a list. The for...in control structure *has the following syntax*:

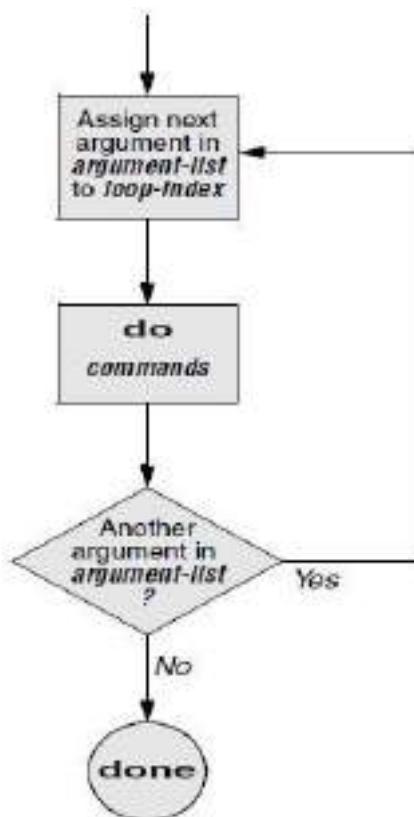
for loop-index in argument-list

do

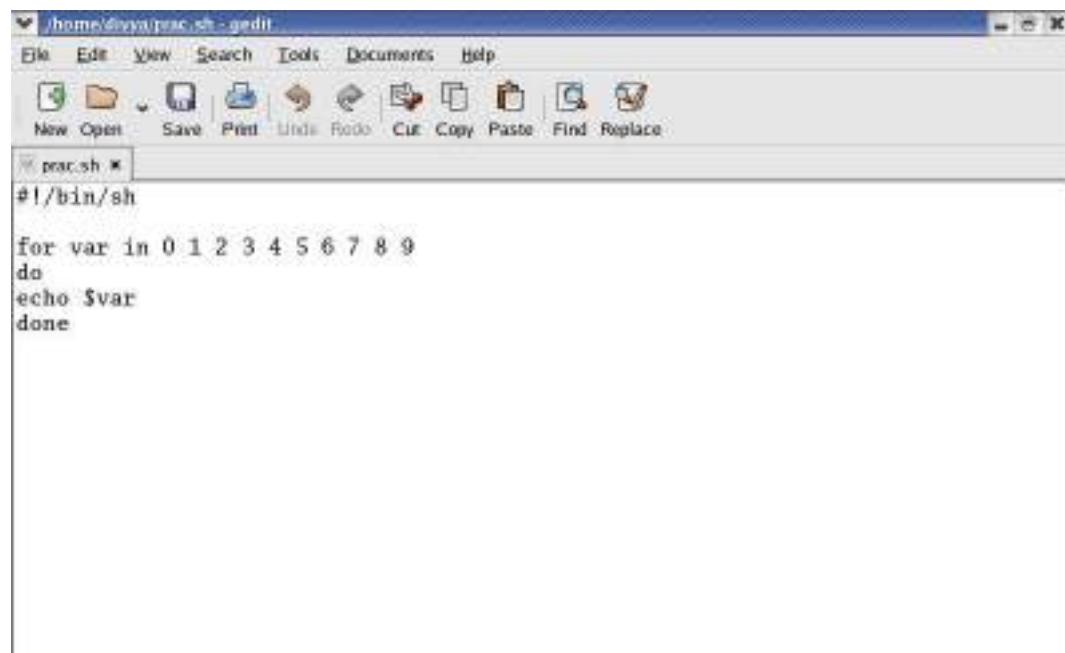
commands

done

Here loop index is the variable you specify in the do section and will contain the item in the loop that you are on. The list of arguments can be anything that returns a space or newline separated list.



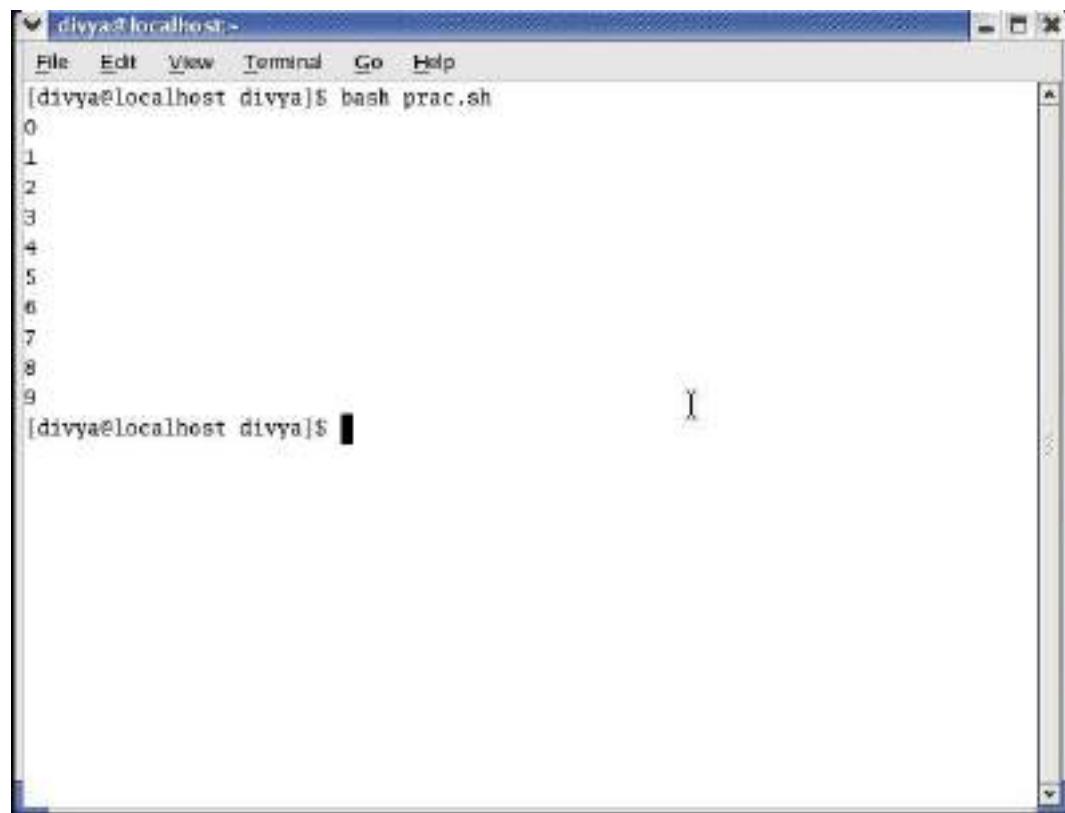
Linux and Shell Programming



A screenshot of a terminal window titled "divya@localhost:~". The window has a menu bar with File, Edit, View, Search, Tools, Documents, and Help. Below the menu is a toolbar with icons for New, Open, Save, Print, Undo, Redo, Cut, Copy, Paste, Find, and Replace. The main area shows the command "bash prac.sh" being run, followed by the output of the script which prints numbers from 0 to 9.

```
#!/bin/sh
for var in 0 1 2 3 4 5 6 7 8 9
do
echo $var
done
```

The output of the program is printing of numbers starting from 0 till 9.



A screenshot of a terminal window titled "divya@localhost:~". The window has a menu bar with File, Edit, View, Terminal, Go, and Help. Below the menu is a toolbar with icons for New, Open, Save, Print, Undo, Redo, Cut, Copy, Paste, Find, and Replace. The main area shows the command "bash prac.sh" being run, followed by the output of the script which prints numbers from 0 to 9.

```
[divya@localhost divya]$ bash prac.sh
0
1
2
3
4
5
6
7
8
9
[divya@localhost divya]$
```

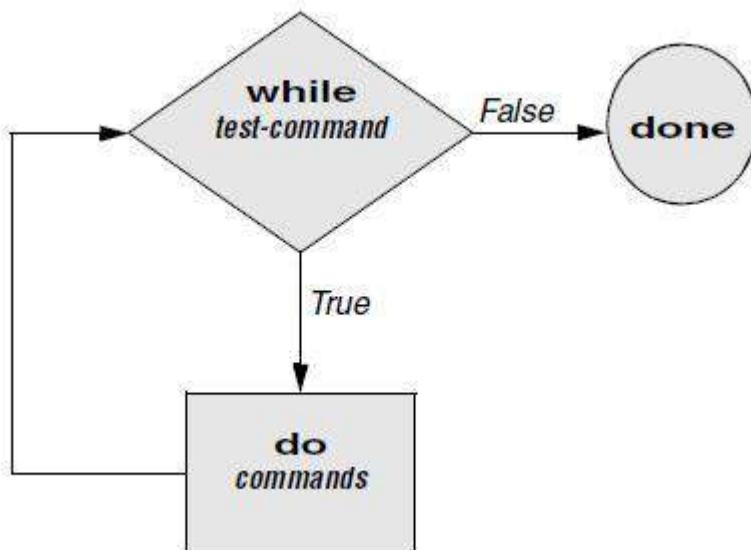


Task: Write a shell script to sum of all numbers starting from 1 to 100 using for loop

While

As long as the test-command (Figure 10-5) returns a true exit status, the while structure continues to execute the series of commands delimited by the do and done statements. Before each loop through the commands, the structure executes the test command. When the exit status of the test-command is false, the structure passes control to the statement after the done statement. The while control structure (not available in tcsh) has the following syntax:

```
while test-command
do
commands
done
```



The variable a is taken and initialized with 0. Till the value of a is less than 10, it keeps on printing and incrementing the values.

A screenshot of a terminal window titled "gedit" showing a shell script named "proc.sh". The script contains the following code:

```

#!/bin/bash

a=0

while [ "$a" -lt 10 ]
do
echo -n "$a"
((a +=1))
done
echo
  
```

The output of the program is: The values starting from 0 till 9 will be printed.

```
[divya@localhost ~]$ bash prac.sh
0123456789
[divya@localhost ~]$
```

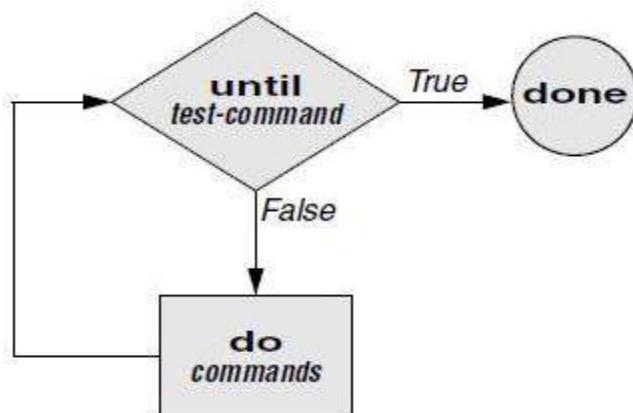


Task: Write a shell script to count odd numbers from 10 to 100.

Until

The until continues to loop until the test-command returns a true exit status. The while loop is perfect for a situation where you need to execute a set of commands while some condition is true. Sometimes you need to execute a set of commands until a condition is true. The syntax of until is:

```
until command
do
Statement(s) to be executed until command is true
done
```



Task: Write a shell script to take input for a number and keep on counting the chances how many times user has not entered a valid number. 'Valid number is -99'.

break and continue

You can interrupt a for, while, or until loop by using a break or continue statement. The break statement transfers control to the statement after the done statement, thereby terminating execution of the loop. The continue command transfers control to the done statement, continuing execution of the loop.

The screenshot shows a terminal window titled 'divya@localhost:~'. The window has a menu bar with File, Edit, View, Search, Tools, Documents, Help. Below the menu is a toolbar with icons for New, Open, Save, Print, Undo, Redo, Cut, Copy, Paste, Find, and Replace. The main area contains the following code:

```
#!/bin/bash

for index in 1 2 3 4 5 6 7 8 9 10
do
if [ $index -le 3 ]; then
echo "continue"
continue
fi
#
echo $index
#
if [ $index -ge 8 ]; then
echo "break"
break
fi
done
```

The screenshot shows a terminal window titled 'divya@localhost:~'. The window has a menu bar with File, Edit, View, Terminal, Go, Help. The command 'bash prac.sh' is entered and executed. The output shows the loop running from index 1 to 10, skipping indices 4 through 7 due to the 'continue' command, and then exiting the loop at index 8 due to the 'break' command.

```
[divya@localhost divya]$ bash prac.sh
continue
continue
continue
4
5
6
7
break
[divya@localhost divya]$
```

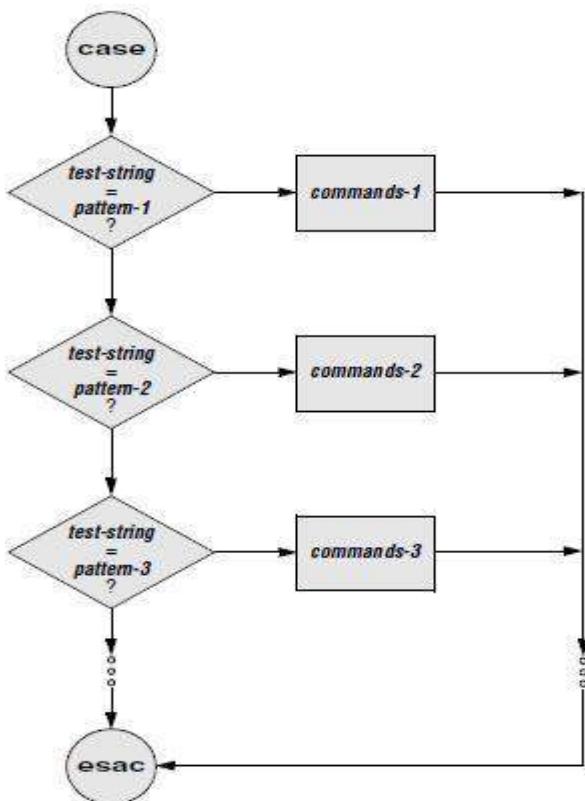


Task: Write a shell script to display square of all numbers from m to n when first multiple of 10 is reached loop should terminate.

Case

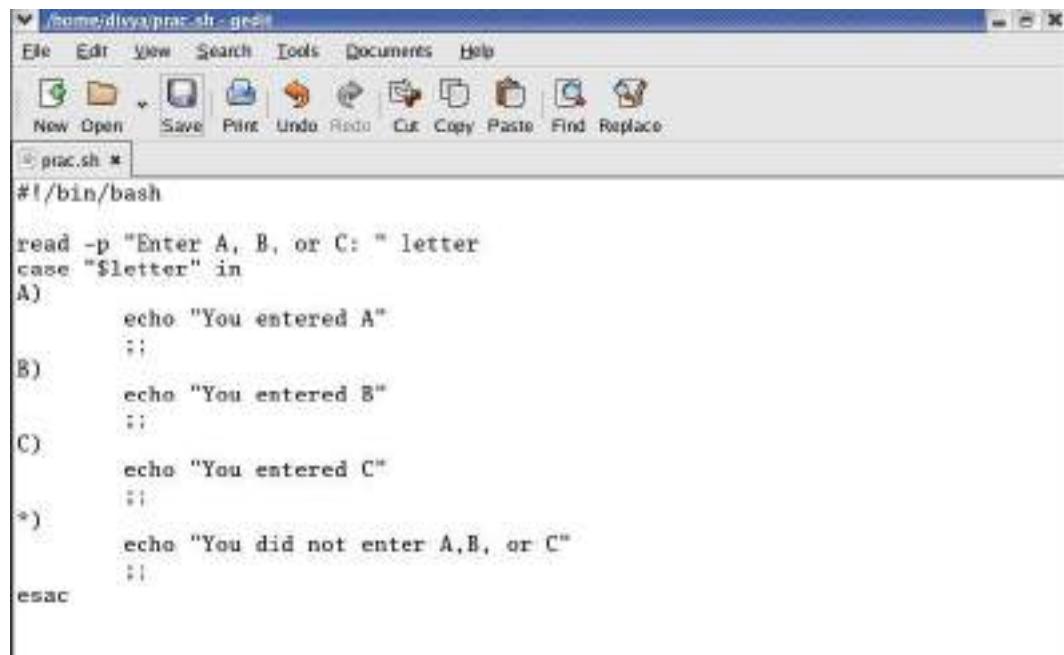
You can use multiple **if...elif** statements to perform a multi way branch. However, this is not always the best solution, especially when all the branches depend on the value of a single variable. It supports **case...esac** statement which handles exactly this situation, and it does so more efficiently than repeated **if...elif** statements. The case structure is a multiple-branch decision mechanism. The path taken through the structure depends on a match or lack of a match between the test-string and one of the patterns. The pattern in the case structure is analogous to an ambiguous file reference. It can be any special character like *, ?, [...] or | .The case control structure has the following syntax:

```
case test-string in
  pattern-1)
    commands-1
    ;;
  pattern-2)
    commands-2
    ;;
  pattern-3)
    commands-3
    ;;
  ...
esac
```



Unit 09: Programming the Bourne Again Shell

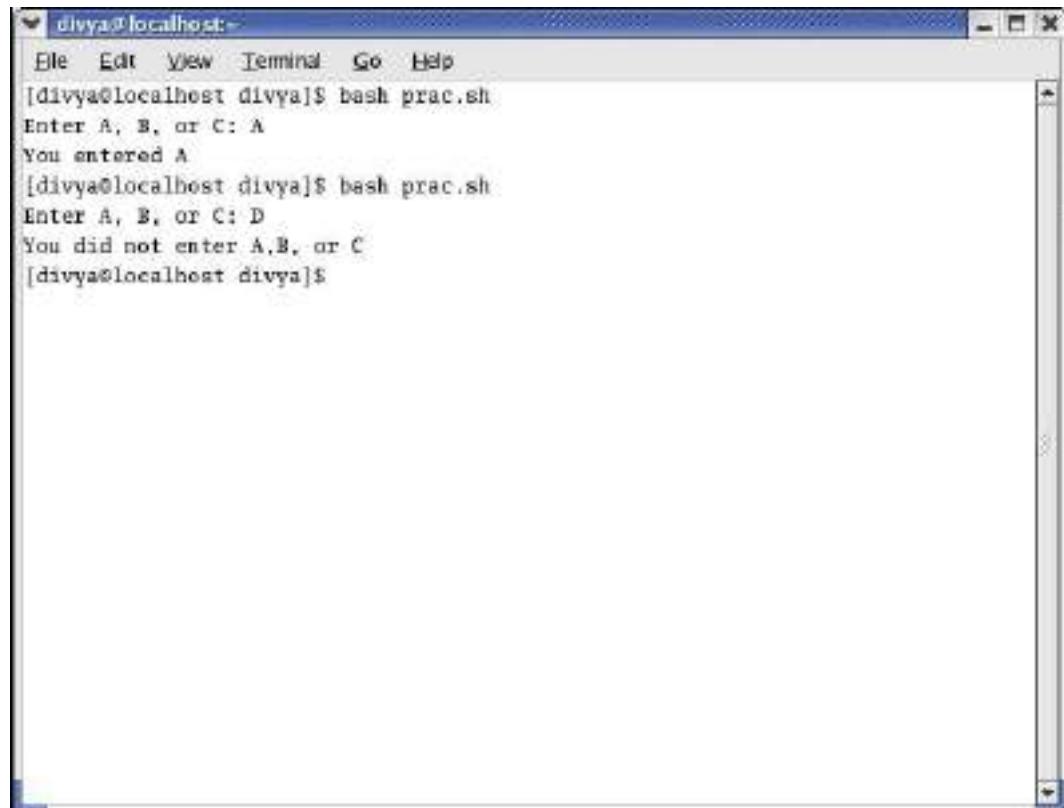
The next program asks to enter any character: A,B or C. If you have entered A, it prints "You entered A". If you have entered B, it prints " You entered B". If you have entered C, it prints " You entered C". If any other character is entered, it prints " You did not enter A, B, or C". The structure ends with esac.



```
#!/bin/bash

read -p "Enter A, B, or C: " letter
case "$letter" in
A)
    echo "You entered A"
;;
B)
    echo "You entered B"
;;
C)
    echo "You entered C"
;;
*)
    echo "You did not enter A,B, or C"
;;
esac
```

The output of the program is:



```
[divya@localhost divya]$ bash prac.sh
Enter A, B, or C: A
You entered A
[divya@localhost divya]$ bash prac.sh
Enter A, B, or C: D
You did not enter A,B, or C
[divya@localhost divya]$
```



Task: Write a shell script to implement arithmetic calculator.

Select

The select control structure is based on the one found in the Korn Shell. It displays a menu, assigns a value to a variable based on the user's choice of items, and executes a series of commands. The select control structure has the following syntax:

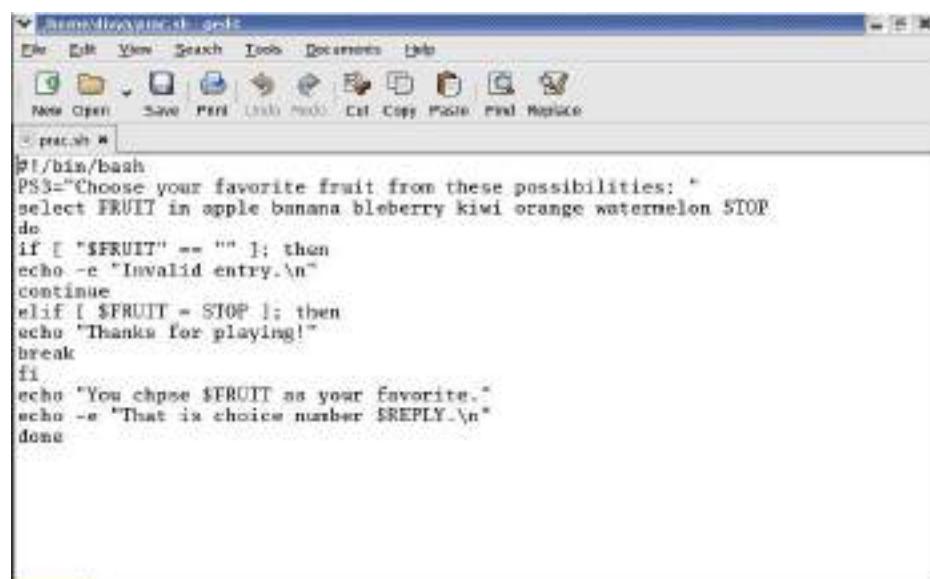
```
select varname [in arg . . . ]
```

```
do
```

```
commands
```

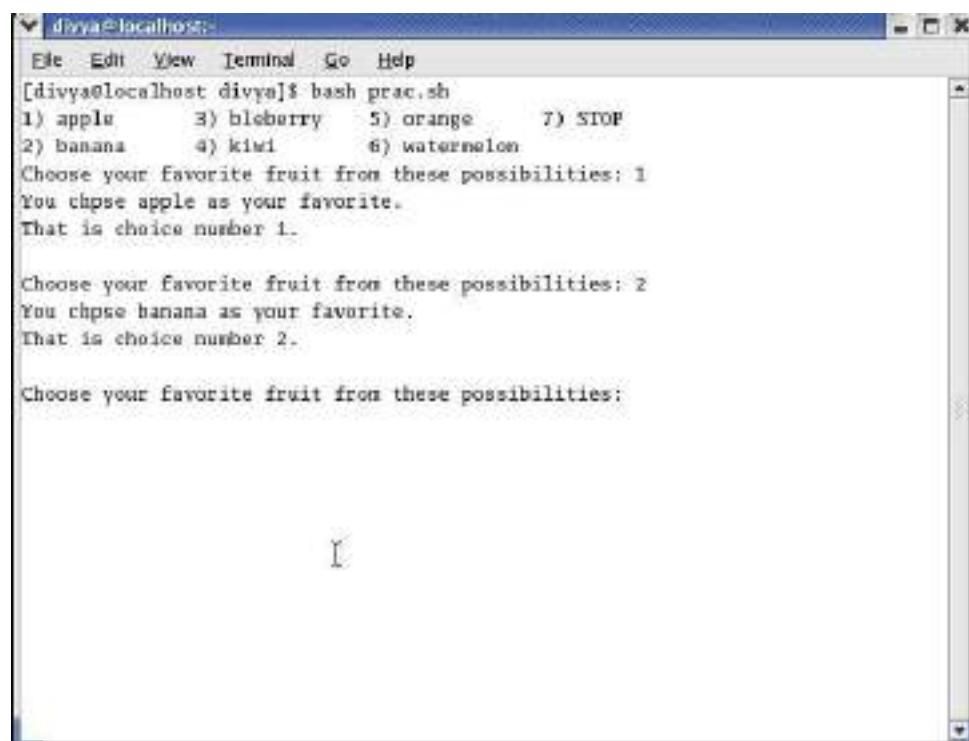
```
done
```

The select structure displays a menu of the arg items. If you omit the keyword in and the list of arguments, select uses the positional parameters in place of the arg items. The menu is formatted with numbers before each item.



```
prac.sh
#!/bin/bash
PS3="Choose your favorite fruit from these possibilities: "
select FRUIT in apple banana blueberry kiwi orange watermelon STOP
do
if [ "$FRUIT" == "" ]; then
echo -e "Invalid entry.\n"
continue
elif [ $FRUIT = STOP ]; then
echo "Thanks for playing!"
break
fi
echo "You chose $FRUIT as your favorite."
echo -e "That is choice number $REPLY.\n"
done
```

The output of the program is:



```
[divya@localhost divya]$ bash prac.sh
1) apple      3) blueberry   5) orange     7) STOP
2) banana     4) kiwi        6) watermelon
Choose your favorite fruit from these possibilities: 1
You chose apple as your favorite.
That is choice number 1.

Choose your favorite fruit from these possibilities: 2
You chose banana as your favorite.
That is choice number 2.

Choose your favorite fruit from these possibilities:
```

9.2 File Descriptor

Before a process can read from or write to a file, it must open that file. When a process opens a file, Linux associates a number (called a file descriptor) with the file. A file descriptor is an index into the process's table of open files. Each process has its own set of open files and its own file descriptors. After opening a file, a process reads from and writes to that file by referring to its file descriptor. When it no longer needs the file, the process closes the file, freeing the file descriptor. A typical Linux process starts with three open files: standard input (file descriptor 0), standard output (file descriptor 1), and standard error (file descriptor 2). Often these are the only files the process needs.

Opening a file descriptor

The Bourne Again Shell opens files using the exec built in as follows: exec n> outfile opens outfile for output and holds it open, associating it with file descriptor n. The exec m< infile opens infile for input and holds it open, associating it with file descriptor m.

Duplicating a file descriptor

The <& token duplicates an input file descriptor; >& duplicates an output file descriptor. You can duplicate a file descriptor by making it refer to the same file as another open file descriptor, such as standard input or output. To open or redirect file descriptor n as a duplicate of file descriptor m: exec n<&m

Once you have opened a file, you can use it for input and output in two ways. First, you can use I/O redirection on any command line, redirecting standard output to a file descriptor with >&n or redirecting standard input from a file descriptor with <&n. Second, you can use the read and echo builtins. If you invoke other commands, including functions, they inherit these open files and file descriptors. When you have finished using a file, you can close it using: exec n<&-

9.3 Parameters and Variables

There are various parameters and variables which are used. These are:

- Array Variables
- Locality of Variables
- Functions
- Special Parameters
- Positional Parameters

Array Variables

The Bourne Again Shell supports one-dimensional array variables. The subscripts are integers with zero-based indexing (i.e., the first element of the array has the subscript 0). The declaration and assignment of values to an array can be done as: name=(element1 element2 ...). An example of assigning four values to the array NAMES can be done as: \$ NAMES=(max helen sam zach). It references a single element of an array as follows: \$ echo \${NAMES[2]}

sam

The declare builtin with the -a option displays the values of the arrays (and reminds you that bash uses zero based indexing for arrays):

```
$ A=("${NAMES[*]}")
$ B=("${NAMES[@]}")
$ declare -a
declare -a A='([0]="max helen sam zach")'
declare -a B='([0]="max" [1]="helen" [2]="sam" [3]="zach")'
```

...
 declare -a NAMES=([0]="max" [1]="helen" [2]="sam" [3]="zach")'

Locality of Variables

By default, variables are local to the process in which they are declared. Thus, a shell script does not have access to variables declared in your login shell unless you explicitly make the variables available (global). Under bash, export makes a variable available to child processes. Under tcsh, setenv assigns a value to a variable and makes it available to child processes.

Locality of Variables(Without export)

```
$ cat extest1
cheese=american
echo "extest1 1: $cheese"
subtest
echo "extest1 2: $cheese"
$ cat subtest
echo "subtest 1: $cheese"
cheese=swiss
echo "subtest 2: $cheese"
$ ./extest1
extest1 1: american
subtest 1:
subtest 2: swiss
extest1 2: American
```

Locality of Variables(With export)

```
$ cat extest2
export cheese=american
echo "extest2 1: $cheese"
subtest
echo "extest2 2: $cheese"
$ ./extest2
extest2 1: american
subtest 1: american
subtest 2: swiss
extest2 2: American
```

An export builtin can optionally include an assignment:export cheese=American. The preceding statement is equivalent to the following two statements:

```
cheese=american
export cheese
```

An export builtin can optionally include an assignment:export cheese=american

The preceding statement is equivalent to the following two statements:

```
cheese=american
export cheese
```

Functions

Because functions run in the same environment as the shell that calls them, variables are implicitly shared by a shell and a function it calls.

```
$ function nam () {
>echo $myname
>myname=zach
>
$ myname=sam
$ nam
sam
$ echo $myname
zach
```

The myname variable is set to sam in the interactive shell. The nam function then displays the value of myname (sam) and sets myname to zach. The final echo shows that, in the interactive shell, the value of myname has been changed to zach. Local variables are helpful in a function written for general use. Because the function is called by many scripts that may be written by different programmers, you need to make sure the names of the variables used within the function do not conflict with (i.e., duplicate) the names of the variables in the programs that call the function. Local variables eliminate this problem. When used within a function, the typeset builtin declares a variable to be local to the function it is defined in.

Special parameters

Special parameters enable you to access useful values pertaining to command-line arguments and the execution of shell commands. You reference a shell special parameter by preceding a special character with a dollar sign (\$). As with positional parameters, it is not possible to modify the value of a special parameter by assignment.

\$\$: PID Number

The shell stores in the \$\$ parameter the PID number of the process that is executing it. In this example, echo displays the value of this variable and the ps utility confirms its value. Both commands show that the shell has a PID number of 5209:

```
$ echo $$
5209
$ ps
PID TTY TIME CMD
5209 pts/1 00:00:00 bash
6015 pts/1 00:00:00 ps
```

Because echo is built into the shell, the shell does not create another process when you give an echo command. However, the results are the same whether echo is a built in or not, because the shell

substitutes the value of \$\$ before it forks a new process to run a command. Incorporating a PID number in a filename is useful for creating unique file names when the meanings of the names do not matter; this technique is often used in shell scripts for creating names of temporary files. When two people are running the same shell script, having unique filenames keeps the users from inadvertently sharing the same temporary file.

\$?: Exit Status

When a process stops executing for any reason, it returns an exit status to its parent process. The exit status is also referred to as a condition code or a return code. The \$? (\$status under tcsh) variable stores the exit status of the last command. By convention a nonzero exit status represents a false value and means the command failed. A zero is true and indicates the command executed successfully. The first ls command succeeds and the second fails, as demonstrated by the exit status:

```
$ ls es
es
$ echo $?
0
$ ls xxx
ls: xxx: No such file or directory
$ echo $?
1
```

You can specify the exit status that a shell script returns by using the exit builtin, followed by a number, to terminate the script. If you do not use exit with a number to terminate a script, the exit status of the script is that of the last command the script ran.

```
$ cat es
echo This program returns an exit status of 7.
exit 7
$ es
This program returns an exit status of 7.
$ echo $?
7
$ echo $?
0
```

Positional Parameters

Positional parameters comprise the command name and command-line arguments. These parameters are called positional because within a shell script, you refer to them by their position on the command line. Only the set builtin allows you to change the values of positional parameters. However, you cannot change the value of the command name from within a script. The tcsh set builtin does not change the values of positional parameters.

\$#: Number of Command-Line Arguments

The \$# parameter holds the number of arguments on the command line (positional parameters), not counting the command itself:

```
$ cat num_args
echo "This script was called with $# arguments."
$ ./num_args sam max zach
This script was called with 3 arguments.
```

\$0: Name of the Calling Program

The shell stores the name of the command you used to call a program in parameter \$0. This parameter is numbered zero because it appears before the first argument on the command line:

```
$ cat abc
echo "The command used to run this script is $0"
$ ./abc
The command used to run this script is ./abc
$ ~sam/abc
The command used to run this script is /home/sam/abc
```

\$1-\$n: Command-Line Arguments

The first argument on the command line is represented by parameter \$1, the second argument by \$2, and so on up to \$n. For values of n greater than 9, the number must be enclosed within braces. For example, the twelfth command-line argument is represented by \${12}. The following script displays positional parameters that hold command-line arguments:

```
$ cat display_5args
echo First 5 arguments are $1 $2 $3 $4 $5
```

```
$ ./display_5args zach max helen
```

First 5 arguments are zach max helen

shift: Promotes Command-Line Arguments

The shift builtin promotes each command-line argument. The first argument (which was \$1) is discarded. The second argument (which was \$2) becomes the first argument (now \$1), the third becomes the second, and so on. Because no “unshift” command exists, you cannot bring back arguments that have been discarded. An optional argument to shift specifies the number of positions to shift (and the number of arguments to discard); the default is 1.

The following demo_shift script is called with three arguments. Double quotation marks around the arguments to echo preserve the spacing of the output. The program displays the arguments and shifts them repeatedly until no more arguments are left to shift:

```
$ cat demo_shift
echo "arg1= $1 arg2= $2 arg3= $3"
shift
$ ./demo_shift alice helen zach
arg1= alice arg2= helen arg3= zach
arg1= helen arg2= zach arg3=
arg1= zach arg2= arg3=
```

arg1= arg2= arg3=

set: Initializes Command-Line Arguments

When you call the set builtin with one or more arguments, it assigns the values of the arguments to the positional parameters, starting with \$1 (not available in tcsh). The following script uses set to assign values to the positional parameters \$1, \$2, and \$3:

```
$ cat set_it
set this is it
echo $3 $2 $1
$ ./set_it
it is this
```

\$* and \$@: Represent All Command-Line Arguments

The \$* parameter represents all command-line arguments, as the display_all program demonstrates:

```
$ cat display_all
echo All arguments are $*
```

```
$ ./display_all a b c d e f g h i j k l m n o p
All arguments are a b c d e f g h i j k l m n o p
```

9.4 Builtin Commands

Builtin commands do not fork a new process when you execute them.

Builtins	Funcexittions
:	Returns 0 or <i>true</i>
.	Executes a shell script as part of the current process
bg	Puts a suspended job in the background
break	Exits from a looping control structure
cd	Changes to another working directory
continue	Starts with the next iteration of a looping control structure

Unit 09: Programming the Bourne Again Shell

echo	Displays its arguments
eval	Scans and evaluates the command line
exec	Executes a shell script or program in place of the current process
export	Exits from the current shell
fg	Brings a job from the background into the foreground
getopts	Parses arguments to a shell script
jobs	Displays a list of background jobs
kill	Sends a signal to a process or job
pwd	Displays the name of the working directory
read	Reads a line from standard input
readonly	Declares a variable to be readonly
set	Sets shell flags or command-line argument variables; with no argument, lists all variables
shift	Promotes each command-line argument
test	Compares arguments
times	Displays total times for the current shell and its children
trap	Traps a signal
type	Displays how each argument would be interpreted as a command

umask	Returns the value of the file-creation mask
unset	Removes a variable or function
wait	Waits for a background process to terminate

9.5 Expressions

An expression comprises constants, variables, and operators that the shell can process to return a value. It contains arithmetic, logical and conditional expressions and operators.

Arithmetic Evaluation

The Bourne Again Shell can perform arithmetic assignments and evaluate many different types of arithmetic expressions, all using integers. The shell performs arithmetic assignments in a number of ways.

One is with arguments to the let builtin:\$ **let "VALUE=VALUE * 10 + NEW"**. Within a let statement you do not need to use dollar signs (\$) in front of variable names. Double quotation marks must enclose a single argument, or expression, that contains SPACEs. Because most expressions contain SPACEs and need to be quoted, bash accepts ((expression)) as a synonym for let "expression", obviating the need for both quotation marks and dollar signs: \$ ((VALUE=VALUE * 10 + NEW))

Logical expressions

You can use the ((expression)) syntax for logical expressions, although that task is frequently left to [[expression]].

```
$ cat age2
#!/bin/bash
echo -n "How old are you? "
read age
if ((30 < age && age < 60)); then
echo "Wow, in $((60-age)) years, you'll be 60!"
else
echo "You are too young or too old to play."
fi
$ ./age2
How old are you? 25
You are too young or too old to play.
```

Logical Evaluation (Conditional expressions)

The syntax of a conditional expression is [[expression]] where expression is a Boolean (logical) expression. You must precede a variable name with a dollar sign (\$) within expression. The result of executing this builtin, as with the test builtin, is a return status. The conditions allowed within the brackets are almost a superset of those accepted by test. Where the test builtin uses -a as a

Boolean AND operator, [[expression]] uses &&. Similarly, where test uses -o as a Boolean OR operator, [[expression]] uses ||.

String comparisons

The test builtin tests whether strings are equal. The [[expression]] syntax adds comparison tests for string operators. The > and < operators compare strings for order (for example, "aa" < "bbb"). The = operator tests for pattern match, not just equality: [[string = pattern]] is true if string matches pattern. This operator is not symmetrical; the pattern must appear on the right side of the equal sign. For example,

[[artist = a*]] is true (= 0), whereas [[a* = artist]] is false (= 1):

```
$ [[ artist = a* ]]
```

```
$ echo $?
```

```
0
```

```
$ [[ a* = artist ]]
```

```
$ echo $?
```

```
1
```

String Pattern Matching

The Bourne Again Shell provides string pattern-matching operators that can manipulate pathnames and other strings. These operators can delete from strings prefixes or suffixes that match patterns.

Operator	Function
#	Removes minimal matching prefixes
##	Removes maximal matching prefixes
%	Removes minimal matching suffixes
%%	Removes maximal matching suffixes

The syntax for these operators is: \${varname op pattern}.

where op is one of the operators and pattern is a match pattern like that used for filename generation.

9.6 Operators

Arithmetic expansion and arithmetic evaluation in bash use the same syntax, precedence, and associativity of expressions as in the C language. Within an expression you can use parentheses to change the order of evaluation.

Type of Operator	Function
Post	
	Var++ Postincrement Var-- Postdecrement
Pre	
	++var Preincrement --var Predecrement
Unary	
	+ Unary Plus - Unary Minus
Negation	
	! Boolean NOT ~ Complement
Exponentiation	
	** Exponent
Multiplication, Division, Remainder	
	* Multiplication
	/ Division
	% Remainder
Addition, Subtraction	
	+ Addition
	- Subtraction

Unit 09: Programming the Bourne Again Shell

Bitwise Shifts	
	<< Left bitwise shift
	>> Right bitwise shift
Comparison	
	<= Less than or equal to
	>= Greater than or equal to
	< less than
	> Greater than
Equality, inequality	
	== Equality
	!= Inequality
Bitwise	
	* Bitwise AND
	^ Bitwise XOR
	Bitwise OR
Boolean (Logical)	
	&& Boolean AND
	Boolean OR
Conditional Evaluation	
	? : Ternary Operator
Assignment	

	=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, = Assignment
Comma	
	, Comma

Pipe

The pipe token has higher precedence than operators. You can use pipes anywhere in a command that you can use simple commands. For example, the command line:

```
$ cmd1 | cmd2 || cmd3 | cmd4 && cmd5 | cmd6
```

is interpreted as if you had typed

```
$ ((cmd1 | cmd2) || (cmd3 | cmd4)) && (cmd5 | cmd6)
```

9.7 Increment and Decrement

The post increment, post decrement, pre increment, and pre decrement operators work with variables. The pre- operators, which appear in front of the variable name(as in `++COUNT` and `--VALUE`), first change the value of the variable (`++` adds 1;`--` subtracts 1) and then provide the result for use in the expression. The post- operators appear after the variable name (as in `COUNT++` and `VALUE--`); they first provide the unchanged value of the variable for use in the expression and then change the value of the variable.

Remainder

The remainder operator (%) yields the remainder when its first operand is divided by its second.

Boolean

The result of a Boolean operation is either 0 (false) or 1 (true). The `&&` (AND) and `||` (OR) Boolean operators are called short-circuiting operators. If the result of using one of these operators can be decided by looking only at the left operand, the right operand is not evaluated. The `&&` operator causes the shell to test the exit status of the command preceding it. If the command succeeded, bash executes the next command; otherwise, it skips the remaining commands on the command line. You can use this construct to execute commands conditionally.

Ternary

The ternary operator, `? :`, decides which of two expressions should be evaluated, based on the value returned by a third expression:**expression1 ? expression2 : expression3**

Assignment: The assignment operators, such as `+=`, are shorthand notations. For example, `N+=3` is the same as `((N=N+3))`.

Summary

- The while loop is perfect for a situation where you need to execute a set of commands while some condition is true.
- You can interrupt a for, while, or until loop by using a break or continue statement.

- The break statement transfers control to the statement after the done statement, thereby terminating execution of the loop.
- The continue command transfers control to the done statement, continuing execution of the loop.
- A file descriptor is an index into the process's table of open files.
- A typical Linux process starts with three open files: standard input (file descriptor 0), standard output (file descriptor 1), and standard error (file descriptor 2).
- The <& token duplicates an input file descriptor; >& duplicates an output file descriptor.
- The Bourne Again Shell supports one-dimensional array variables.
- By default, variables are local to the process in which they are declared.
- Special parameters enable you to access useful values pertaining to command-line arguments and the execution of shell commands.
- Positional parameters comprise the command name and command-line arguments. These parameters are called positional because within a shell script, you refer to them by their position on the command line.
- An expression comprises constants, variables, and operators that the shell can process to return a value.

Keywords

- **Break:** The break statement transfers control to the statement after the done statement, thereby terminating execution of the loop.
- **Continue:** The continue command transfers control to the done statement, continuing execution of the loop.
- **File Descriptor:** A file descriptor is an index into the process's table of open files.
- **exec n> outfile:** It opens outfile for output and holds it open, associating it with file descriptor n.
- **exec m< infile:** It opens infile for input and holds it open, associating it with file descriptor m.
- **exec n<&~:** When you have finished using a file, you can close it using:exec n<&~
- **export:** Under bash, export makes a variable available to child processes.
- **Setenv:** Under tcsh, setenv assigns a value to a variable and makes it available to child processes.
- **Pipe:** The pipe token has higher precedence than operators. You can use pipes anywhere in a command that you can use simple commands.

Self Assessment

1. Continue statement
 - A. Breaks loop and goes to next statement after loop
 - B. does not break loop but starts new iteration
 - C. exits the program
 - D. Starts from beginning of program

2. >& duplicates

- A. Input file descriptor
- B. Output file descriptor
- C. Error file descriptor
- D. None of the above

3. << represents

- A. Left bitwise shift
- B. Right bitwise shift
- C. Centre bitwise shift
- D. None of the above

4. Which of these is assignment operator?

- A. =
- B. *=
- C. /=
- D. All of the above

5. Which of these builtin removes a variable or function?

- A. set
- B. unset
- C. mask
- D. umask

6. A typical Linux process has

- A. File descriptor 0
- B. File descriptor 1
- C. File descriptor 2
- D. All of the above mentioned

7. Before a file can read/write to a file, it must _____ the file.

- A. Open
- B. Close
- C. Check
- D. None of the above

8. We can reference a shell special parameter by preceding a special character with a _____

- A. !
- B. @
- C. #
- D. \$

9. Which of these operators has the higher precedence?

- A. Pipe
- B. AND
- C. OR
- D. NOT

10. Instead of using if....else multiple times, we can use one _____.

- A. case....esac
- B. if....fi
- C. else.....else
- D. None of the above

11. Which of these control structures are available in Linux?

- A. If.....then
- B. For.....in
- C. While
- D. All of the above mentioned

12. The file descriptor is associated with _____

- A. Opening of file
- B. Reading from file
- C. Writing from file
- D. None of the above mentioned

13. The loops can be interrupted by using

- A. break
- B. continue
- C. Both break and continue
- D. None of the above

14. ^ represents

- A. Bitwise AND
- B. Bitwise OR
- C. Bitwise XOR
- D. None of the above

15. Which of these builtin removes a variable or function?

- A. set
- B. unset
- C. mask
- D. umask

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. B | 2. B | 3. A | 4. B | 5. B |
| 6. D | 7. D | 8. A | 9. A | 10. A |
| 11. D | 12. A | 13. C | 14. C | 15. B |

Review Questions

1. Write a shell script to use a switch statement to process a menu selection, using both upper- and lower-case options.
2. Write a shell script that displays the names of all directory files, but no other types of files, in the working directory.
3. What is a control structure? Explain different types of control structures with examples.
4. Explain the syntax of while and for...in loop with examples.
5. What is a file descriptor? How can we open and duplicate a file descriptor?
6. What are special and positional parameters in Linux?
7. What is a builtin? Give ten examples of builtin commands.



Further Readings

Mark G. Sobell, A Practical Guide to Linux Commands, Editors and Shell Scripting, Second Edition, Prentice Hall.



Web Links

[https://eng.libretexts.org/Bookshelves/Computer_Science/Operating_Systems/Linux_-_The_Penguin_Marches_On_\(McClanahan\)/13%3A_Working_with_Bash_Scripts/4.10%3A_Shell_Control_Statements](https://eng.libretexts.org/Bookshelves/Computer_Science/Operating_Systems/Linux_-_The_Penguin_Marches_On_(McClanahan)/13%3A_Working_with_Bash_Scripts/4.10%3A_Shell_Control_Statements)

Unit 10: Linux System Administration

CONTENTS

Objectives

Introduction

10.1 System Administrator and Superuser

10.2 System Administration Tools

10.3 Rescue Mode

10.4 Security Enhanced Linux

10.5 Run levels

10.6 Booting the System

10.7 System Administration Utilities

10.8 Standard Rules in Configuration Files

10.9 The xinetd Superserver

10.10 DHCP: Configures Hosts

10.11 Important files and directories in Linux

10.12 Types of files

10.13 Filesystems

10.14 mount: Mounts a Filesystem

10.15 Configuring User and Group Accounts

10.16 Backing Up Files

10.17 Scheduling Tasks

Summary:

Keywords

Self Assessment

Answer for Self Assessment

Review Questions

Further Readings

Objectives

After studying this unit, you will be able to

- Know about the system administrator and superuser
- Understand the rescue mode
- Understand the SELinux
- Understand the system operations and system administration utilities
- Understand how to set up a server
- know important files and directories in Linux
- Understand the file types and file systems
- Know how to back up files and schedule tasks
- Understand how to configure user and group accounts, system reports and parted

Introduction

A well-maintained system:

- Runs quickly enough so users do not get too frustrated waiting for the system to respond or complete a task.
- Has enough storage to accommodate users' reasonable needs.
- Provides a working environment appropriate to each user's abilities and requirements.
- Is secure from malicious and accidental acts altering its performance or compromising the security of the data it holds and exchanges with other systems.
- Is backed up regularly, with recently backed-up files readily available to users.
- Has recent copies of the software that users need to get their jobs done.
- Is easier to administer than a poorly maintained system.

10.1 System Administrator and Superuser

A system administrator should be available to help users with all types of system-related problems from logging in to obtaining and installing software updates to tracking down and fixing obscure network issues. Much of what a system administrator does is work that ordinary user do not have permission to do. When performing one of these tasks, the system administrator logs in as root to have system wide powers that are beyond those of ordinary users: A user with root privileges is referred to as **Super user**. The username is root by default.

Superuser has the following powers:

- Some commands, such as those that add new users, partition hard drives, and change system configuration, can be executed only by root.
- Superuser can use certain tools, such as sudo, to give specific users permission to perform tasks that are normally reserved for Superuser.
- Read, write, and execute file access and directory access permissions do not affect root: Superuser can read from, write to, and execute all files, as well as examine and work in all directories.
- Some restrictions and safeguards that are built into some commands do not apply to root. For example, root can change any user's password without knowing the old password.

When you are running with root (Superuser) privileges, the shell by convention displays a special prompt to remind you of your status. By default, this prompt is or ends with a pound sign (#).

Gain Superuser Privileges

You can gain or grant Superuser privileges in several ways:

- 1) When you bring the system up in single-user mode, you are Superuser.
- 2) Once the system is up and running in multiuser mode, you can log in as root. When you supply the proper password, you will be Superuser.
- 3) You can give an su (substitute user) command while you are logged in as yourself and, with the proper password, you will have Superuser privileges.
- 4) You can use sudo selectively to give users Superuser privileges for a limited amount of time on a per-user and per-command basis. The sudo utility is controlled by the /etc/sudoers file, which must be set up by root.
- 5) Any user can create a setuid (set user ID) file. Setuid programs run on behalf of the owner of the file and have all the access privileges that the owner has. While you are running as Superuser, you can change the permissions of a file owned by root to setuid. When an ordinary user executes a file that is owned by root and has setuid permissions, the program has full root privileges.

6) Some programs ask you for a password (either your password or the root password, depending on the command and the configuration of the system) when they start. When you provide the root password, the program runs with root privileges.

10.2 System Administration Tools

Many tools can help you be an efficient and thorough system administrator.

- su: Gives You Another User's Privileges
- console helper: Runs Programs as root
- kill: Sends a Signal to a Process

su: Gives You Another User's Privileges

The su (substitute user) utility can create a shell or execute a program with the identity and permissions of a specified user. Follow su on the command line with the name of a user; if you are working with root privileges or if you know the user's password, you take on the identity of that user. When you give an su command without an argument, su defaults to Superuser so that you take on the identity of root (you have to know the root password). It is better to use /bin/su which is the official version of su to avoid the trouble. When you give an su command to become Superuser, you spawn a new shell, which displays the # prompt. You return to your normal status (and your former shell and prompt) by terminating this shell: Press CONTROL-D or give an exit command.

Console helper: Runs Programs as root

The console helper utility can make it easier for someone who is logged in on the system console but not logged in as root to run system programs that normally can be run only by root.

kill: Sends a Signal to a Process

The kill built in sends a signal to a process. This signal may or may not terminate (kill) the process, depending on which signal is sent and how the process is designed.

10.3 Rescue Mode

Rescue mode is an environment you can use to fix a system that does not boot normally. To bring a system up in rescue mode, boot the system from the first installation CD, the Net Boot CD, or the install DVD. From the install DVD, select Rescue installed system from the Welcome menu. From the first installation CD and the Net Boot CD, enter the rescue (FEDORA) or boot rescue (RHEL) boot parameter. In rescue mode, you can change or replace configuration files, check, and repair partitions using fsck, rewrite boot information, and more. The rescue screen first asks if you want to set up the network interface. This interface is required if you want to copy files from other systems on the LAN or download files from the Internet.

Avoiding a Trojan Horse

A Trojan horse is a program that does something destructive or disruptive to a system while appearing to be benign. As an example, you could store the following script in an executable file named mkfs:

```
while true
do
echo 'Good Morning Mr. Jones. How are you? Ha Ha Ha.' > /dev/console
done
```

Linux and Shell Scripting

If you are running as Superuser when you run this command, it would continuously write a message to the console. If the programmer were malicious, it could do worse. The only thing missing in this plot is access permissions. A malicious user could implement this Trojan horse by changing Superuser's PATH variable to include a publicly writable directory at the start of the PATH string. A good way to help prevent the execution of a Trojan horse is to make sure that your PATH variable does not contain a single colon (:) at the beginning or end of the PATH string or a period (.) or double colon (::) anywhere in the PATH string. This precaution ensures that you will not execute a file in the working directory by accident.

10.4 Security Enhanced Linux

Traditional Linux security, i.e., DAC is based on users and groups. Because a process run by a user has access to anything the user has access to, fine-grained access control is difficult to achieve. SELinux was developed by the U.S. NSA, implements MAC in the Linux kernel. MAC enforces security policies that limit what a user or program can do. It defines a security policy that controls some or all objects, such as files, devices, sockets, and ports, and some or all subjects, such as processes. Using SELinux, you can grant a process only those permissions it needs to be functional, following the principle of least privilege. MAC is an important tool for limiting security threats that come from user errors, software flaws, and malicious users. The kernel checks MAC rules after it checks DAC rules.

States/Modes of SELinux

SELinux can be in one of three states (modes):

- **Enforcing** – This is the default state.
- **Permissive** – This is the diagnostic state.
- **Disabled** – No policy.

Policies of SELinux

SELinux implements one of the following policies:

- **Targeted** – Applies SELinux MAC controls only to certain (targeted) processes (default).
- **MLS** – Multilevel Security protection.
- **Strict** – Applies SELinux MAC controls to all processes (RHEL).

Turning off SELinux

There are two ways to disable SELinux: You can modify the /etc/selinux/config file so that it includes the line SELINUX=disabled and reboot the system, or you can use system-config-selinux.

config: The SELinux Configuration File

The /etc/selinux/config file, which has a link at /etc/sysconfig/selinux, controls the state of SELinux on the local system. Although you can modify this file, it may be more straightforward to work with system-config-selinux. In the following example, the policy is set to targeted, but that setting is of no consequence because SELinux is disabled:

```
$ cat /etc/selinux/config
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
# enforcing - SELinux security policy is enforced.
# permissive - SELinux prints warnings instead of enforcing.
```

```
# disabled - SELinux is fully disabled.  
SELINUX=disabled  
# SELINUXTYPE= type of policy in use. Possible values are:  
# targeted - Only targeted network daemons are protected.  
# strict - Full SELinux protection.  
SELINUXTYPE=targeted
```

To put SELinux in enforcing mode, change the line containing the SELINUX assignment to SELINUX=enforcing. Similarly, you can change the policy by setting SELINUXTYPE.

getenforce, setenforce, and sestatus: Work with SELinux

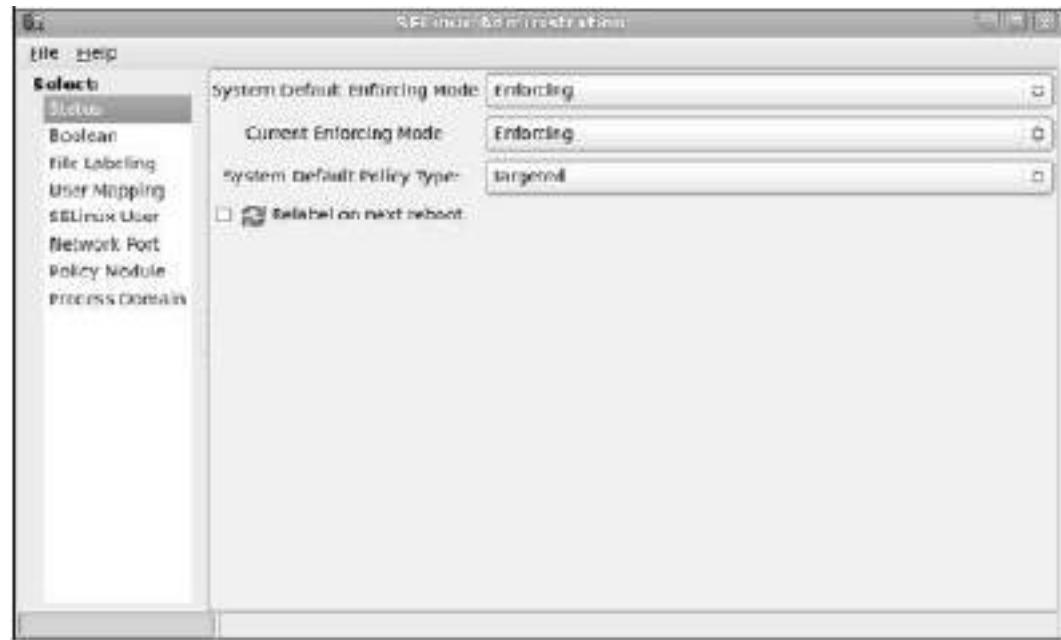
The getenforce and setenforce utilities report on and temporarily set the SELinux mode. The sestatus utility displays a summary of the state of SELinux:

```
# getenforce  
Enforcing  
# setenforce permissive  
# sestatus  
SELinux status: enabled  
SELinuxfs mount: /selinux  
Current mode: permissive  
Mode from config file: enforcing  
Policy version: 24  
Policy from config file: targeted
```

Setting the Targeted Policy with system-config-selinux

The system-config-selinux utility displays the SELinux Administration window, which controls SELinux. To run this utility, enter system-config-selinux from a command line in a graphical environment or select Main menu: System □ Administration □ SELinux Management.

SELinux Administration window



With Status highlighted on the left side of the SELinux Administration window, choose Enforcing (default), Permissive, or Disabled from the drop-down list labeled System Default Enforcing Mode. The mode you choose becomes effective next time you reboot the system. You can use the drop-down list labeled Current Enforcing Mode to change between Enforcing and Permissive modes immediately. When you change the mode using this list, the system resumes the default mode when you reboot it. To modify the SELinux policy, highlight Boolean on the left side of the SELinux Administration window and scroll through the list of modules. To find modules that pertain to NFS, type nfs in the text box labeled Filter and then press RETURN. The SELinux Administration window displays all modules with the string nfs in their descriptions. The modules with tick marks in the Active column are in use.

10.5 Run levels

Number	Name	Login	Level	Filesystems
0	Halt			
1	Single user	Textual	Down	Mounted
2	Multiuser without NFS	Textual	Up	Mounted
3	Multiuser	Textual	Up	Mounted
4	User Defined			
5	Multiuser with X	Graphical	Up	Mounted
6	Reboot			

- **Default Run level:** By default, Fedora systems boot to graphical multiuser mode (runlevel 5).
- **runlevel utility:** The runlevel utility displays the previous and current runlevels. This utility is a transitional tool; it provides compatibility with SysVinit. In the following example, the N indicates that the system does not know what the previous runlevel was and the 5 indicates that the system is in multiuser mode.

```
$ runlevel
```

```
N 5
```

- **telinit utility:** The telinit utility allows a user with root privileges to bring the system down, reboot the system, or change between recovery (single-user) and multiuser modes. The telinit utility is a transitional tool; it provides compatibility with SysVinit. The format of a telinit command is: telinit runlevel.
- **Recovery mode and the root password:** When the system enters recovery (single-user) mode, init requests the root password before displaying the root prompt. When the system enters graphical multiuser mode, it displays a graphical login screen.

10.6 Booting the System

Booting a system is the process of reading the Linux kernel into system memory and starting it running. As the last step of the boot procedure, Linux runs the init program as PID number 1. The init program is the first genuine process to run after booting and is the parent of all system processes. (That is why when you run as root and kill process 1, the system dies.)

- **initdefault:** The initdefault entry in the /etc/inittab file tells init which runlevel to bring the system to. Set initdefault to 3 to cause the system to present a text login message when it boots; set it to 5 to present a graphical login screen (default).
- **init daemon:** As the last step of the boot procedure, Linux starts the init daemon as PID number 1. The init daemon is the first genuine process to run after booting and is the parent of all system processes.

Init Scripts: Start and Stop System Services

The first script that runs is /etc/rc.d/rc.sysinit, which performs basic system configuration, including setting the system clock, hostname, and keyboard mapping; setting up swap partitions; checking the filesystems for errors; and turning on quota management.

service: Configures Services I

Fedora/RHEL provides service, a handy utility that can report on or change the status of any of the system services in /etc/rc.d/init.d.

system-config-services: Configures Services II

The system-config-services utility displays the Service Configuration window. This utility has two functions: It turns system services on and off immediately, and it controls which services are stopped and started when the system enters and leaves runlevels 2–5. The system-config-services utility works with many of the services listed in /etc/rc.d/init.d as well as with those controlled by xinetd and listed in /etc/xinetd.d (or specified in /etc/xinetd.conf). To run system-config-services, enter system-config-services from a command line in a graphical environment or select Main menu: System | Administration | Services.

chkconfig: Configures Services III

The chkconfig character-based utility duplicates much of what the system-config services utility does: It makes it easier for a system administrator to maintain the /etc/rc.d directory hierarchy.

This utility can add, remove, list startup information, and check the state of system services. It changes the configuration only—it does not change the current state of any service.

Single-User Mode

When the system is in single-user mode, only the system console is enabled. You can run programs from the console in single-user mode just as you would from any terminal in multiuser mode. The only difference is that few of the system daemons will be running. The scripts in /etc/rc.d/rc1.d are run as part of single-user initialization. With the system in single-user mode, you can perform system maintenance that requires file systems to be unmounted or that requires just a quiet system—no one except you using it, so that no user programs interfere with disk maintenance and backup programs.

Going to Multiuser Mode

After you have determined that all is well with the filesystems, you can bring the operating system up to multiuser mode. When you exit from the single-user shell, init brings the system to the default run level—usually 5. Alternatively, you can give the following command in response to the Superuser prompt to bring the system to textual multiuser mode (use 5 to go to graphical multiuser mode): # /sbin/telinit 3. When it goes from single-user to textual multiuser mode, the system executes the K (kill or stop) scripts and then the S (start) scripts in /etc/rc.d/rc3.d.

Graphical Multiuser Mode

Graphical multiuser mode is the default state for a Fedora/RHEL system. In this mode all appropriate filesystems are mounted, and users can log in from all connected terminals, dial-up lines, and network connections. All support services and daemons are enabled and running. Once the system is in graphical multiuser mode, a login screen appears on the console. Most systems are set up to boot directly to graphical multiuser mode without stopping at single-user mode.

Logging In

- Textual login
- Graphical login

Textual login

With a textual login, the system uses init, mingetty, and login to allow a user to log in; login uses PAM modules to authenticate users. The system is in multiuser mode, the Upstart init daemon is responsible for spawning a mingetty process on each of the lines that a user can use to log in.

Graphical login

With a graphical login, the Upstart init daemon starts gdm (the GNOME display manager) by default on the first free virtual terminal, providing features similar to those offered by mingetty and login. The gdm utility starts an X server and presents a login window. The gdm display manager then uses PAM to authenticate the user and runs the scripts in the /etc/gdm/PreSession directory.

Logging Out

When the system displays a shell prompt, you can either execute a program or exit from the shell. If you exit from the shell, the process running the shell dies and the parent process wakes up. When the shell is a child of another shell, the parent shell wakes up and displays a prompt. Exiting from a login shell causes the operating system to send Upstart a signal that one of its children has died. Upon receiving this signal, Upstart takes action based on the contents of the appropriate tty job definition file. In the case of a process controlling a line for a terminal, Upstart informs mingetty that the line is free for another user. When you are at runlevel 5 and exit from a GUI, the GNOME display manager, gdm, initiates a new login display.

Bringing the System Down

The shutdown and halt utilities perform the tasks needed to bring the system down safely. These utilities can restart the system, prepare the system to be turned off, put the system in single-user mode, and, on some hardware, power down the system. The poweroff and reboot utilities are linked to halt. If you call halt when the system is not shutting down (runlevel 0) or rebooting (runlevel 6), halt calls shutdown. CONTROL-ALT-DEL: Reboots the System

Crash

A crash occurs when the system stops suddenly or fails unexpectedly. A crash may result from software or hardware problems or from a loss of power. As a running system loses power, it may behave in erratic or unpredictable ways. In a fraction of a second, some components are supplied with enough voltage; others are not. Buffers are not flushed, corrupt data may be written to the hard disk, and so on. IDE drives do not behave as predictably as SCSI drives under these circumstances. After a crash, you must bring the operating system up carefully to minimize possible damage to the filesystems. On many occasions, little or no damage will have occurred.

10.7 System Administration Utilities

These utilities can help you perform system administration tasks.

Fedora/RHEL configuration tools

Most of the Fedora/RHEL configuration tools are named `system-config-*`. These tools bring up a graphical display when called from a GUI; some display a textual interface when called from a non-GUI command line. Some, such as `system-config-firewall-tui`, use a name with a `-tui` extension for the textual interface.

Tool	Function
<code>system-config-authentication</code>	Displays the Authentication Configuration window with three tabs. User Information tab: It allows you to enable NIS, LDAP, Hesiod, and Winbind support. Authentication tab: It allows you to work with Kerberos, LDAP, Smart Card, Fingerprint Reader, and Windbind. Options tab: It allows you to use shadow and sha512 passwords as well as to enable other system options.
<code>system-config-bind</code>	Displays the Domain Name Service window.
<code>system-config-boot</code>	Allows you to specify a default kernel and timeout for the <code>grub.conf</code> file
<code>system-config-display</code>	Brings up the Display Settings window with three tabs: Settings, Hardware, and Dual Head.

system-config-date	Displays the Date/Time Properties window with two tabs: Date & Time and Time Zone. Date & Time: You can set the date and time or enable NTP (Network Time Protocol) from the first tab. Time: The Time Zone tab allows you to specify the time zone of the system clock or set the system clock to <i>UTC</i>
system-config-firewall[-tui] (FEDORA)	Displays the Firewall Configuration window
system-config-httd	Displays the HTTP window with four tabs: Main, Virtual Hosts, Server, and Performance Tuning
system-config-keyboard	Displays the Keyboard window, which allows you to select the type of keyboard attached to the system. You use this utility to select the keyboard when you install the system.
system-config-language	Displays the Language Selection window, which allows you to specify the default system language from among those that are installed. You use this utility to select the system language when you install the system.
system-config-lvm	Displays the Logical Volume Management window, which allows you to modify existing logical volumes
system-config-network[-tui]	Displays the Network Configuration window
system-config-network-cmd	Displays the parameters that system-config-network uses.
system-config-nfs	Displays the NFS Server Configuration window
system-config-packages (RHEL)	Runs pirut
system-config-printer	Displays the Printer Configuration window, which allows you to set up printers and edit printer configurations
system-config-rootpassword	Displays the Root Password window, which allows you to change the root password. While logged in as root, you can also use passwd from a command line to change the root password.
system-config-samba	Displays the Samba Server Configuration window, which can help you configure Samba

system-config-selinux (FEDORA)	Displays the SELinux Administration window, which controls SELinux
system-config-services	Displays the Service Configuration window, which allows you to specify which daemons (services) run at each runlevel.
system-config-soundcard (RHEL)	Displays the Audio Devices window, which tells you which audio device the system detected and gives you the option of playing a sound to test the device
system-config-users	Displays the User Manager window, which allows you to work with users and groups

Command-Line Utilities

- **chsh:** Changes the login shell for a user. When you call chsh without an argument, you change your own login shell. Superuser can change the shell for any user by calling chsh with that user's username as an argument.
- **clear:** Clears the screen. You can also use CONTROL-L from the bash shell to clear the screen.
- **dmesg:** Displays the kernel ring buffer.
- **e2label:** Displays or creates a volume label on an ext2, ext3, or ext4 filesystem. An e2label command has the following format: e2label device [newlabel]

where device is the name of the device (e.g., /dev/hda2, /dev/sdb1, /dev/fd0) you want to work with. When you include the optional newlabel parameter, e2label changes the label on device to newlabel. Without this parameter, e2label displays the label. You can also create a volume label with the -L option of tune2fs.

- **mkfs:** Creates a new filesystem on a device. This utility is a front end for many utilities, each of which builds a different type of filesystem. By default, mkfs builds an ext2 filesystem and works on either a hard disk partition or a floppy diskette. Although it can take many options and arguments, you can use mkfs simply as # mkfs device

where device is the name of the device (e.g., /dev/hda2, /dev/sdb1, /dev/fd0) you want to make a file system on.

- **ping:** Sends packets to a remote system. This utility determines whether you can reach a remote system through the network and tells you how much time it takes to exchange messages with the remote system.
- **reset (link to tset):** Resets terminal characteristics. The value of the environment variable TERM determines how to reset the screen. The screen is cleared, the kill and interrupt characters are set to their default values, and character echo is turned on. When given from a graphical terminal emulator, this command also changes the size of the window to its default. The reset utility is useful to restore your screen to a sane state after it has been corrupted.
- **setserial:** Gets and sets serial port information. Superuser can use this utility to configure a serial port. The following command sets the input address of /dev/ttys0 to 0x100, the interrupt (IRQ) to 5, and the baud rate to 115,000 baud:

```
# setserial /dev/ttys0 port 0x100 irq 5 spd_vhi
```
- **stat:** Displays information about a file or filesystem. Giving the -f (filesystem) option followed by the device name or mount point of a filesystem displays information about the filesystem including the maximum length of filenames.

```
$ stat -f /dev/sda
```

Linux and Shell Scripting

File: "/dev/sda"

ID: 0 Namelen: 255 Type: tmpfs

Block size: 4096 Fundamental block size: 4096

Blocks: Total: 121237 Free: 121206 Available: 121206

Inodes: Total: 121237 Free: 120932

- **umask:** shell builtin that specifies a mask the system uses to set up access permissions when you create a file. A umask command has the following format: `umask [mask]`.

-where mask is a three-digit octal number or a symbolic value such as you would use with chmod. The mask specifies the permissions that are not allowed.

When mask is an octal number, the digits correspond to the permissions for the owner of the file, members of the group the file is associated with, and everyone else. Because mask specifies the permissions that are not allowed, the system subtracts each of the three digits from 7 when you create a file. A mask that you specify using symbolic values indicates the permissions that are allowed.

- **uname:** Displays information about the system. Without any arguments, this utility displays the name of the operating system (Linux). With a -a (all) option, it displays the operating system name, hostname, version number and release date of the operating system, and type of hardware you are using:

```
$ uname -a
```

```
Linux F12 2.6.31.6-145.fc12.i686.PAE #1 SMP Sat Nov 21 16:12:37 EST 2009 i686 athlon i386
GNU/Linux
```

10.8 Standard Rules in Configuration Files

Most configuration files, which are typically named *.conf, rely on the following conventions:

- 1) Blank lines are ignored.
- 2) A # anywhere on a line starts a comment that continues to the end of the line. Comments are ignored.
- 3) When a name contains a SPACE, you must quote the SPACE by preceding it with a backslash (\) or by enclosing the entire name within single or double quotation marks.
- 4) To make long lines easier to read and edit, you can break them into several shorter lines. Break a line by inserting a backslash (\) immediately followed by a NEWLINE (press RETURN in a text editor).

Specifying Clients

Some common ways to specify a host or a subnet:

Client name pattern	Matches
n.n.n.n	One IP address.
name	One hostname, either local or remote.
Name that starts with .	Matches a hostname that ends with the specified string. For example, .tcorp.com matches the systems kudos.tcorp.com and speedy.tcorp.com, among others.

IP address that ends with .	Matches a host address that starts with the specified numbers. For example, 192.168.0. matches 192.168.0.0 – 192.168.0.255. If you omit the trailing period, this format does not work.
Starts with @	Specifies a netgroup.
n.n.n.n/m.m.m.m or n.n.n.n/mm	An IP address and subnet mask specify a subnet.
Starts with /	An absolute pathname of a file containing one or more names or addresses as specified in this table.

Wildcard	Matches
* and ?	Matches one (?) or more (*) characters in a simple hostname or IP address. These wildcards do not match periods in a domain name.
ALL	Always matches.
LOCAL	Matches any hostname that does not contain a period.

Operator	Function
EXCEPT	Matches anything in the preceding list that is not in the following list. For example, a b c d EXCEPT c matches a, b, and d. Thus you could use 192.168. EXCEPT 192.168.0.1 to match all IP addresses that start with 192.168. except 192.168.0.1.

Specifying a Subnet

When you set up a server, you frequently need to specify which clients are allowed to connect to the server. Sometimes it is convenient to specify a range of IP addresses, called a subnet. Usually, you can specify a subnet as

n.n.n.n/m.m.m.m

or

n.n.n.n/maskbits

where n.n.n.n is the base IP address and the subnet is represented by m.m.m.m (the subnet mask) or maskbits (the number of bits used for the subnet mask). Example: 192.168.0.1/255.255.255.0 represents the same subnet as 192.168.0.1/24. In binary, decimal 255.255.255.0 is represented by 24 ones followed by 8 zeros. The /24 is shorthand for a subnet mask with 24 ones.

Different ways to represent a subnet

Bits	Mask	Range
10.0.0.0/8	10.0.0.0/255.0.0.0	10.0.0.0 – 10.255.255.255
172.16.0.0/12	172.16.0.0/255.240.0.0	172.16.0.0 – 172.31.255.255
192.168.0.0/16	192.168.0.0/255.255.0.0	192.168.0.0 – 192.168.255.255

rpcinfo: Displays Information About rpcbind

Fedora uses the rpcbind daemon while RHEL uses portmap for the same purpose. The rpcinfo utility displays information about programs registered with rpcbind and makes RPC calls to programs to see if they are alive. The rpcinfo utility takes the following options and arguments:

```
rpcinfo -p [host]
rpcinfo [-n port] -u | -t host program [version]
rpcinfo -b | -d program version
```

There are various options available, and the associated functions are given in the below table:

Option	Function
-b (broadcast)	Makes an RPC broadcast to version of program and lists hosts that respond.
-d (delete)	Removes local RPC registration for version of program. Available to Superuser only.
-n (port number)	With -t or -u, uses the port numbered port instead of the port number specified by rpcbind.
-p (probe)	Lists all RPC programs registered with rpcbind on host or on the local system if host is not specified.
-t (TCP)	Makes a TCP RPC call to version (if specified) of program on host and reports whether it received a response.
-u (UDP)	Makes a UDP RPC call to version (if specified) of program on host and reports whether it received a response.

Give the following command to see which RPC programs are registered with the rpcbind daemon (portmapper) on the system named peach:

```
$ /usr/sbin/rpcinfo -p peach
program vers proto port
100000 2 tcp 111 portmapper
100000 2 udp 111 portmapper
```

```
100024 1 udp 32768 status
100024 1 tcp 32768 status
100021 1 udp 32769 nlockmgr
100021 3 udp 32769 nlockmgr
```

Locking down rpcbind

Because the rpcbind daemon holds information about which servers are running on the local system and which port each server is running on, only trusted systems should have access to this information.

- One way to ensure only selected systems have access to rpcbind is to lock it down in the /etc/hosts.allow and /etc/hosts.deny files.
- Put the following line in hosts.deny preventing all systems from using rpcbind on the local (server) system: rpcbind: ALL

10.9 The xinetd Superserver

RHEL uses the xinetd daemon, a more secure replacement for the inetd superserver that was originally shipped with 4.3BSD. Fedora uses the Upstart init daemon for runlevel control and most servers. However, some Fedora servers still require xinetd to be installed and running. The xinetd superserver listens for network connections. When one is made, it launches a specified server daemon and forwards the data from the socket to the daemon's standard input. The version of xinetd distributed with Fedora/RHEL is linked against libwrap.so, so it can use the /etc/hosts.allow and /etc/hosts.deny files for access control. Using TCP wrappers can simplify configuration but hides some of the more advanced features of xinetd. The base configuration for xinetd is stored in the /etc/xinetd.conf file. If this file is not present, xinetd is probably not installed. Working as root, give the following command to install xinetd: # yum install xinetd. The default xinetd.conf file is well commented.

Securing a Server

You may secure a server either by using TCP wrappers or by setting up a chroot jail.

TCP Wrappers: Client/Server Security (hosts.allow and hosts.deny)

When you open a local system to access from remote systems, you must ensure that the following criteria are met: Open the local system only to systems you want to allow to access it. Allow each remote system to access only the data you want it to access. Allow each remote system to access data only in the appropriate manner (readonly, read/write, write only). As part of the client/server model, TCP wrappers, which can be used for any daemon that is linked against libwrap.so, rely on the /etc/hosts.allow and /etc/hosts.deny files as the basis of a simple access control language. This access control language defines rules that selectively allow clients to access server daemons on a local system based on the client's address and the daemon the client tries to access.

Each line in the hosts.allow and hosts.deny files has the following format:

daemon_list : client_list [: command]

where daemon_list is a comma-separated list of one or more server daemons (such as rpcbind, vsftpd, or sshd), client_list is a comma-separated list of one or more clients and the optional command is the command that is executed when a client from client_list tries to access a server daemon from daemon_list.

When a client requests a connection with a local server, the hosts.allow and hosts.deny files are consulted in the following manner until a match is found:

1. If the daemon/client pair matches a line in hosts.allow, access is granted.
2. If the daemon/client pair matches a line in hosts.deny, access is denied.

3. If there is no match in either the hosts.allow or hosts.deny files, access is granted.

Setting Up a chroot Jail

On early UNIX systems, the root directory was a fixed point in the filesystem. On modern UNIX variants, including Linux, you can define the root directory on a preprocess basis. The chroot utility allows you to run a process with a root directory other than /. The root directory appears at the top of the directory hierarchy and has no parent: A process cannot access any files above the root directory (because they do not exist). If, for example, you run a program (process) and specify its root directory as /home/sam/jail, the program would have no concept of any files in /home/sam or above: jail is the program's root directory and is labeled / (not jail). By creating an artificial root directory, frequently called a (chroot) jail, you prevent a program from accessing or modifying—possibly maliciously—files outside the directory hierarchy starting at its root. You must set up a chroot jail properly to increase security: If you do not set up the chroot jail correctly, you can make it easier for a malicious user to gain access to a system than if there were no chroot jail.

10.10 DHCP: Configures Hosts

Instead of storing network configuration information in local files on each system, DHCP (Dynamic Host Configuration Protocol) enables client systems to retrieve network configuration information each time they connect to the network. A DHCP server assigns IP addresses from a pool of addresses to clients as needed. Assigned addresses are typically temporary but need not be. This technique has several advantages over storing network configuration information in local files:

- 1) A new user can set up an Internet connection without having to deal with IP addresses, netmasks, DNS addresses, and other technical details. An experienced user can set up a connection more quickly.
- 2) DHCP facilitates assignment and management of IP addresses and related network information by centralizing the process on a server. A system administrator can configure new systems, including laptops that connect to the network from different locations, to use DHCP; DHCP then assigns IP addresses only when each system connects to the network. The pool of IP addresses is managed as a group on the DHCP server.
- 3) IP addresses can be used by more than one system, reducing the total number of IP addresses needed. This conservation of addresses is important because the Internet is quickly running out of IPv4 addresses. Although a particular IP address can be used by only one system at a time, many end-user systems require addresses only occasionally, when they connect to the Internet. By reusing IP addresses, DHCP lengthens the life of the IPv4 protocol.

DHCP is particularly useful for administrators who are responsible for maintaining many systems because individual systems no longer need to store unique configuration information.

How DHCP Works

The client daemon, dhclient (part of the dhcp package), contacts the server daemon, dhcpcd, to obtain the IP address, netmask, broadcast address, nameserver address, and other networking parameters. The server provides a lease on the IP address to the client. The client can request the specific terms of the lease, including its duration; the server can, in turn, limit these terms. While connected to the network, a client typically requests extensions of its lease as necessary, so its IP address remains the same. The lease can expire once the client is disconnected from the network, with the server giving the client a new IP address when it requests a new lease. You can also set up a DHCP server to provide static IP addresses for specific clients

DHCP Client

A DHCP client requests network configuration parameter from the DHCP server and uses those parameters to configure its network interface. The prerequisites is to install the following package: dhclient. When a DHCP client system connects to the network, dhclient requests a lease from the DHCP server and configures the client's network interface(s). Once a DHCP client has requested

and established a lease, it stores information about the lease in a file named dhclient.leases, which is stored in the /var/lib/dhclient directory. This information is used to reestablish a lease when either the server or the client needs to reboot. The DHCP client configuration file, /etc/dhclient.conf, is required only for custom configurations.

DHCP Server

The DHCP server maintains a list of IP addresses and other configuration parameters. When requested to do so, the DHCP server provides configuration parameters to a client. The prerequisites is to install the following package: **dhcp**. Run chkconfig to cause dhcpcd to start when the system enters multiuser mode: # /sbin/chkconfig dhcpcd on.

Start dhcpcd: # /sbin/service dhcpcd start. A simple DHCP server allows you to add clients to a network without maintaining a list of assigned IP addresses. A simple network, such as a home LAN sharing an Internet connection, can use DHCP to assign a dynamic IP address to almost all nodes. The exceptions are servers and routers, which must be at known network locations to be able to receive connections. If servers and routers are configured without DHCP, you can specify a simple DHCP server configuration in /etc/dhcp/dhcpcd.conf (FEDORA) or /etc/dhcpcd.conf (RHEL):

```
$ cat /etc/dhcp/dhcpcd.conf
default-lease-time 600;
max-lease-time 86400;
option subnet-mask 255.255.255.0;
option broadcast-address 192.168.1.255;
option routers 192.168.1.1;
option domain-name-servers 192.168.1.1;
subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.2 192.168.1.200;
}
```

Once you have configured a DHCP server, you can start (or restart) it by using the dhcpcd init script: # /sbin/service dhcpcd restart. Once the server is running, clients configured to obtain an IP address from the server using DHCP should be able to do so.

Static IP Addresses

Routers and servers typically require static IP addresses. While you can manually configure IP addresses for these systems, it may be more convenient to have the DHCP server provide them with static IP addresses. When a system that requires a specific static IP address connects to the network and contacts the DHCP server, the server needs a way to identify the system so the server can assign the proper IP address to the system. The DHCP server uses the MAC address of the system's Ethernet card (NIC) as an identifier. When you set up the server, you must know the MAC address of each system that requires a static IP address. You can use ifconfig to display the MAC addresses of the Ethernet cards (NICs) in a system. The MAC addresses are the colon-separated series of hexadecimal number pairs following HWaddr:

```
$ /sbin/ifconfig | grep -i hwaddr
eth0 Link encap:Ethernet HWaddr BA:DF:00:DF:C0:FF
eth1 Link encap:Ethernet HWaddr 00:02:B3:41:35:98
```

10.11 Important files and directories in Linux

Filesystems hold directories of files. These structures store user data and system data that are the basis of users' work on the system and the system's existence.

~/.bash_profile

It Contains an individual user's login shell initialization script. The shell executes the commands in this file in the same environment as the shell each time a user logs in. The file must be located in a user's home directory. The default Fedora/RHEL .bash_profile file executes the commands in ~./bashrc. You can use .bash_profile to specify a terminal type (for vi, terminal emulators, and other programs), run stty to establish the terminal characteristics, set up aliases, and perform other housekeeping functions when a user logs in. A simple .bash_profile file specifying a vt100 terminal and CONTROL-H as the erase key follows:

```
$ cat .bash_profile
export TERM=vt100
stty erase '^h'
```

~/.bashrc

It Contains an individual user's interactive, nonlogin shell initialization script. The shell executes the commands in this file in the same environment as the (new) shell each time a user creates a new interactive shell. The .bashrc script differs from .bash_profile in that it is executed each time a new shell is spawned, not just when a user logs in. The default Fedora/RHEL .bash_profile file executes the commands in ~./bashrc so that these commands are executed when a user logs in.

/dev

It contains files representing pseudo devices and physical devices that may be attached to the system.

- **/dev/fd0:** The first floppy disk. The second floppy disk is named /dev/fd1.
- **/dev/had:** The master disk on the primary IDE controller. The slave disk on the primary IDE controller is named /dev/hdb. This disk may be a CDROM drive.
- **/dev/hdc:** The master disk on the secondary IDE controller. The slave disk on the secondary IDE controller is named /dev/hdd. This disk may be a CD-ROM drive.
- **/dev/sda:** Traditionally the first SCSI disk; now the first non-IDE drive, including SATA and USB drives. Other, similar drives are named /dev/sdb, /dev/sdc, etc.

These names, such as /dev/sda, represent the order of the devices on the bus the devices are connected to, not the device itself. For example, if you swap the data cables on the disks referred to as /dev/sda and /dev/sdb, the drive's designations will change. Similarly, if you remove the device referred to as /dev/sda, the device that was referred to as /dev/sdb will now be referred to as /dev/sda.

/dev/disk/by-id

It holds symbolic links to local devices. The names of the devices in this directory identify the devices. Each entry points to the device in /dev that it refers to.

/dev/disk/by-uuid

It holds symbolic links to local devices. The names of the devices in this directory consist of the UUID numbers of the devices. Each entry points to the device in /dev that it refers to.

/dev/null

It is also called a bit bucket, output sent to this file disappears. The /dev/null file is a device file and must be created with mknod. Input that you redirect to come from this file appears as nulls, creating an empty file. You can create an empty file named nothing by giving the following command:

Linux and Shell Scripting

It is used by the mail delivery system (typically sendmail) to hold aliases for users. Edit this file to suit local needs.

/etc/at.allow, /etc/at.deny, /etc/cron.allow, and /etc/cron.deny

By default, users can use the at and cron tab utilities. The at.allow file lists the users who are allowed to use at. The cron.allow file works in the same manner for crontab. The at.deny and cron.deny files specify users who are not permitted to use the corresponding utilities. As Fedora/RHEL is configured, an empty at.deny file and the absence of an at.allow file allows anyone to use at; the absence of cron.allow and cron.deny files allows anyone to use crontab. To prevent anyone except Superuser from using at, remove the at.allow and at.deny files. To prevent anyone except Superuser from using crontab, create a cron.allow file with the single-entry root.

/etc/dumpdates

It contains information about the last execution of dump. For each filesystem, it stores the time of the last dump at a given dump level. The dump utility uses this information to determine which files to back up when executing at a particular dump level.

/etc/fstab

Filesystem (mount) table Contains a list of all mountable devices as specified by the system administrator. Programs do not write to this file but only read from it.

/etc/group

Groups allow users to share files or programs without giving all system users access to those files or programs. This scheme is useful when several users are working with files that are not public. The /etc/group file associates one or more usernames with each group (number). An entry in the /etc/group file has four fields arranged in the following format: **group-name:password:group-ID:login-name-list**

The group-name is the name of the group. The password is an optional encrypted password. This field frequently contains an x, indicating that group passwords are not used. The group-ID is a number, with 1-499 reserved for system accounts. The login-name-list is a comma-separated list of users who belong to the group. The login-name-list is a comma-separated list of users who belong to the group. If an entry is too long to fit on one line, end the line with a backslash (\), which quotes the following RETURN, and continue the entry on the next line.

/etc/hosts

The /etc/hosts file stores the name, IP address, and optional aliases of the other systems that the local system knows about. At the very least, this file must have the hostname and IP address that you have chosen for the local system and a special entry for localhost. This entry supports the loopback service, which allows the local system to talk to itself (for example, for RPC services). The IP address of the loopback service is always 127.0.0.1. Following is a simple /etc/hosts file for the system named rose with an IP address of 192.168.0.10:

```
$ cat /etc/hosts
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1 rose localhost.localdomain localhost
192.168.0.1 bravo.example.com bravo
192.168.0.4 mp3server
192.168.0.5 workstation
192.168.0.10 rose
```

...

/etc/inittab (RHEL)

Initialization table Under RHEL, this file controls how the System V init process behaves. Fedora has replaced the System V init daemon with the Upstart init daemon. Each line in inittab contains four colon-separated fields: **id:runlevel:action:process**. The id uniquely identifies an entry in the inittab file. The runlevel is the system runlevel(s) at which process is executed. The runlevel consists of zero or more characters chosen from 0123456S. If more than one runlevel is listed, the associated process is executed at each of the specified runlevels. The action is one of the following keywords: respawn, wait, once, boot, bootwait, ondemand, powerfail, powerwait, powerokwait, powerfailnow, ctrlaltdel, kbrequest, off, ondemand, initdefault, or sysinit. The wait keyword instructs init to start the process and wait for it to terminate.

/etc/motd

It contains the message of the day, which can be displayed each time someone logs in using a textual login. This file typically contains site policy and legal information. Keep this file short because users tend to see the message many times.

/etc/mtab

When you call mount without any arguments, it consults this file and displays a list of mounted devices. Each time you (or an init script) call mount or umount, these utilities make the necessary changes to mtab. Although this is an ASCII text file, you should not edit it.

/etc/netgroup

It defines netgroups, which are used for checking permissions when performing remote logins and remote mounts and when starting remote shells.

/etc/nsswitch.conf

It specifies whether a system uses as the source of certain information NIS, DNS, local files, or a combination, and in what order it consults these services.

/etc/pam.d

Files in this directory specify the authentication methods used by PAM applications.

/etc/passwd

It describes users to the system. Do not edit this file directly. Each line in passwd has seven colon-separated fields that describe one user: **login-name:dummy-password:user-ID:group-ID:info:directory: program**

The login-name is the user's username—the name you enter in response to the login: prompt or GUI login screen. The value of the dummy-password is the character x. An encrypted/hashed password is stored in /etc/shadow. For security reasons, every account should have a password. By convention, disabled accounts have an asterisk (*) in this field. The user-ID is a number, with 0 indicating Superuser and 1-499 being reserved for system accounts. The group-ID identifies the user as a member of a group. It is a number, with 0-499 being reserved for system accounts; see /etc/group. You can change these values and set maximum values in /etc/login.defs. The info is information that various programs, such as accounting programs and email, use to identify the user further. Normally it contains at least the first and last names of the user. It is referred to as the GECOS field. The directory is the absolute pathname of the user's home directory. The program is the program that runs once the user logs in. If program is not present, a value of /bin/bash is assumed. You can put /bin/tcsh here to log in using the TC Shell or /bin/zsh to log in using the Z Shell, assuming the shell you specify is installed.

/etc/printcap

The printer capability database. This file describes system printers and is derived from 4.3BSD UNIX.

/etc/profile

It contains a systemwide interactive shell initialization script for environment and start-up programs. When you log in, the shell immediately executes the commands in this file in the same environment as the shell. Following is an example of a /etc/profile file that displays the message of the day (the /etc/motd file), sets the file-creation mask, and sets the interrupt character to CONTROL-C:

```
# cat /etc/profile
cat /etc/motd
umask 022
stty intr '^c'
```

/etc/protocols

It provides protocol numbers, aliases, and brief definitions for DARPA Internet TCP/IP protocols. Do not modify this file.

/etc/rc.d

It holds the system init scripts, also called run command (rc) scripts. The init program executes several init scripts each time the system changes state or runlevel.

/etc/resolv.conf

The resolver configuration file, used to provide access to DNS. The following example shows the resolv.conf file for the example.com domain. A resolv.conf file usually has at least two lines—a domain line and a nameserver line: # cat /etc/resolv.conf

```
domain example.com
nameserver 10.0.0.50
nameserver 10.0.0.51
```

/etc/rpc

It maps RPC services to RPC numbers. The three columns in this file show the name of the server for the RPC program, the RPC program number, and any aliases.

/etc/services

It lists system services. The three columns in this file show the informal name of the service, the port number/protocol the service frequently uses, and any aliases for the service. This file does not specify which services are running on the local system, nor does it map services to port numbers. The services file is used internally to map port numbers to services for display purposes.

/etc/shadow

It contains encrypted or MD5 hashed user passwords. Each entry occupies one line composed of nine fields, separated by colons: **login-name: password: last-mod: min: max: warn: inactive: expire: flag**. The login-name is the user's username—the name that the user enters in response to the login: prompt or GUI login screen. The password is an encrypted or hashed. The last-mod field

indicates when the password was last modified. The min is the minimum number of days that must elapse before the password can be changed. The max is the maximum number of days before the password must be changed. The warn specifies how much advance warning (in days) to give the user before the password expires. The account will be closed if the number of days between login sessions exceeds the number of days specified in the inactive field. The account will also be closed as of the date in the expire field. The last field in an entry, flag, is reserved for future use. You can use the Password Info tab in system-config-users to modify these fields.

/etc/sysconfig

A directory containing a hierarchy of system configuration files.

/proc

The /proc pseudofilesystem provides a window into the Linux kernel. Through /proc you can obtain information on any process running on your computer, including its current state, memory usage, CPU usage, terminal, parent, and group. You can extract information directly from the files in /proc.

/sbin/shutdown

A utility that brings the system down.

swap

Even though swap is not a file, swap space can be added and deleted from the system dynamically. Swap space is used by the virtual memory subsystem. When it runs low on real memory (RAM), the system writes memory pages from RAM to the swap space on the disk. Which pages are written and when they are written are controlled by finely tuned algorithms in the Linux kernel. When needed by running programs, these pages are brought back into RAM – a technique called paging. When a system is running very short on memory, an entire process may be paged out to disk.

/sys

The /sys pseudofilesystem was added in the Linux 2.6 kernel to make it easy for programs running in kernelspace, such as device drivers, to exchange information with programs running in userspace.

/usr/share/magic

Most files begin with a unique identifier called a magic number. This file is a text database listing all known magic numbers on the system. When you use the file utility, it consults /usr/share/magic to determine the type of a file. Occasionally you may acquire a new tool that creates a new type of file that is unrecognized by the file utility. In this situation you need to update the /usr/share/magic file

/var/log

It holds system log files.

/var/log/messages

It contains messages from daemons, the Linux kernel, and security programs. For example, you will find filesystem full warning messages, error messages from system daemons (NFS, rsyslog, printer daemons), SCSI and IDE disk error messages, and more in messages. Check /var/log/messages periodically to keep informed about important system events. Much of the information displayed

on the system console is also sent to messages. If the system experiences a problem and you cannot access the console, check this file for messages about the problem.

/var/log/secure

It holds messages from security-related programs such as su and the sshd daemon.

10.12 Types of files

Linux supports many types of files:

- 1) Ordinary files, directories, links, and inodes.
- 2) Symbolic links.
- 3) Special files.
- 4) FIFO special file.
- 5) Sockets.
- 6) Block and character devices.
- 7) Raw devices.

Ordinary Files, Directories, Links, and Inodes

Ordinary and directory files

An ordinary file stores user data, such as textual information, programs, or images, such as a jpeg or tiff file. A directory is a standard-format disk file that stores information, including names, about ordinary files, and other directory files.

Inodes

An inode is a data structure, stored on disk, that defines a file's existence and is identified by an inode number. An inode contains critical information, such as the name of the owner of the file, where it is physically located on the disk, and how many hard links point to it. In addition, SELinux stores extended information about files in inodes. A directory relates each of the filenames it stores to an inode. When you move (mv) a file within a filesystem, you change the filename portion of the directory entry associated with the inode that describes the file. You do not create a new inode.

If you move a file to another filesystem, mv first creates a new inode on the destination filesystem and then deletes the original inode. You can also use mv to move a directory recursively, in which case all files in the directory are copied and deleted. When you remove (rm) a file, you delete the directory entry that describes the file. When you remove the last hard link to a file, the operating system puts all blocks the inode pointed to back in the free list (the list of blocks that are available for use on the disk) and frees the inode to be used again.

The . and .. directory entries

Every directory contains at least two entries (. and ..). The . entry is a link to the directory itself. The .. entry is a link to the parent directory. In the case of the root directory, there is no parent and the .. entry is a link to the root directory itself. It is not possible to create hard links to directories.

Symbolic links

Because each filesystem has a separate set of inodes, you can create hard links to a file only from within the filesystem that holds that file. To get around this limitation, Linux provides symbolic links, which are files that point to other files. Files that are linked by a symbolic link do not share an inode. Therefore, you can create a symbolic link to a file from any filesystem. You can also create a symbolic link to a directory, device, or other special file.

Special Files

Special files represent Linux kernel routines that provide access to an operating system feature. FIFO (first in, first out) special files allow unrelated programs to exchange information. Sockets allow unrelated processes on the same or different computers to exchange information. One type of socket, the UNIX domain socket, is a special file. Symbolic links are another type of special file.

Device files

Device files, which include both block and character special files, represent device drivers that let you communicate with peripheral devices, such as terminals, printers, and hard disks. By convention, device files appear in the /dev directory and its subdirectories. Each device file represents a device: You read from and write to the file to read from and write to the device it represents. For example, using cat to send an audio file to /dev/dsp plays the file. The following example shows part of the output that an ls -l command produces for the /dev directory:

```
$ ls -l /dev
total 0
crw-rw--- 1 root root 14, 12 Jan 25 08:33 adsp
crw----- 1 root root 10, 175 Jan 25 08:33 agpgart
crw----- 1 zach root 14, 4 Jan 25 08:33 audio
drwxr-xr-x 3 root root 60 Jan 25 08:33 bus
lrwxrwxrwx 1 root root 3 Jan 25 08:33 cdrom -> hdb
lrwxrwxrwx 1 root root 3 Jan 25 08:33 cdwriter -> hdb
.......
```

The first character of each line is always -, b, c, d, l, or p, representing ordinary (plain), block, character, directory, symbolic link, or named pipe (see the following section), respectively. The next nine characters identify the permissions for the file, followed by the number of hard links and the names of the owner and group. Where the number of bytes in a file would appear for an ordinary or directory file, a device file shows major and minor device numbers separated by a comma. The rest of the line is the same as any other ls -l listing.

udev

The udev utility manages device naming dynamically. It replaces the earlier devfs and moves the device naming functionality from the kernel to userspace. Because devices are added to and removed from a system infrequently, the performance penalty associated with this change is minimal. The benefit of the move is that a bug in udev cannot compromise or crash the kernel. The udev utility is part of the hotplug system. When a device is added to or removed from the system, the kernel creates a device name in the /sys pseudofilesystem and notifies hotplug of the event, which is received by udev. The udev utility then creates the device file, usually in the /dev directory, or removes the device file from the system. The udev utility can also rename network interfaces.

Hotplug

The hotplug system allows you to plug a device into the system and use it immediately. Although hotplug was available in the Linux 2.4 kernel, the 2.6 kernel integrates hotplug with the unified device driver model framework.

FIFO Special Files (Named Pipe)

A FIFO special file, also called a named pipe, represents a pipe: You read from and write to the file to read from and write to the pipe. The term FIFO stands for first in, first out – the way any pipe

Linux and Shell Scripting

works. In other words, the first information that you put in one end is the first information that comes out the other end. When you use a pipe on a command line to send the output of a program to the printer, the printer outputs the information in the same order that the program produced it and sent it to the pipe. The UNIX and Linux systems have included pipes for many generations. Without named pipes, only processes that were children of the same ancestor could use pipes to exchange information. Using named pipes, any two processes on a single system can exchange information. When one program writes to a FIFO special file, another program can read from the same file. The programs do not have to run at the same time or be aware of each other's activity. The operating system handles all buffering and information storage. This type of communication is termed asynchronous (async) because programs on the ends of the pipe do not have to be synchronized.

Sockets

Like a FIFO special file, a socket allows asynchronous processes that are not children of the same ancestor to exchange information. Sockets are the central mechanism of the interprocess communication that forms the basis of the networking facility. When you use networking utilities, pairs of cooperating sockets manage the communication between the processes on the local computer and the remote computer. Sockets form the basis of such utilities as ssh and scp.

Major and Minor Device Numbers

A major device number points to a driver in the kernel that works with a class of hardware devices: terminal, printer, tape drive, hard disk, and so on. In the list of the /dev directory, all of the hard disk partitions have a major device number of 3. A minor device number represents a particular piece of hardware within a class. Although all hard disk partitions are grouped together by their major device number, each has a different minor device number (sda1 is 1, sda2 is 2, and so on). This setup allows one piece of software (the device driver) to service all similar hardware yet to be able to distinguish among different physical units.

Block and Character Devices

A block device is an I/O (input/output) device that is characterized by

- Being able to perform random access reads.
- Having a specific block size.
- Handling only single blocks of data at a time.
- Accepting only transactions that involve whole blocks of data.
- Being able to have a filesystem mounted on it.
- Having the Linux kernel buffer its input and output.

Appearing to the operating system as a series of blocks numbered from 0 through n - 1, where n is the number of blocks on the device. Block devices commonly found on a Linux system include hard disks, floppy diskettes, and CDs.

A character device is any device that is not a block device. Examples of character devices include printers, terminals, tape drives, and modems. The device driver for a character device determines how a program reads from and writes to that device. For example, the device driver for a terminal allows a program to read the information you type on the terminal in two ways:

- First, a program can read single characters from a terminal in raw mode—that is, without the driver doing any interpretation of the characters.
- Alternatively, a program can read one line at a time. When a program reads one line at a time, the driver handles the erase and kill characters, so the program never sees typing mistakes that have been corrected. In this case, the program reads everything from the beginning of a line to the RETURN that ends a line; the number of characters in a line can vary.

Raw Devices

Device driver programs for block devices usually have two entry points so they can be used in two ways: as block devices or as character devices. The character device form of a block device is called a raw device. A raw device is characterized by

- Direct I/O (no buffering through the Linux kernel).
- A one-to-one correspondence between system calls and hardware requests.
- Device-dependent restrictions on I/O.

An example of a utility that uses a raw device is fsck. It is more efficient for fsck to operate on the disk as a raw device, rather than being restricted by the fixed size of blocks in the block device interface. Because it has full knowledge of the underlying filesystem structure, fsck can operate on the raw device using the largest possible units. When a filesystem is mounted, processes normally access the disk through the block device interface, which explains why it is important to allow fsck to modify only an unmounted filesystem. On a mounted filesystem, there is the danger that, while fsck is rearranging the underlying structure through the raw device, another process could change a disk block using the block device, resulting in a corrupted filesystem.

10.13 Filesystems

Filesystem	Features
adfs	Advanced Disc Filing System. Used on Acorn computers. The word Advanced differentiated this filesystem from its predecessor, DFS, which did not support advanced features such as a hierarchical filesystem.
affs	Amiga Fast Filesystem (FFS).
autofs	Automounting filesystem
coda	CODA distributed filesystem
devpts	A pseudofilesystem for pseudoterminals
ext2	A standard filesystem for Fedora/RHEL systems, usually with the ext3 extension.
ext3	A journaling extension to the ext2 filesystem. It greatly improves recovery time from crashes (it takes a lot less time to run fsck), promoting increased availability. As with any filesystem, a journaling filesystem can lose data during a system crash or hardware failure.
ext4	An extension of the ext3 filesystem
GFS	Global Filesystem. GFS is a journaling, clustering filesystem. It enables a cluster of Linux servers to share a common storage pool.
hfs	Hierarchical Filesystem: used by older Macintosh systems. Newer Macintosh

	systems use hfs+.
hpfs	High-Performance Filesystem: the native filesystem for IBM's OS/2.
iso9660	The standard filesystem for CD-ROMs.
minix	Very similar to Linux, the filesystem of a small operating system that was written for educational purposes by Andrew S. Tanenbaum
msdos	The filesystem used by DOS and subsequent Microsoft operating systems. Do not use msdos for mounting Windows filesystems; it does not read VFAT attributes.
ncpfs	Novell NetWare NCP Protocol Filesystem: used to mount remote filesystems under NetWare.
nfs	Network Filesystem. Developed by Sun Microsystems, this protocol allows a computer to access remote files over a network as if they were local
ntfs	NT Filesystem: the native filesystem of Windows NT.
proc	An interface to several Linux kernel <i>data structures</i> <i>that behaves</i> like a filesystem
qnx4	QNX 4 operating system filesystem
reiserfs	A journaling filesystem, based on balanced-tree algorithms.
romfs	A dumb, readonly filesystem used mainly for <i>RAM disks during</i> installation
smbfs	Samba Filesystem
software RAID	RAID implemented in software.
sysv	System V UNIX filesystem.
ufs	Default filesystem under Sun's Solaris operating system and other UNIXs.
umsdos	A full-feature UNIX-like filesystem that runs on top of a DOS FAT filesystem.
vfat	Developed by Microsoft, a standard that allows long filenames on FAT partitions.
xfs	SGI's journaled filesystem (ported from Irix).

10.14 mount: Mounts a Filesystem

The mount utility connects directory hierarchies—typically filesystems—to the Linux directory hierarchy. These directory hierarchies can be on remote and local disks, CDs, and floppy diskettes. Linux also allows you to mount virtual filesystems that have been built inside ordinary files, filesystems built for other operating systems, and the special /proc filesystem, which maps useful Linux kernel information to a pseudodirectory.

Mount point:

The mount point for the filesystem/directory hierarchy that you are mounting is a directory in the local filesystem. This directory must exist before you can mount a filesystem; its contents disappear as long as a filesystem is mounted on it and reappear when you unmount the filesystem. Without any arguments, mount lists the currently mounted filesystems, showing the physical device holding each filesystem, the mount point, the type of filesystem, and any options set when each filesystem was mounted:

```
$ mount
proc on /proc type proc (rw)
/dev/hdb1 on / type ext2 (rw)
/dev/hdb4 on /tmp type ext2 (rw)
/dev/hda5 on /usr type ext3 (rw)
/dev/sda1 on /usr/X386 type ext2 (rw)
/dev/sda3 on /usr/local type ext2 (rw)
/dev/hdb3 on /home type ext3 (rw)
/dev/hda1 on /dos type msdos (rw,umask=000)
tuna:/p04 on /p04 type nfs (rw,addr=192.168.0.8)
/dev/scd0 on /mnt/cdrom type iso9660 (ro,noexec,nosuid,nodev)
```

The mount utility gets this information from the /etc/mtab file. The first entry in the preceding example shows the /proc pseudofilesystem. The next six entries identify disk partitions holding standard Linux ext2 and ext3 filesystems. These partitions are found on three disks: two IDE disks (hda, hdb) and one SCSI disk (sda). Disk partition /dev/hda1 has a DOS (msdos) filesystem mounted at the directory /dos in the Linux filesystem. You can access the DOS files and directories on this partition as if they were Linux files and directories, using Linux utilities and applications. The line starting with tuna shows a mounted, remote NFS filesystem. The last line shows a CD mounted on a SCSI CD drive (/dev/scd0).

Do not mount anything on root (/):

Always mount network directory hierarchies and removable devices at least one level below the root level of the filesystem. The root filesystem is mounted on /; you cannot mount two filesystems in the same place. If you were to try to mount something on /, all files, directories, and filesystems that were under the root directory would no longer be available, and the system would crash.

Mount Options

The mount utility takes many options, which you can specify on the command line or in the /etc/fstab file. For a complete list of mount options for local filesystems, see the mount man page; for remote directory hierarchies, see the nfs man page. The **noauto** option causes Linux not to mount the filesystem automatically. The **nosuid** option forces mounted setuid executables to run with regular permissions (no effective user ID change) on the local system (the system that mounted the filesystem). Unless you specify the user, users, or owner option, only Superuser can mount and unmount a filesystem. The user option means that any user can mount the filesystem, but the filesystem can be unmounted only by the same user who mounted it; users means that any

Linux and Shell Scripting

user can mount and unmount the filesystem. These options are frequently specified for CD and floppy drives. The owner option, which is used only under special circumstances, is similar to the user option except that the user mounting the device must own the device.

Mounting a Linux Floppy Diskette

Mounting a Linux floppy diskette is similar to mounting a partition of a hard disk. Put an entry similar to the following in /etc/fstab for a diskette in the first floppy drive:

```
#/dev/fd0/mnt/floppy auto noauto,users 0 0
```

Specifying a filesystem type of auto causes the system to probe the filesystem to determine its type and allows users to mount a variety of diskettes. Create the /mnt/floppy directory if necessary. Insert a diskette and try to mount it. The diskette must be formatted (use fdformat). Use mkfs to create a filesystem—but be careful, because mkfs destroys all data on the diskette: # **mount /dev/fd0**

mount: you must specify the filesystem type

```
# mkfs /dev/fd0
```

mke2fs 1.41.9 (22-Aug-2009)

Filesystem label=

OS type: Linux

Block size=1024 (log=0)

Fragment size=1024 (log=0)

184 inodes, 1440 blocks

72 blocks (5.00%) reserved for the super user

First data block=1

Maximum filesystem blocks=1572864

1 block group

8192 blocks per group, 8192 fragments per group

184 inodes per group

Writing inode tables: done

Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 24 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.

Try the mount command again:

```
# mount /dev/fd0
```

```
# mount
```

...

```
/dev/fd0 on /mnt/floppy type ext2 (rw,noexec,nosuid,nodev)
```

```
# df -h /dev/fd0
```

Filesystem Size Used Avail Use% Mounted on

```
/dev/fd0 1.4M 19K 1.3M 2% /mnt/floppy
```

The mount command without any arguments and df -h /dev/fd0 show the floppy is mounted and ready for use.

umount: Unmounts a Filesystem

The **umount** utility unmounts a filesystem as long as it does not contain any files or directories that are in use (open). For example, a logged-in user's working directory must not be on the filesystem you want to unmount. The next command unmounts the CD mounted earlier: \$ **umount /mnt/cdrom**. Unmount a floppy or a remote directory hierarchy the same way you would unmount a partition of a hard drive. The umount utility consults /etc/fstab to get the necessary information and then unmounts the appropriate filesystem from its server. When a process has a file open on the filesystem that you are trying to unmount, umount displays a message similar to the following:

```
umount: /home: device is busy
```

Use the -a option to umount to unmount all filesystems, except for the one mounted at /, which can never be unmounted. You can combine -a with the -t option to unmount filesystems of a given type (ext3, nfs, or others). For example, the following command unmounts all mounted nfs directory hierarchies that are not being used:

```
# umount -at nfs
```

fstab: Keeps Track of Filesystems

The system administrator maintains the /etc/fstab file, which lists local and remote directory hierarchies, most of which the system mounts automatically when it boots. The fstab file has six columns, where a hyphen is a placeholder for a column that has no value:

- 1) Name—The name, label, or UUID number of a local block device or a pointer to a remote directory hierarchy. When you install the system, Fedora/RHEL uses UUID numbers for fixed devices. It prefaces each line in fstab that specifies a UUID with a comment that specifies the device name. Using UUID numbers in fstab during installation circumvents the need for consistent device naming.
- 2) Mount point—The name of the directory file that the filesystem/directory hierarchy is to be mounted on. If it does not already exist, create this directory using mkdir.
- 3) Type—The type of filesystem/directory hierarchy that is to be mounted. Local filesystems are generally of type ext2, ext3, or iso9660, and remote directory hierarchies are of type nfs or cifs.
- 4) Mount options—A comma-separated list of mount options, such as whether the filesystem is mounted for reading and writing (rw, the default) or readonly (ro).
- 5) Dump—Used by dump to determine when to back up the filesystem.
- 6) Fsck—Specifies the order in which fsck checks filesystems. Root (/) should have a 1 in this column. Filesystems that are mounted to a directory just below the root directory should have a 2. Filesystems that are mounted on another mounted filesystem (other than root) should have a 3.

The following example shows a typical **fstab** file:

```
# cat /etc/fstab
LABEL=/ 1 / ext3 defaults 1 1
LABEL=/boot1 /boot ext3 defaults 1 2
devpts /dev/pts devpts gid=5,mode=620 0 0
tmpfs /dev/shm tmpfs defaults 0 0
LABEL=/home1 /home ext3 defaults 1 2
proc /proc proc defaults 0 0
sysfs /sys sysfs defaults 0 0
LABEL=SWAP-hda5 swap swap defaults 0 0
/dev/hda3 /oldhome ext3 defaults 0 0
tuna:/p04 /p04 nfs defaults 0 0
/dev/fd0 /mnt/floppy auto noauto,users 0 0
```

fsck: Checks Filesystem Integrity

The **fsck** (**f**ilesystem **c**heck) utility verifies the integrity of filesystems and, if possible, repairs any problems it finds. Because many filesystem repairs can destroy data, particularly on a nonjournaling filesystem, such as ext2, by default fsck asks you for confirmation before making each repair. The following command checks all unmounted filesystems that are marked to be checked in /etc/fstab except for the root filesystem: **# fsck -AR**

The **-A** option causes fsck to check filesystems listed in fstab. The **-R** option checks the same filesystems as **-A** except it does not check the root filesystem. You can check a specific filesystem with a command similar to one of the following:

```
# fsck /home
```

or

```
# fsck /dev/sda6
```

Crash flag: The /etc/rc.d/rc.sysinit start-up script looks for two flags in the root directory of each partition to determine whether fsck needs to be run on that partition before it is mounted. The .autofsck flag (the crash flag) indicates that the partition should be checked.

tune2fs: Changes Filesystem Parameters

The tune2fs utility displays and modifies filesystem parameters on ext2, ext3, and ext4 filesystems. This utility can also set up journaling on an ext2 filesystem, turning it into an ext3 filesystem. With the introduction of increasingly more reliable hardware and software, systems are rebooted less frequently, so it is important to check filesystems regularly. By default, fsck is run on each partition while the system is brought up, before the partition is mounted. Depending on the flags, fsck may do nothing more than display a message saying that the filesystem is clean. The larger the partition, the more time it takes to check it, assuming a nonjournaling filesystem. These checks are often unnecessary. The tune2fs utility helps you to find a happy medium between checking filesystems each time you reboot the system and never checking them. It does so by scheduling when fsck checks a filesystem (these checks occur only when the system is booted). You can use two scheduling patterns: time elapsed since the last check and number of mounts since the last check. The following command causes /dev/sda6 to be checked when fsck runs after it has been mounted eight times or after 15 days have elapsed since its last check, whichever happens first:

```
# tune2fs -c 8 -i 15 /dev/sda6
tune2fs 1.41.9 (22-Aug-2009)
Setting maximal mount count to 8
Setting interval between checks to 1296000 seconds
```

The next tune2fs command is similar but works on a different partition and sets the current mount count to 4. When you do not specify a current mount count, it is set to zero:

```
# tune2fs -c 8 -i 15 -C 4 /dev/sda6
tune2fs 1.41.9 (22-Aug-2009)
Setting maximal mount count to 8
Setting current mount count to 4
Setting interval between checks to 1296000 seconds
```

RAID Filesystem

RAID (Redundant Arrays of Inexpensive/Independent Disks) spreads information across several disks to combine several physical disks into one larger virtual device. RAID improves performance and creates redundancy. More than six types of RAID configurations exist. Using Fedora/RHEL

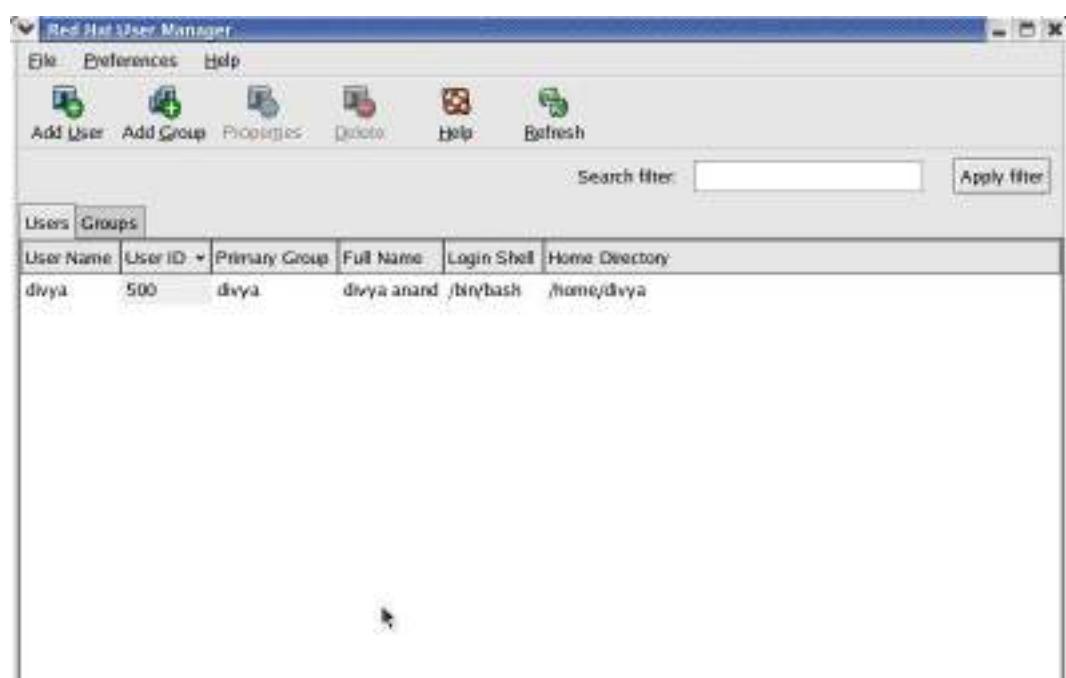
tools, you can set up software RAID. Hardware RAID requires hardware that is designed to implement RAID. RAID can be an effective addition to a backup. Fedora/RHEL offers RAID software that you can install either when you install a Fedora/RHEL system or as an afterthought. The Linux kernel can automatically detect RAID disk partitions at boot time if the partition ID is set to 0xfd, which fdisk recognizes as Linux **raid autodetect**. Software RAID, as implemented in the kernel, is much cheaper than hardware RAID. Not only does this software avoid specialized RAID disk controllers, but it also works with the less expensive IDE disks as well as SCSI disks.

10.15 Configuring User and Group Accounts

More than a username is required for a user to be able to log in and use a system. At a minimum a user must have an entry in the `/etc/passwd` and `/etc/shadow` files and a **home directory**.

system-config-users: Manages User Accounts

The **system-config-users** utility displays the User Manager window and enables you to add, delete, and modify system users and groups. To display the User Manager window, enter **system-config-users** on a command line or select **Main menu: System | Administration | Users and Groups**.



This window has two tabs: **Users and Groups**, where each tab displays information appropriate to its name.

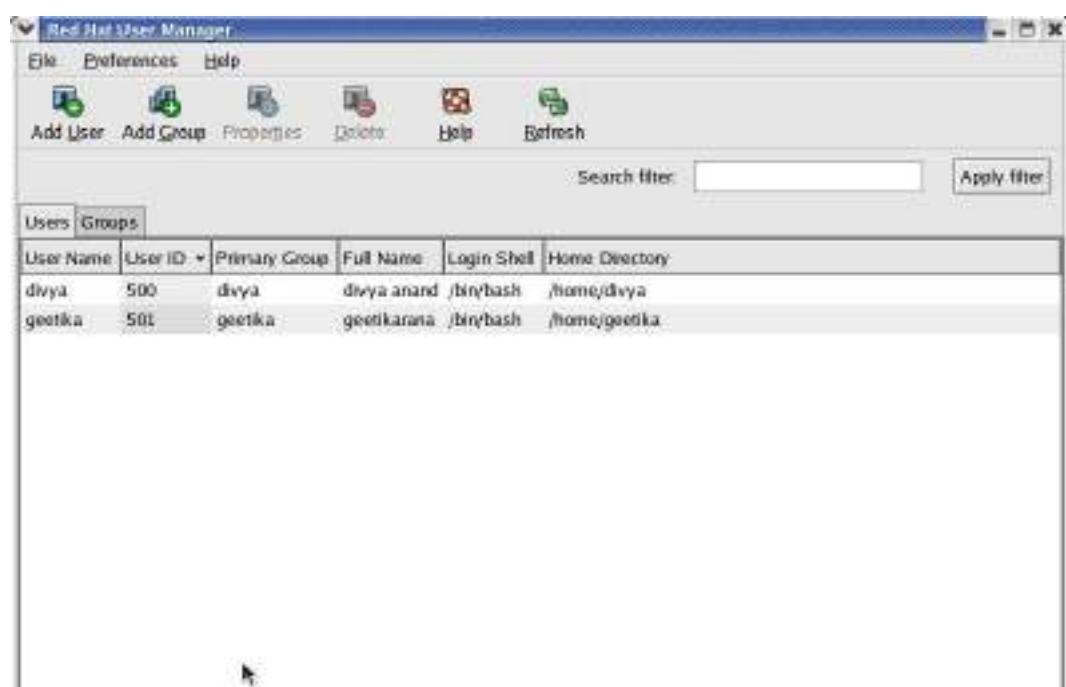
Search filter

The **Search filter**, located just below the toolbar, selects users or groups whose names match the string, which can include wildcards, that you enter in the Search filter text box. The string matches the beginning of a name. For example, *nob matches nobody and nfsnobody, whereas nob matches only nobody.

Adding a user

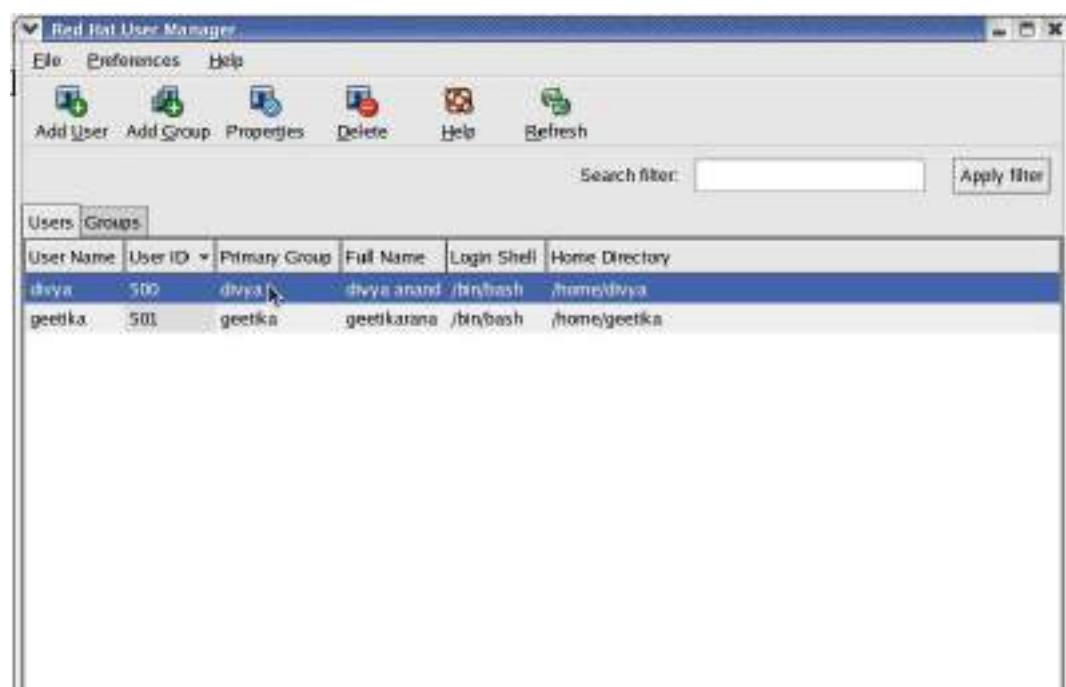
To create a new user, click the **Add User button** on the toolbar. The User Manager displays the Create New User window. Enter the information for the new user and click OK.





Modifying a user

Once you create a user, you can **modify the user** to add/change/remove information. To modify a user, highlight the user in the User Manager window and click Properties on the toolbar; the utility displays the User Properties window.



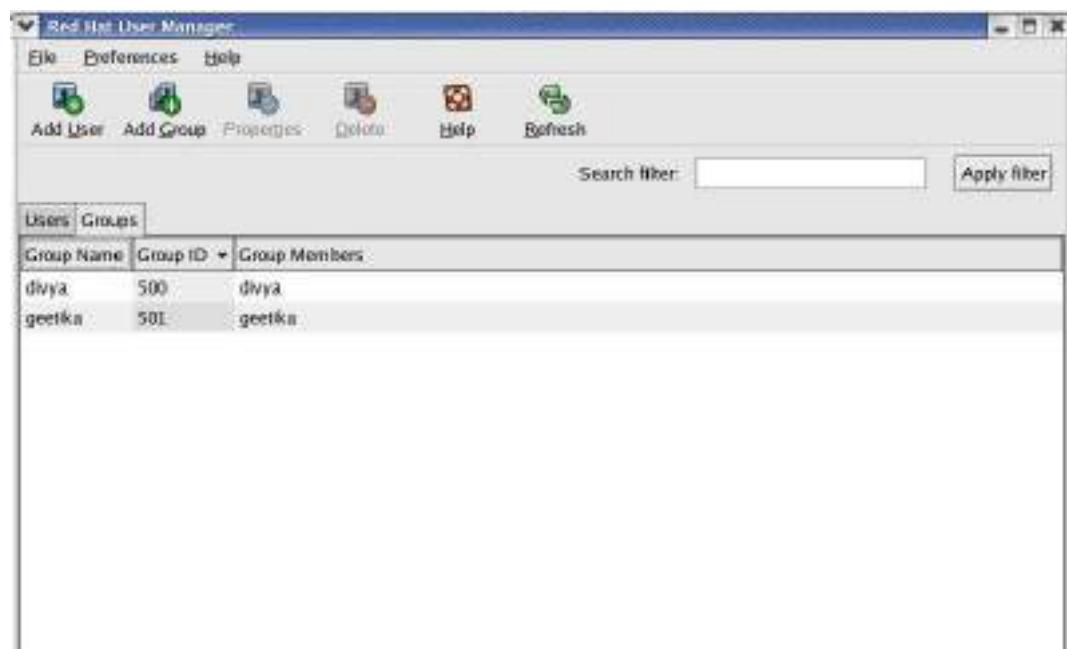


The User Properties window has four tabs: User Data, Account Info, Password Info, and Groups.

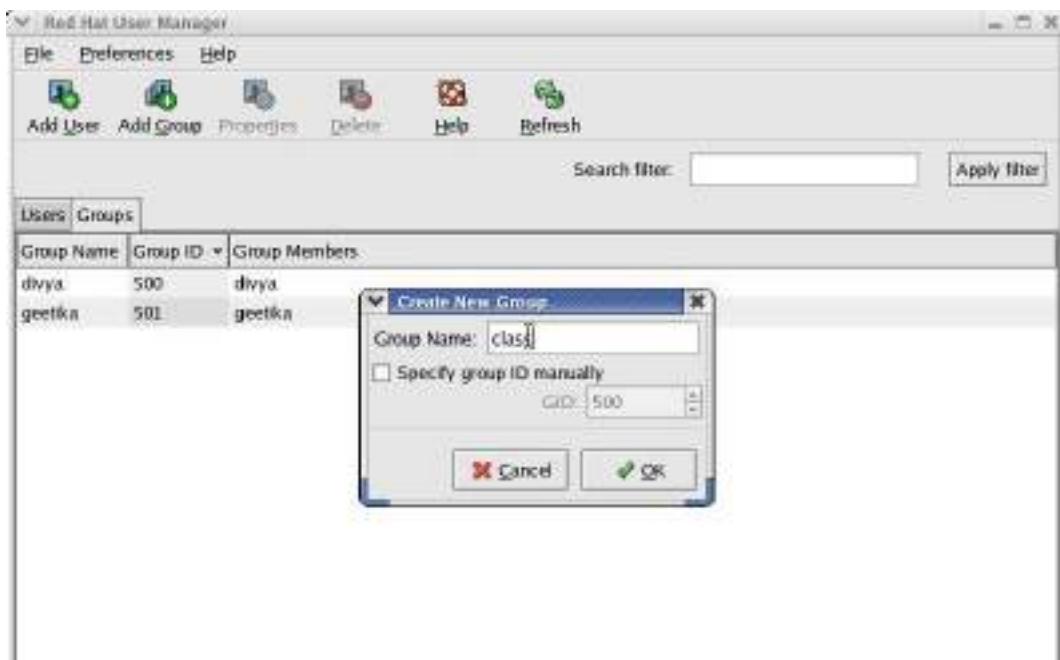
- **User Data tab:** It holds basic user information such as name and password.
- **Account Info tab:** It allows you to specify an expiration date for the account and to lock the account so the user cannot log in.
- **Password Info tab:** It allows you to turn on password expiration and specify various related parameters.
- **Groups tab:** You can specify the groups that the user is a member of.

Working with groups

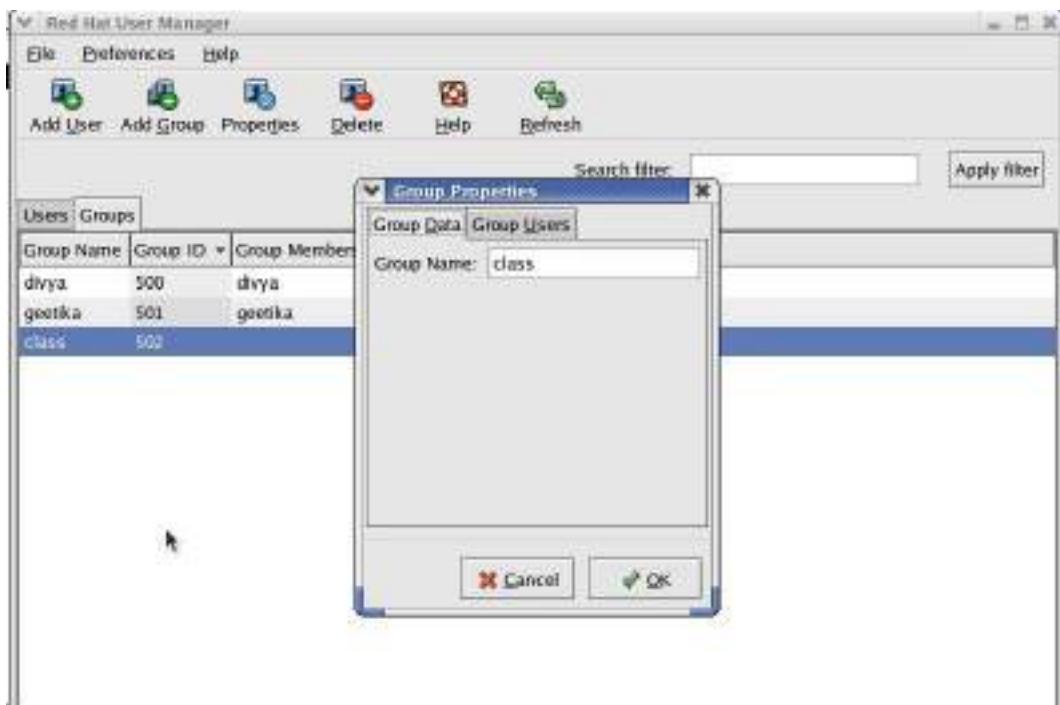
- Click the **Groups tab** in the User Manager window to work with groups.



- To create a group, click **Add Group** on the toolbar and specify the name of the group.

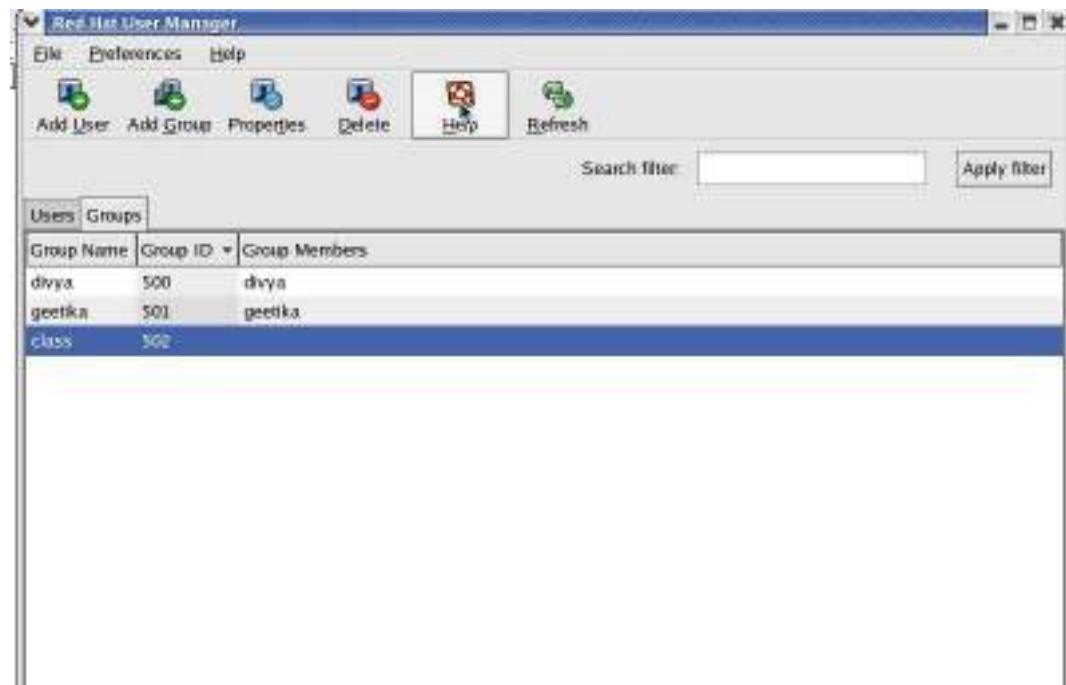


- To change the name of a group or to add or remove users from a group, highlight the group and click Properties on the toolbar.



Help:

- The User Manager provides extensive help. To access it, click Help on the toolbar.
- When you are done working with users and groups, close the window.



useradd: Adds a User Account

The **useradd utility** adds a new user account to the system. By default, useradd assigns the next highest unused user ID to a new account and specifies bash as the user's login shell. This command creates the user's home directory (in /home), specifies the user's group ID, and puts the user's full name in the comment field: # **useradd -g 500 -c "Alex Watson" alex**

userdel: Removes a User Account

The **userdel utility** deletes user accounts. This command removes alex's account and his home directory hierarchy: # **userdel -r alex**. To turn off a user's account temporarily, you can use **usermod** to change the expiration date for the account. Because it specifies that his account expired in the past (December 31, 2009), the following command line prevents alex from logging in:

```
# usermod -e "12/31/09" alex
```

groupadd: Adds a Group

Just as useradd adds a new user to the system, **groupadd** adds a new group by adding an entry for it in **/etc/group**. creates a new group named rtfm: # **groupadd -g 1024 rtfm**

Unless you use the **-g** option to assign a group ID, the system picks the next available sequential number greater than 500. The **-o** option allows the group ID to be nonunique if you want to have multiple names for the same group ID.

10.16 Backing Up Files

The backup copies are vital in three instances:

- 1) When the system malfunctions and files are lost,
- 2) When a catastrophic disaster (fire, earthquake, and so on) occurs,
- 3) When a user or the system administrator deletes or corrupts a file by accident.

Even when you set up RAID, you still need to back up files. Although **RAID provides fault tolerance** (helpful in the event of disk failure), it does not help when a catastrophic disaster occurs

or when a file is corrupted or accidentally removed. You must back up filesystems on a regular basis. Backup files are usually kept on magnetic tape or some other removable media. A **full backup** makes copies of all files, regardless of when they were created or accessed. An **incremental backup** makes copies of those files that have been created or modified since the last (usually full) backup. The more people using the system, the more often you should back up the filesystems. One popular schedule is to perform an incremental backup one or two times a day and a full backup one or two times a week.

Choosing a Backup Medium

If the local system is connected to a network, you can write your backups to a tape drive on another system. This technique is often used with networked computers to avoid the cost of having a tape drive on each computer in the network and to simplify management of backing up many computers in a network. Most likely you want to use a tape system for backups. Other options for holding backups are writable CDs, DVDs, and removable hard disks. These devices, although not as cost-effective or able to store as much information as tape systems, offer convenience and improved performance over using tapes.

Backup Utilities

A number of utilities help you back up the system, and most work with any media. Most Linux backup utilities are based on one of the archive programs—**tar** or **cpio**—and augment these basic programs with bookkeeping support for managing backups conveniently. You can use any of the **tar**, **cpio**, or **dump/restore** utilities to construct full or partial backups of the system. Each utility constructs a large file that contains, or archives, other files. In addition to file contents, an archive includes header information for each file it holds.

The **amanda utility (Advanced Maryland Automatic Network Disk Archiver)**, one of the more popular backup systems, uses dump or tar and takes advantage of Samba to back up Windows systems. The amanda utility backs up a LAN of heterogeneous hosts to a single tape drive. You can use yum to install amanda.

tar: Archives Files

The **tar (tape archive)** utility stores and retrieves files from an archive and can compress the archive to conserve space. If you do not specify an archive device, tar uses standard output and standard input. For displaying the options: `# tar --help | less`

It combines single-letter options into a single command-line argument: `# tar -ztf /dev/st0`. This command uses descriptive words for the same options: `# tar --gzip --list --verbose --file /dev/st0`. Both commands tell tar to generate a (v, verbose) table of contents (t, list) from the tape on /dev/st0 (f, file), using gzip (z, gzip) to decompress the files.

Option	Effect
--append (-r)	Appends files to an archive
--catenate (-A)	Adds one or more archives to the end of an existing archive
--create (-c)	Creates a new archive
--delete	Deletes files in an archive (not on tapes)
--diff (-d)	Compares files in an archive with disk files --extract

--extract (-x)	Extracts files from an archive
--help	Displays a help list of tar options
--list (-t)	Lists the files in an archive
--update (-u)	Like the -r option, but the file is not appended if a newer version is already in the archive

cpio: Archives Files

The **cpio** (**copy in/out**) program is similar to tar but can use archive files in a variety of formats, including the one used by tar. Normally cpio reads the names of the files to insert into the archive from standard input and produces the archive file as standard output. When extracting files from an archive, cpio reads the archive as standard input.

Performing a Simple Backup

When you prepare to make a major change to a system, such as replacing a disk drive or updating the Linux kernel, it is a good idea to archive some or all of the files so you can restore any that become damaged if something goes wrong. For this type of backup, tar or cpio works well. For example, if you have a SCSI tape drive as device **/dev/st0** that is capable of holding all the files on a single tape, you can use the following commands to construct a backup tape of the entire system:

```
# cd /
# tar -cf /dev/st0 .
```

The tar command then creates an archive (c) on the device **/dev/st0** (f). **# tar -cjf /dev/st0 .** You can back up the system with a combination of find and cpio. These commands create an output file and set the I/O block size to 5120 bytes (the default is 512 bytes):

```
# cd /
# find . -depth | cpio -oB >/dev/st0
```

This command restores the files in the **/home** directory from the preceding backup. The options extract files from an archive (-i) in verbose mode, keeping the modification times and creating directories as needed.

```
# cd /
# cpio -ivmd /home/\* </dev/st0
```

dump, restore: Back Up and Restore Filesystems

The **dump utility**, which first appeared in UNIX version 6, backs up either an entire filesystem or only those files that have changed since the last dump. The **restore utility** restores an entire filesystem, an individual file, or a directory hierarchy. This command backs up all files (including directories and special files) on the root (/) partition to SCSI tape 0. Frequently there is a link to the active tape drive, named **/dev/tape**, which you can use in place of the actual entry in the **/dev** directory.

```
# dump -0uf /dev/st0 /
```

The option specifies that the entire filesystem is to be backed up (a full backup). There are ten dump levels: 0–9. Zero is the highest (most complete) level and always backs up the entire filesystem. Each additional level is incremental with respect to the level above it. For example, 1 is incremental to 0 and backs up only files that have changed since the last level 0 dump; 2 is incremental to 1 and backs up only files that have changed since the last level 1 dump; and so on. The **u option** updates

the **/etc/dumpdates** file with filesystem, date, and dump level information for use by the next incremental dump.

- The **f option** and its argument write the backup to the device named **/dev/st0**.
- The **u option** updates the **/etc/dumpdates** file with filesystem, date, and dump level information for use by the next incremental dump.
- The **f option** and its argument write the backup to the device named **/dev/st0**. This command makes a partial backup containing all files that have changed since the last level 0 dump. The first argument is a 1, specifying a level 1 dump:
`# dump -1uf /dev/st0/`
- To restore an entire filesystem from a tape, first restore the most recent complete (level 0) backup. Perform this operation carefully because restore can overwrite the existing filesystem. When you are logged in as Superuser, cd to the directory the filesystem is mounted on and give this command:
`# restore -if /dev/st0`

i: Interactive mode

f: specifies the name of the device that the backup medium is mounted on.

10.17 Scheduling Tasks

It is a good practice to schedule certain routine tasks to run automatically. For example, you may want to remove old core files once a week, summarize accounting data daily, and rotate system log files monthly.

crond and crontab: Schedule Routine Tasks

Using **crontab**, you can submit a list of commands in a format that can be read and executed by **crond**. Working as **Superuser**, you can put commands in one of the **/etc/cron.*** directories to be run at intervals specified by the directory name, such as cron.daily.

at: Runs Occasional Tasks

Like the cron utility, at allows you to run a job sometime in the future. Unlike cron, at runs a job only once.

System Reports

Many utilities report on one thing or another. The **who**, **finger**, **ls**, **ps**, and other utilities generate simple end-user reports. In some cases, these reports can help with system administration.

vmstat: Reports Virtual Memory Statistics

The **vmstat** utility (procps package) generates virtual memory information along with (limited) disk and CPU activity data.

```
$ vmstat 3 7
procs -----memory----- -----swap-----io-----system-----cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa
0 2 0 684328 33924 219916 0 0 430 105 1052 134 2 4 86 8
0 2 0 654632 34160 248840 0 0 4897 7683 1142 237 0 5 0 95
0 3 0 623528 34224 279080 0 0 5056 8237 1094 178 0 4 0 95
0 2 0 603176 34576 298936 0 0 3416 141 1161 255 0 4 0 96
0 2 0 575912 34792 325616 0 0 4516 7267 1147 231 0 4 0 96
1 2 0 549032 35164 351464 0 0 4429 77 1120 210 0 4 0 96
0 2 0 523432 35448 376376 0 0 4173 6577 1135 234 0 4 0 95
```

vmstat column heads

This shows virtual memory statistics in 3-second intervals for seven iterations. The first line covers the time since the system was last booted; the rest of the lines cover the period since the previous line.

procs	Process information
	r Number of waiting, runnable processes
	b Number of blocked processes (in uninterruptable sleep)
Memory	Memory Information in Kilobytes
	swpd Used virtual memory
	free Idle memory
	buff Memory used as buffers
	cache Memory used as cache
swap	System paging activity in kilobytes per second
	si Memory swapped in from disk
	so Memory swapped out to disk
io	System I/O activity in blocks per second
	bi Blocks received from a block device

	bo Blocks sent to a block device
system	Values are per second
	in Interrupts (including the clock)
	cs Context switches
cpu	Percentage of total CPU time spent in each of these states
	us User (nonkernel)
	sy System (kernel)
	id Idle
	wa Waiting for I/O

top: Lists Processes Using the Most Resources

The **top utility** is a useful supplement to ps. At its simplest, top displays system information at the top and the most CPU-intensive processes below the system information. The top utility updates itself periodically; type q to quit.

```
$ top
top - 21:30:26 up 18 min,  2 users,  load average: 0.95, 0.30, 0.14
Tasks: 63 total,  4 running, 58 sleeping,  1 stopped,  0 zombie
Cpu(s): 30.9% us, 22.9% sy,  0.0% ni,  0.0% id, 45.2% wa,  1.0% hi,  0.0%si
Mem: 1036820k total, 1032276k used,    4544k free,   40908k buffers
Swap: 2048276k total,      0k used, 2048276k free, 846744k cached

 PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM     TIME+ COMMAND
1285 root      25   0 9272 6892 1312 R 29.3  0.7  0:00.88 bzip2
1276 root      18   0 3048  860 1372 R  3.7  0.1  0:05.25 cp
  7 root      15   0    0    0  0 S  0.7  0.0  0:00.27 pdflush
  6 root      15   0    0    0  0 S  0.3  0.0  0:00.11 pdflush
  8 root      15   0    0    0  0 S  0.3  0.0  0:00.06 kswapd0
 300 root      15   0    0    0  0 S  0.3  0.0  0:00.24 kjournald
1064 mgs2     16   0 8144 2276 6808 S  0.3  0.2  0:00.69 sshd
1224 root      16   0 4964 1360 3944 S  0.3  0.1  0:00.03 bash
1275 mgs2     16   0 2840  936 1784 R  0.3  0.1  0:00.15 top
1284 root      15   0 2736  668 1416 S  0.3  0.1  0:00.01 tar
  1 root      16   0 2624  520 1312 S  0.0  0.1  0:06.51 init
```

top: interactive commands

Command	Function
F	Specify a sort field.
h or ?	Displays a Help screen.
k	Prompts for a PID number and type of signal and sends the process that signal. Defaults to signal 15 (SIGTERM); specify 9 (SIGKILL) only when 15 does not work.
M	Sorts processes by memory usage.
O	Specify a sort field.
P	Sorts processes by CPU usage (default).
q	Quits.
S	Prompts for time between updates in seconds. Use 0 for continuous updates.
SPACE	Updates the display immediately.
T	Sorts tasks by time.
W	Writes a startup file named <code>~/.toprc</code> so that next time you start top, it uses the same parameters it is currently using.

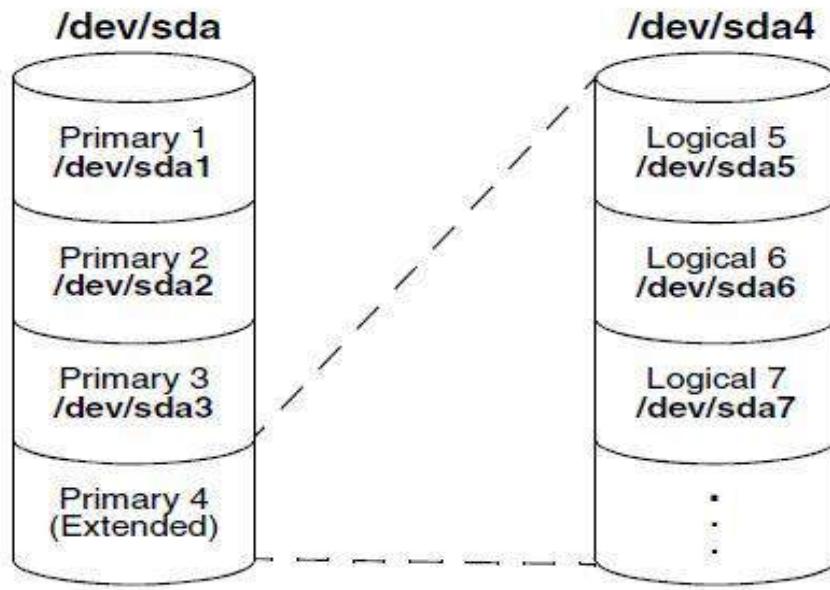
parted: Reports on and Partitions a Hard Disk

- The **parted (partition editor) utility** reports on and manipulates hard disk partitions.

The following example shows how to use parted from the command line

```
# parted /dev/sda print
Disk geometry for /dev/sda: 0kB - 165GB
Disk label type: msdos
Number  Start   End     Size    Type      File system  Flags
 1      32kB   1045MB  1045MB  primary   ext3          boot
 2      1045MB  12GB    10GB    primary   ext3
 3      12GB    22GB    10GB    primary   ext3
 4      22GB    165GB   143GB   extended
 5      22GB    23GB    1045MB  logical   linux-swap
 6      23GB    41GB    18GB    logical   ext3
 7      41GB    82GB    41GB    logical   ext3
Information: Don't forget to update /etc/fstab, if necessary.
```

Partitions



`parted`: Reports on and Partitions a Hard Disk

The `print` command displays the following columns:

- **Number**—The minor device number of the device holding the partition. This number is the same as the last number in the device name. In the example, 5 corresponds to `/dev/sda5`.
- **Start**—The location on the disk where the partition starts. The `parted` utility specifies a location on the disk as the distance (in bytes) from the beginning of the disk. Thus partition 3 starts 12 gigabytes from the beginning of the disk.
- **End**—The location on the disk where the partition stops. Although partition 2 ends 12 gigabytes from the beginning of the disk and partition 3 starts at the same location, `parted` takes care that the partitions do not overlap at this single byte.
- **Size**—The size of the partition in kilobytes (kB), megabytes (MB), or gigabytes (GB).
- **Type**—The partition type: primary, extended, or logical.
- **File system**—The filesystem type: ext2, ext3, fat32, linux-swap, and so on.
- **Flags**—The flags that are turned on for the partition, including boot, raid, and lvm. In the example, partition 1 is bootable

Summary:

- Superuser can use certain tools, such as `sudo`, to give specific users permission to perform tasks that are normally reserved for Superuser.
- To bring a system up in rescue mode, boot the system from the first installation CD, the Net Boot CD, or the install DVD.
- SELinux can be in one of three states: enforcing, permissive and disabled.
- The `system-config-selinux` utility displays the SELinux Administration window, which controls SELinux.
- By default, Fedora systems boot to graphical multiuser mode (runlevel 5).
- The `system-config-services` utility displays the Service Configuration window.
- The `shutdown` and `halt` utilities perform the tasks needed to bring the system down safely.
- The key combination `CONTROL-ALT-DEL` Reboots the System

- Most of the Fedora/RHEL configuration tools are named system-config-*.
- When you set up a server, you frequently need to specify which clients are allowed to connect to the server. Sometimes it is convenient to specify a range of IP addresses, called a subnet.
- RHEL uses the xinetd daemon, a more secure replacement for the inetd superserver that was originally shipped with 4.3BSD.
- You may secure a server either by using TCP wrappers or by setting up a chroot jail.
- An ordinary file stores user data, such as textual information, programs, or images, such as a jpeg or tiff file.
- A character device is any device that is not a block device. Examples of character devices include printers, terminals, tape drives, and modems.

Keywords

- **System Administrator:** A system administrator should be available to help users with all types of system-related problems—from logging in to obtaining and installing software updates to tracking down and fixing obscure network issues.
- **su:** The su (substitute user) utility can create a shell or execute a program with the identity and permissions of a specified user.
- **Consolehelper:** The consolehelper utility can make it easier for someone who is logged in on the system console but not logged in as root to run system programs that normally can be run only by root.
- **Trojan House:** A Trojan horse is a program that does something destructive or disruptive to a system while appearing to be benign.
- **runlevel utility:** The runlevel utility displays the previous and current runlevels. This utility is a transitional tool; it provides compatibility with SysVinit.
- **Booting:** Booting a system is the process of reading the Linux kernel into system memory and starting it running.
- **rpcinfo:** The rpcinfo utility displays information about programs registered with rpcbind and makes RPC calls to programs to see if they are alive.
- **Inode:** An inode is a data structure, stored on disk, that defines a file's existence and is identified by an inode number.
- **Sockets:** Sockets allow unrelated processes on the same or different computers to exchange information.
- **Hotplug:** The hotplug system allows you to plug a device into the system and use it immediately.
- **Mount point:** The mount point for the filesystem/directory hierarchy that you are mounting is a directory in the local filesystem.
- **umount:** The umount utility unmounts a filesystem as long as it does not contain any files or directories that are in use (open).

Self Assessment

1. su stands for
 - A. substitute user
 - B. switch user
 - C. substandard user

- D. None of the above
2. Which of these tools gives you another user's privileges?
- A. kill
 - B. su
 - C. consolehelper
 - D. None of the above
3. Which of these tools runs programs as a root?
- A. kill
 - B. su
 - C. consolehelper
 - D. None of the above
4. Who is a superuser in Linux environment?
- A. Root
 - B. Normal user
 - C. Machine
 - D. None of the above
5. What are the modes of SELinux?
- A. Enforcing
 - B. Permissive
 - C. Disabled
 - D. All of the above
6. The policies of SELinux are
- A. Targeted
 - B. MLS
 - C. Strict
 - D. All of the above
7. Which of these key combinations reboots the system?
- A. CTRL-ALT-HOME
 - B. CTRL-DEL-END
 - C. CTRL-ALT-DEL
 - D. CTRL-TAB-DEL
8. Which of these utility displays the kernel ring buffer?
- A. chsh
 - B. clear
 - C. dmesg
 - D. None of the above

9. Which of these utility creates a new filesystem on device?
 - A. mkfs
 - B. chsh
 - C. dmesg
 - D. clear

10. Which superserver listens for network connection?
 - A. xinetd
 - B. Machine
 - C. Fedora
 - D. None of the above

11. How can we secure a server?
 - A. By using TCP wrappers
 - B. By setting up a chroot jail
 - C. By using both of the above
 - D. None of the above

12. The user's interactive non-login shell initialization script is located in _____ file.
 - A. `~/.bash_profile`
 - B. `~/.bashrc`
 - C. `/dev`
 - D. None of the above

13. Which of these is known as a bit bucket?
 - A. `/dev/empty`
 - B. `/dev/bucket`
 - C. `/dev/null`
 - D. None of the above

14. The _____ option causes Linux not to mount the filesystem automatically.
 - A. noauto
 - B. nosuid
 - C. nomount
 - D. None of the above

15. The _____ backup makes copies of all the files.
 - A. Full
 - B. Incremental
 - C. Decremental
 - D. Half

Answer for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. A | 2. B | 3. C | 4. A | 5. D |
| 6. D | 7. C | 8. C | 9. A | 10. A |
| 11. C | 12. B | 13. C | 14. A | 15. A |

Review Questions

1. What is a well-maintained system? What kind of powers a superuser has?
2. How can we gain/grant the superuser privileges?
3. What are system administration tools? Explain.
4. What is a rescue mode? How can we avoid trojan horse?
5. What is security enhanced linux? What are states and policies of SELinux?
6. What is a run-level? Explain various utilities associated with this.
7. What is Fedora/RHEL configuration tools? Explain.
8. What are command line utilities?
9. What are standard rules in configuration files? How can we specify clients?
10. How can we secure a server?
11. What are important files and directories?
12. What kind of files are supported by Linux?
13. What is fstab? How can we keep track of filesystems? Explain the columns of fstab file.
14. What is a special file in Linux?
15. Explain various types of file systems in Linux?
16. What is backing up of files? How can we choose a backup medium? Explain various backup utilities.
17. How can we schedule tasks? What are vmstat column heads?

**Further Readings**

Mark G. Sobell, A Practical Guide to Fedora and RedHat Enterprise Linux, Fifth Edition, Prentice Hall

**Web Links**

<https://www.beyondtrust.com/resources/glossary/superuser-superuser-accounts#:~:text=In%20Linux%20and%20Unix%2Dlike,any%20permissions%20for%20other%20users.>

<https://searchsecurity.techtarget.com/definition/sudo-superuser-do>

Unit 11: Web Server Configuration

CONTENTS

- Objectives
- Introduction
- 11.1 The Apache Web Server
- 11.2 Installing Apache
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions:
- Further Readings

Objectives

After studying this unit, you will be able to

- Know what a web server is
- Understand how to set up a web server
- Know how to install a web server on Linux machine
- Understand Apache web server
- Understand the important packages of Apache web server

Introduction

A web server is nothing but software and hardware that uses the Hypertext Transfer Protocol, commonly known as HTTP, and some other protocols that respond to request from clients made on the World Wide Web. The main job that the webserver performs is to display the content of the website, which it does by storing, then processing, and eventually delivering the webpages to the user who has requested it. The web server also supports Simple Mail Transfer Protocol or SMTP and File Transfer Protocol or FTP and HTTP. These are used to transfer files for emailing and even for storage.

The web server hardware gets connected to the internet, allowing the exchange of data with the other devices related to it. The web server software controls how the user assesses the files that have been hosted. The web server process is basically a client and server model example. All the computers that host websites should have the webserver software. Web servers find use in web hosting or the hosting of data for websites as well as for all kinds of web-based applications. The web server software gets accessed using the domain name of a website. This then ensures that the content of the site gets delivered to the user who has requested it. The software part of the webserver is also made up of various components and has at least a single HTTP server. The HTTP server understands the URLs and the HTTP. The web server hardware is basically a computer that will store the webserver software as well as the files that are related to the website. These include documents, HTML, JavaScript files, and images.

Uses of Web Server

The web server is basically a part of a large internet package. It also offers many programs that are related to the intranet. The web server is used to:

- Send and receive emails.
- Download the file transfer protocol or FTP request
- Build and publish webpages.

The basic kinds of web servers can support the scripting on the server-side, which is used to employ the scripts on the webserver. This can be customized as per the request of the client. The serve side scripting works on the server machine, and this comes with a broad feature set that offers access to the database. The server-side scripting makes use of Active Server Pages or ASP, Hypertext Pre-processor or PHP, and many other scripting languages. The process also lets the HTML documents to get created.

Dynamic and Static Web Servers

A web server may be used as static or dynamic content. The static content is the one that is fixed. The static web server contains HTTP software and computer. This is static because the server sends hosted files as is present to the browser.

On the other hand, the dynamic web browser will have the webserver and the software like the database and the applications server. This is dynamic because the application server is used to update the files hosted before these are sent to the browser. The web server generates content when the database requests it. The process is flexible but complicated too.

A web server can host a single website or many websites with the help of the same software and hardware resource. This is called virtual hosting. The answer to what is the role of a web server is here. Web servers are also capable of limiting the speed of the response to various clients, which in turn does not allow a single client to dominate the resources. This is used to satisfy the requests of many clients. The web servers will typically host the websites that are internet accessible. These can also be used to communicate between the web clients and the servers in the local network area. This could be like through the intranet of a company. The web server could be embedded in a device like a digital camera. This lets the users communicate with the device using a commonly available web browser.

Basic Common Features

HTTP: The support for one or more versions of HTTP protocol in order to send versions of HTTP responses compatible with versions of client HTTP requests, e.g., HTTP/1.0, HTTP/1.1 plus, if available, HTTP/2, HTTP/3.

Logging: Usually web servers have also the capability of logging some information, about client requests and server responses, to log files for security and statistical purposes.

Setting up a web server

When we want to publish web pages on the Internet or on an intranet, we use a web server. In essence, a web server is an application that does two things:

- 1) It listens for page requests.
- 2) When it receives a page request, it examines the request and responds with the page that was requested.

Of course, there are many different web browsers in existence (including Mozilla, Opera, Internet Explorer, and others), and there are also a great many types of web server software. To enable a browser to request pages from a web server, they communicate using Hypertext Transfer Protocol (HTTP) – this is the standard protocol for the Internet. The request and response messages are composed using HTTP, and this is what allows any browser to request web pages from any type of web server. By default, all web servers listen for HTTP requests on port 80. Web servers also use port 443 to listen for requests made through secure HTTP connections, over SSL (secure sockets

layer), through a protocol called HTTPS. So, if you want to publish your own web site, you'll need a machine with some web server software.

Why to install a web server on Red Hat Linux Machine

Well, here are two scenarios:

- First, if you're building a web site, then you'll need a web server so that you can test your site as you're developing it.
- Second, although you might not host an Internet site from your own machine, you might host an intranet site – a private web site available only to other machines inside your private network.

Commercial and open-source web servers

Commercial

- Netscape
- Iplanet
- SunONE
- Microsoft
- Zeus

Open Source

- Apache
- Thttpd
- Redhat TUX

11.1 The Apache Web Server

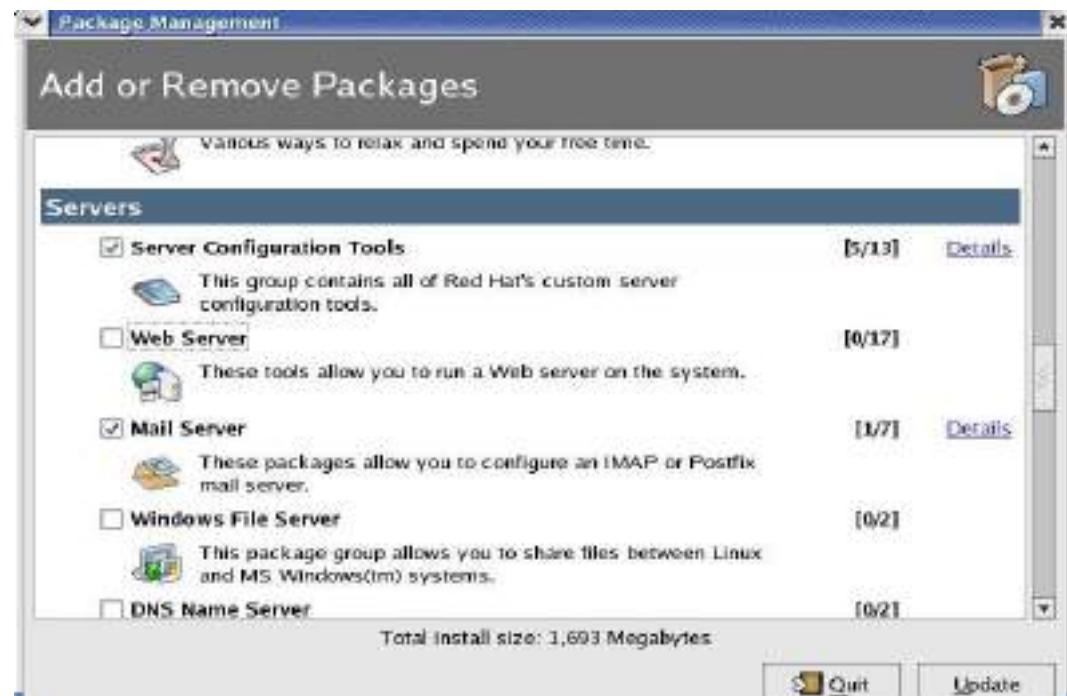
At the time of writing, 66% of all web sites are hosted on Apache web servers, most of them running on Linux or Unix operating systems. Apache's popularity is due not only of its open-source pedigree, but also to its highly competitive levels of performance, functionality, stability, flexibility, and security.

- 1) **Flexibility:** Apache's flexibility comes from the fact that it is a modular web server. That means that you can meet your requirements by plugging any number of external modules into the core httpd daemon. Of course, being open-source software, you also have access to Apache's source code, which you can customize to fit your needs.
- 2) **Scalable & Portable:** Apache is also very scalable. You can run Apache on high-end hardware, and it's possible to increase the capacity of Apache web servers by sharing the load across any number of servers. It's also very portable, being available for several operating systems.
- 3) **Security:** Apache's security is very good in comparison to other web servers. Moreover, the Apache Foundation is extremely active in the continued defense of Apache from security problems – particularly in the form of announcements and patches.
- 4) **Functionality:** Apache performs very well – it boasts a highly optimized daemon for serving static content which dramatically outperforms its nearest rivals. Moreover, it rarely crashes and achieves extremely long up-times.
- 5) **Documentation:** Apache comes with detailed documentation, which helps to make the setup and configuration easy.
- 6) **Support:** There's a wide network of support for Apache, in the form of mailing lists, newsgroups, and commercial vendors like Red Hat.
- 7) **Stability:** Apache development is active. The Apache Foundation is actively involved in development of new modules; new versions of Apache to make it reliable stable and secure.

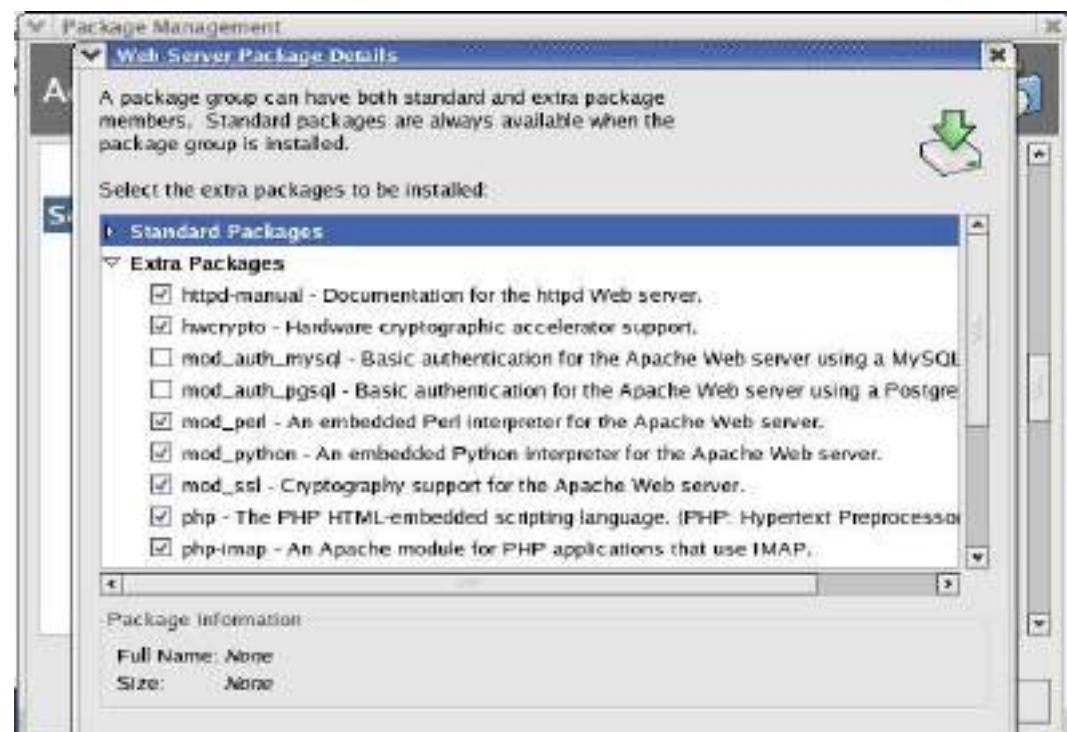
11.2 Installing Apache

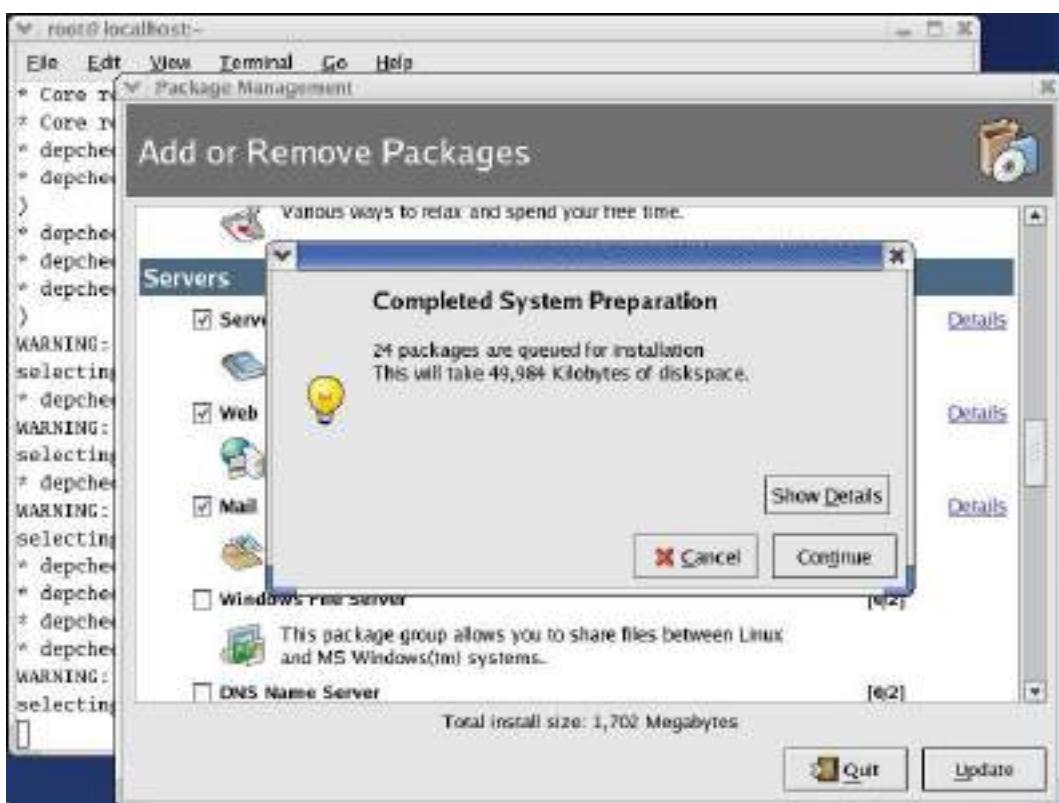
Apache is a modular server – the core server provides the basic functionality, with extended features available in various modules. This makes it very flexible and easy to configure, because you need to configure only the modules you need. So, it's worth looking at how to control the installation and removal of these modules. There are two ways to start RPM's graphical interface:

One is to go to Main Menu | then System Settings | then Add/Remove Applications. Other is to open \$ red hat-config-packages.



In this we need to search web server package.





Important packages

There are some important packages which needs to be installed for web server. These are:

Package	Description
httpd-manual	Contains the documentation for the Apache web server. After installation, you can access this documentation from the command line by typing man httpd.
hwcrypto	Provides support for hardware SSL acceleration cards. This package should be installed if you have hardware SSL acceleration cards like Ncipher Nforce on your server.
mod_ssl	Provides an SSL interface to the HTTPS web server, and hence enables the Apache web server to support SSL. This package should be installed if you want to provide secure connections to your clients.
PHP	Provides the PHP module for Apache, which enables the web server to serve PHP web pages. This package is required if you want to host web sites which contain pages written with the PHP scripting language.
webalizer	Provides programs for web server log file analysis. This package enables you to generate HTML usage reports for your website.

Apache Configuration Files

There are various configuration files for configuring the Apache web server in the computer system.

- 1) The /etc/httpd/httpd.conf file is Apache's main configuration file.
- 2) The /etc/httpd/conf.d directory contains configuration files for any installed modules (such as PHP, SSL, and so on).
- 3) The /etc/httpd/logs directory is a symbolic link to /var/log/httpd directory, which contains all the Apache log files.
- 4) The /etc/httpd/modules directory is a symbolic link to /usr/lib/httpd/modules directory, which contains all the Apache modules configured as dynamic shared objects. Dynamic shared objects (or DSOs) are modules that are compiled separately from the Apache httpd binary. They are so-called because they can be loaded on demand.
- 5) The /etc/httpd/run directory is a symbolic link to /var/run, which contains the process ID file (httpd.pid) of the httpd process.
- 6) /etc/rc.d/init.d/httpd is a shell script, used for starting and stopping the Apache web server.

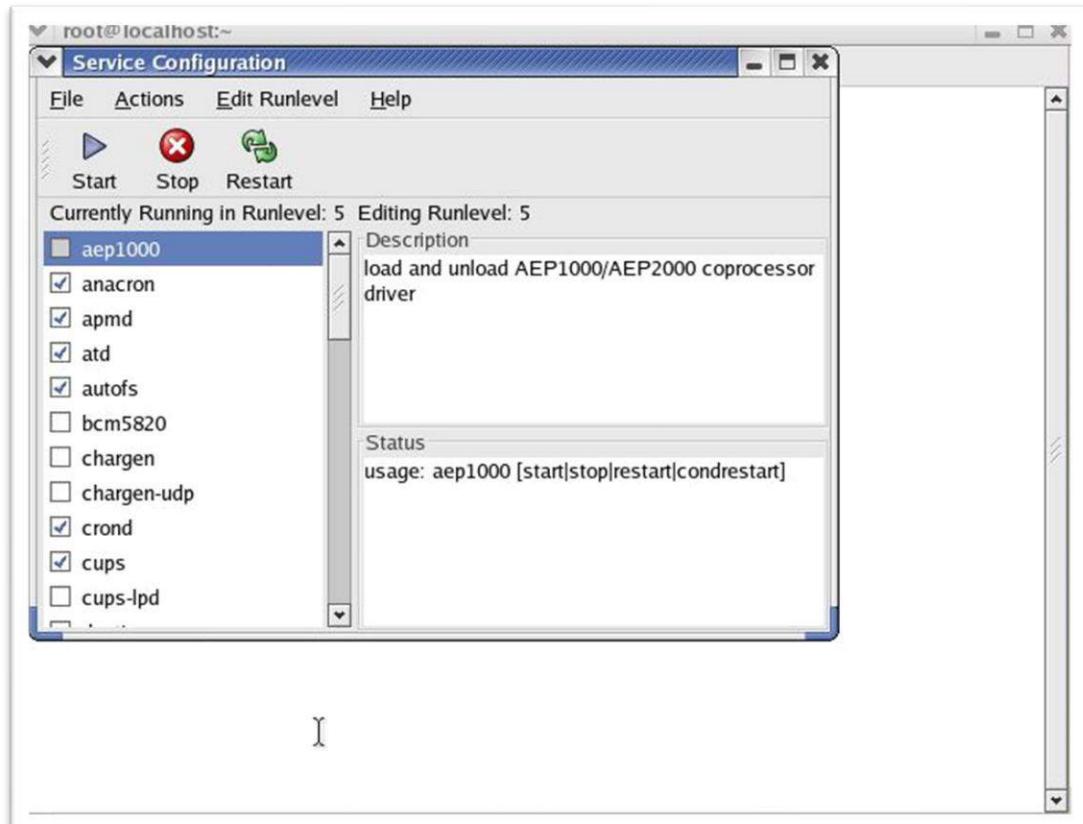
Starting Apache for the First Time

There are two ways by which Apache can be started. These are:

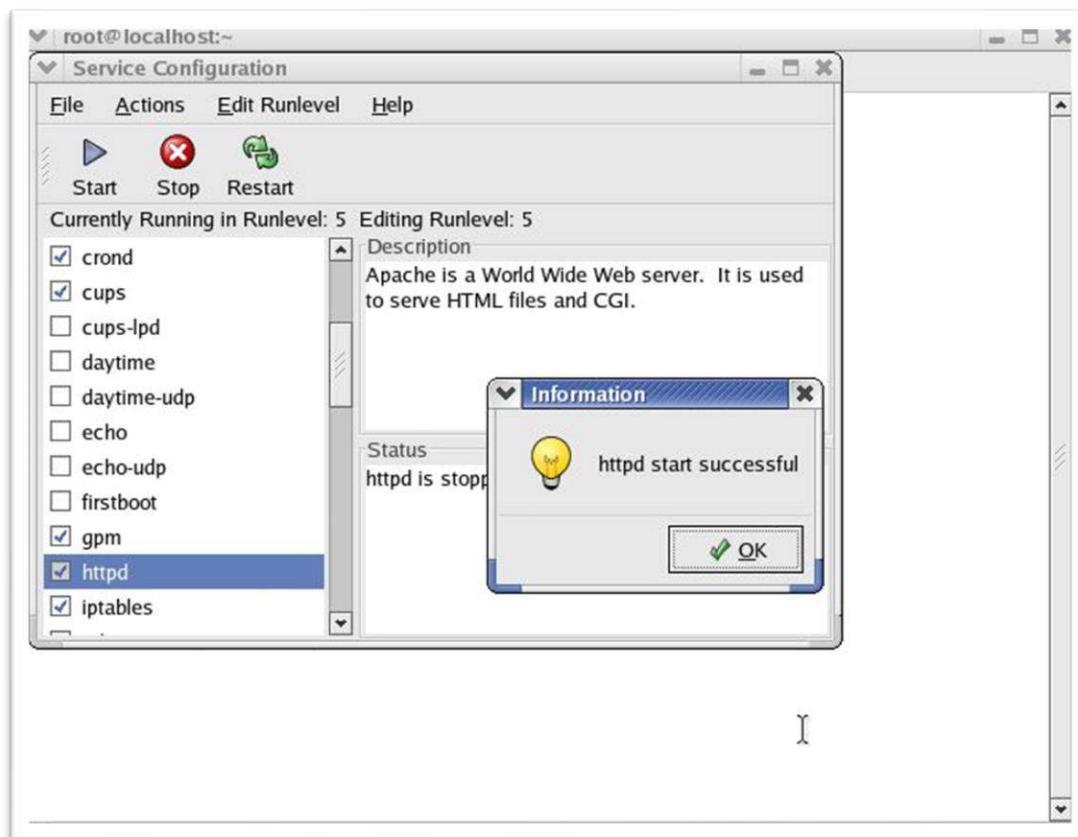
- 1) Main Menu | System Settings | Server Settings | Services
- 2) \$ redhat-config-services

We can opt any of the ways. It would ask for root password if you started as a normal user.

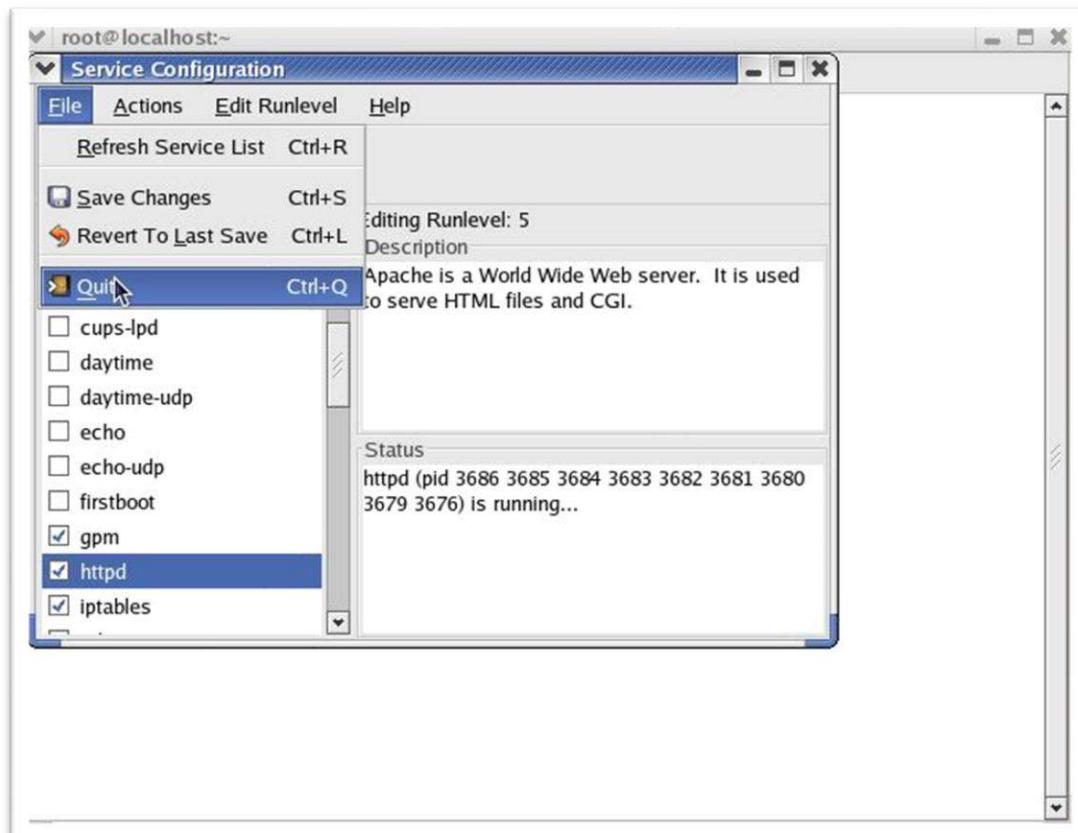
Service Configuration Dialog: Here we need to start the service.



Httpd need to be checked here.



When this is done. We need to save the changes and quit.



To control the Apache web server from the command line, we can use the service command to fire the httpdscript. Here's how we use it to start the web server:

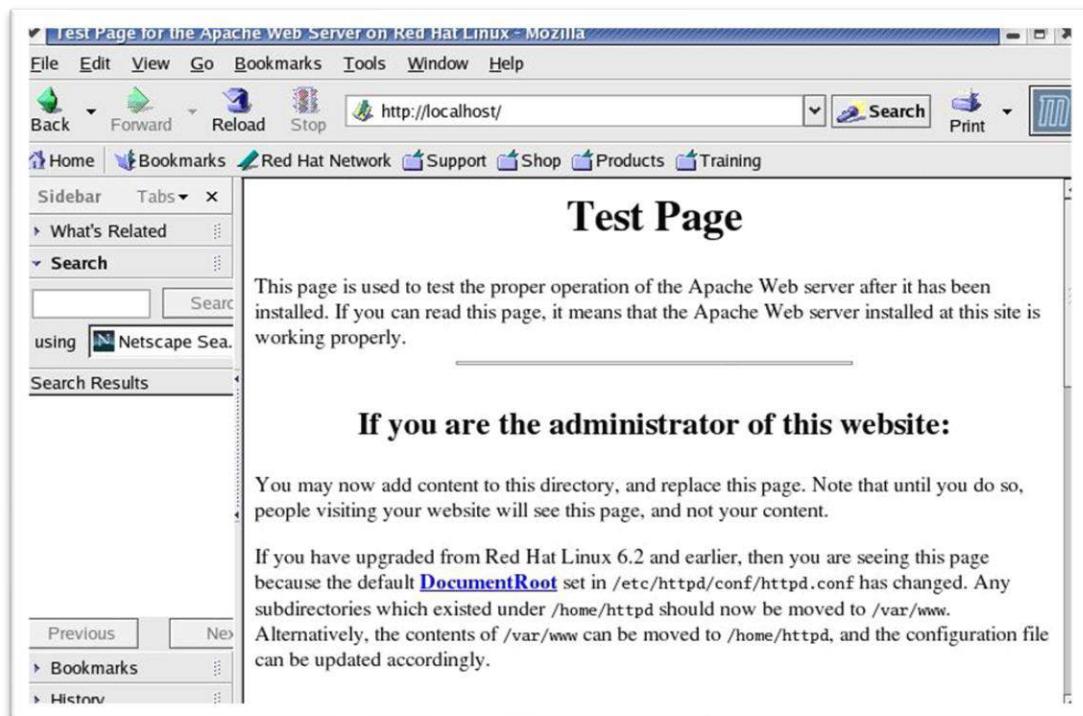
```
# service httpd start  
Starting httpd: [ OK ]
```

If there are difficulties in starting the web server, then you'll find out about it here. For example, if you attempt to do this without root privileges, then you'll get a message telling you that permission is denied. And here's another example:

```
# service httpd start  
Starting httpd: httpd: Could not determine the server's fully qualified  
domain name, using 192.168.0.99 for ServerName [ OK ]
```

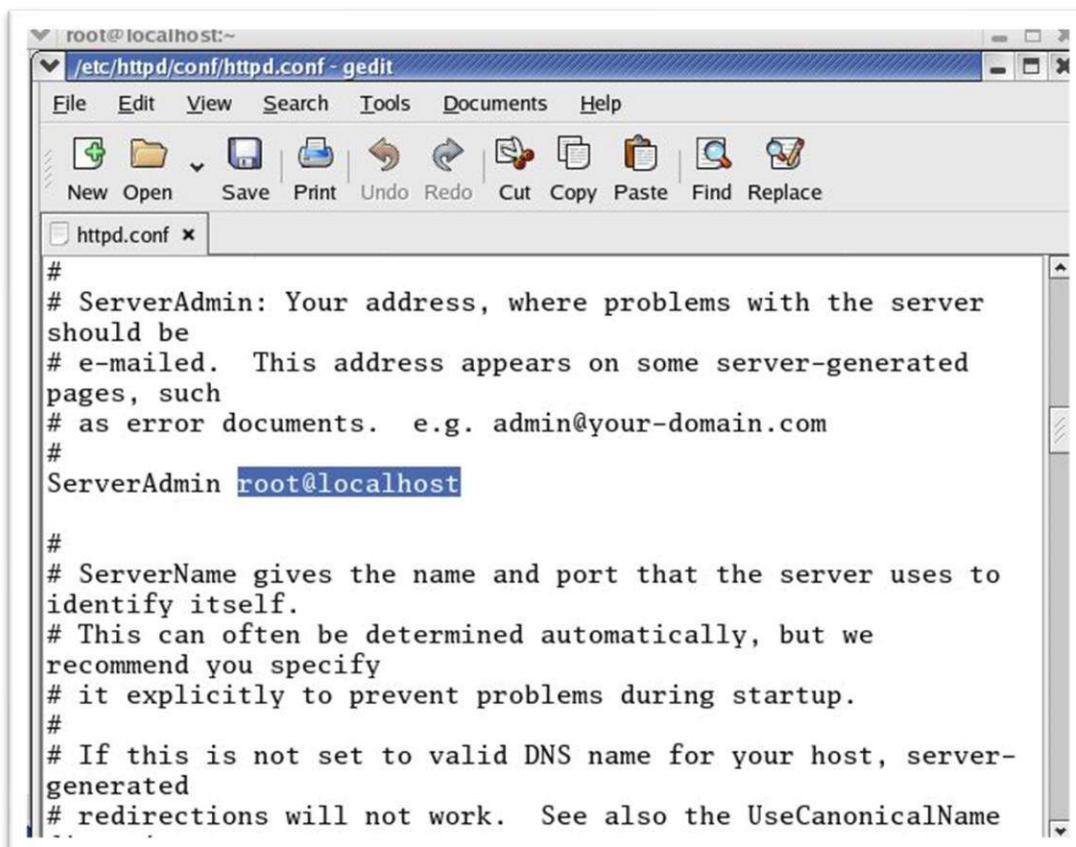
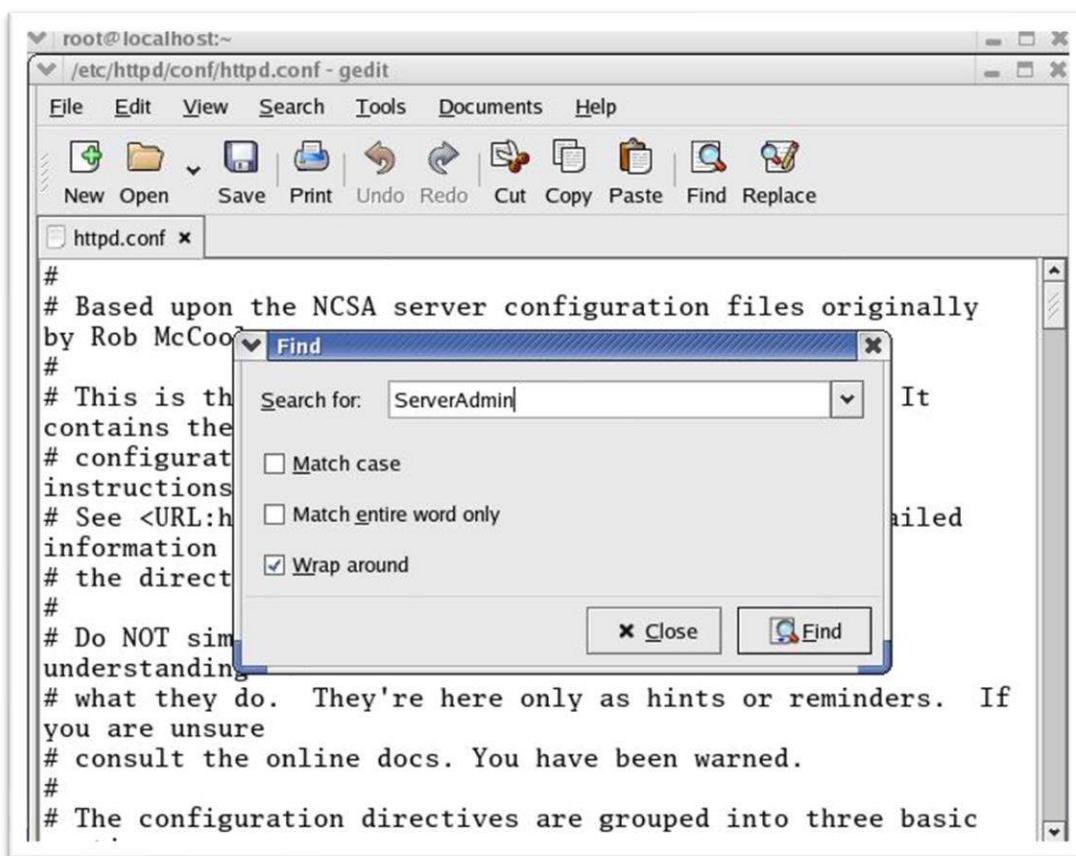
Testing the Apache Web Server

Once you've started the Apache web server, you should test it to see if it's working properly. To do that, we'll use a web browser to request a web page from our server!. There's a page provided by default for this purpose, and you can request it via the URL `http://localhost`. So, launch a web browser (Main Menu | Internet | Mozilla Web Browser), and type this URL into the address box:

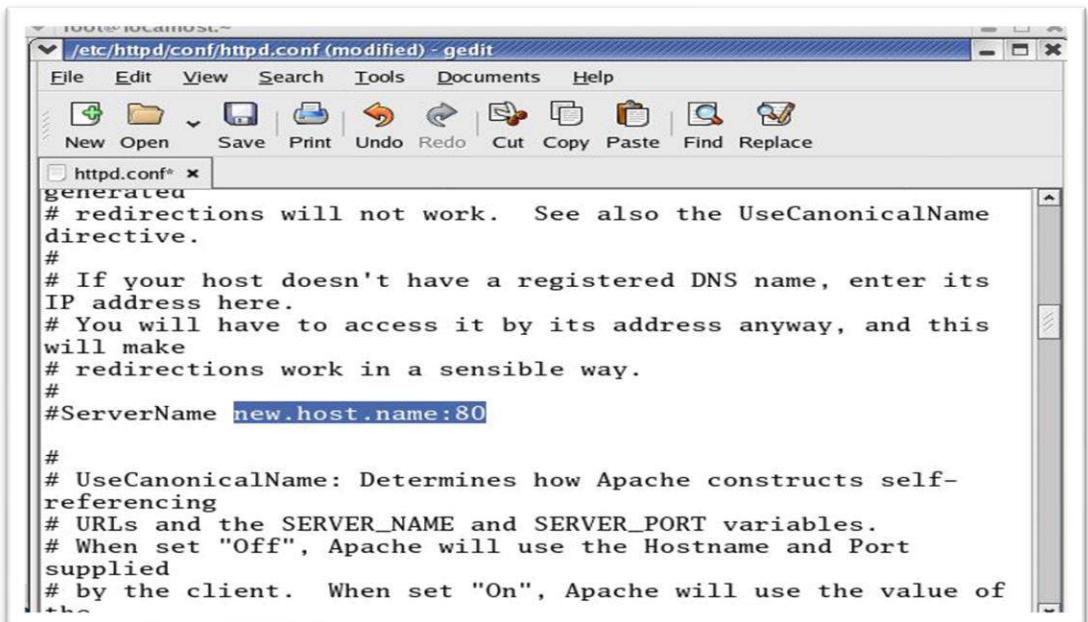
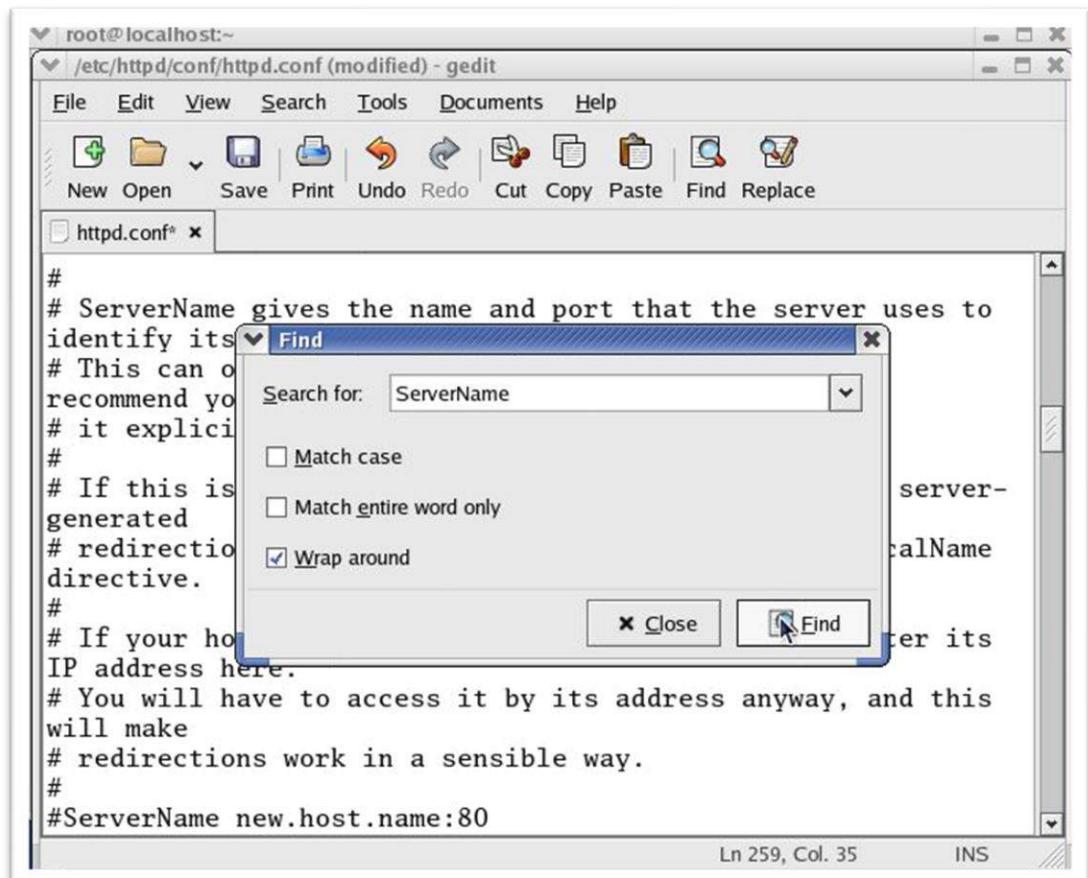


Configuring your Web Server

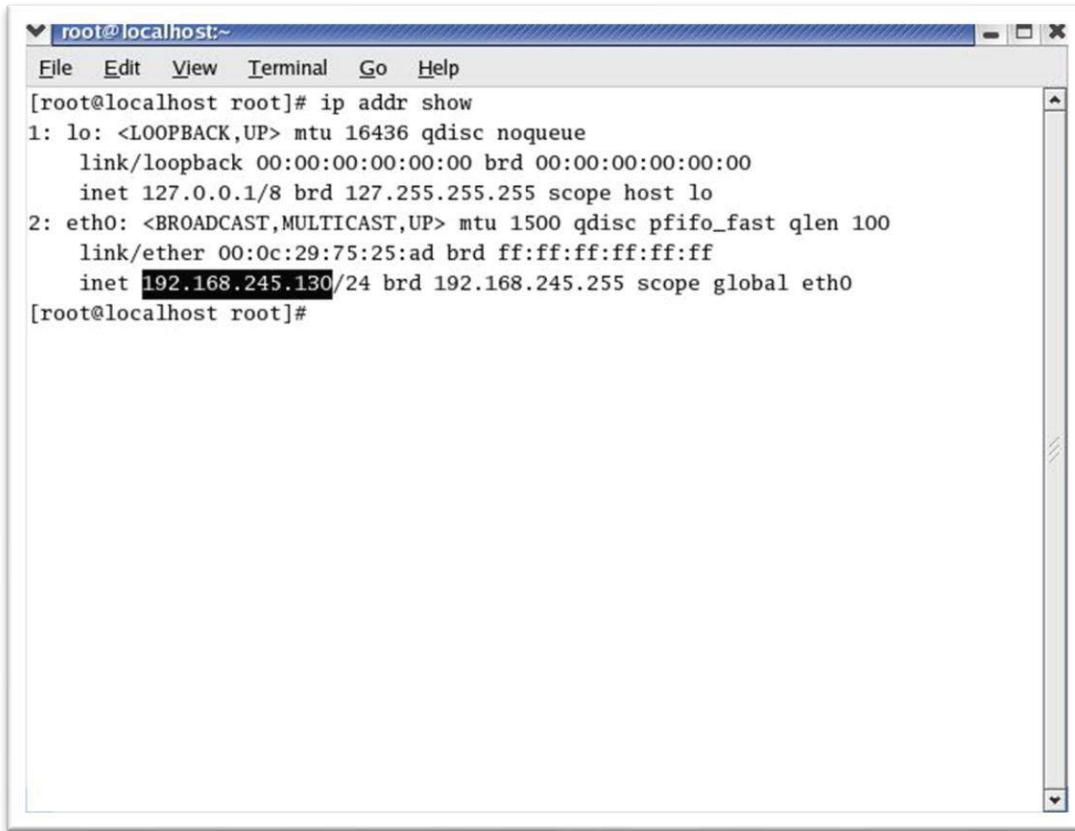
Launch the gedit text editor (by selecting Main Menu | Accessories | Text Editor). Use it to open the file `/etc/httpd/conf/httpd.conf`. Select Search | Find and use the resulting dialog to find the word Server Admin in the file. The first occurrence should be the Server Admin directive. Write your own email id here in the place of `root@localhost`.



Next thing we are going to search is ServerName. So in front of ServerName, new.host.name:80 is written. Here, we need to write the IP address of our machine.



By giving the command ip addr show, it will show us the ip address of the machine. So, in front of ServerName write the ip address of the machine.

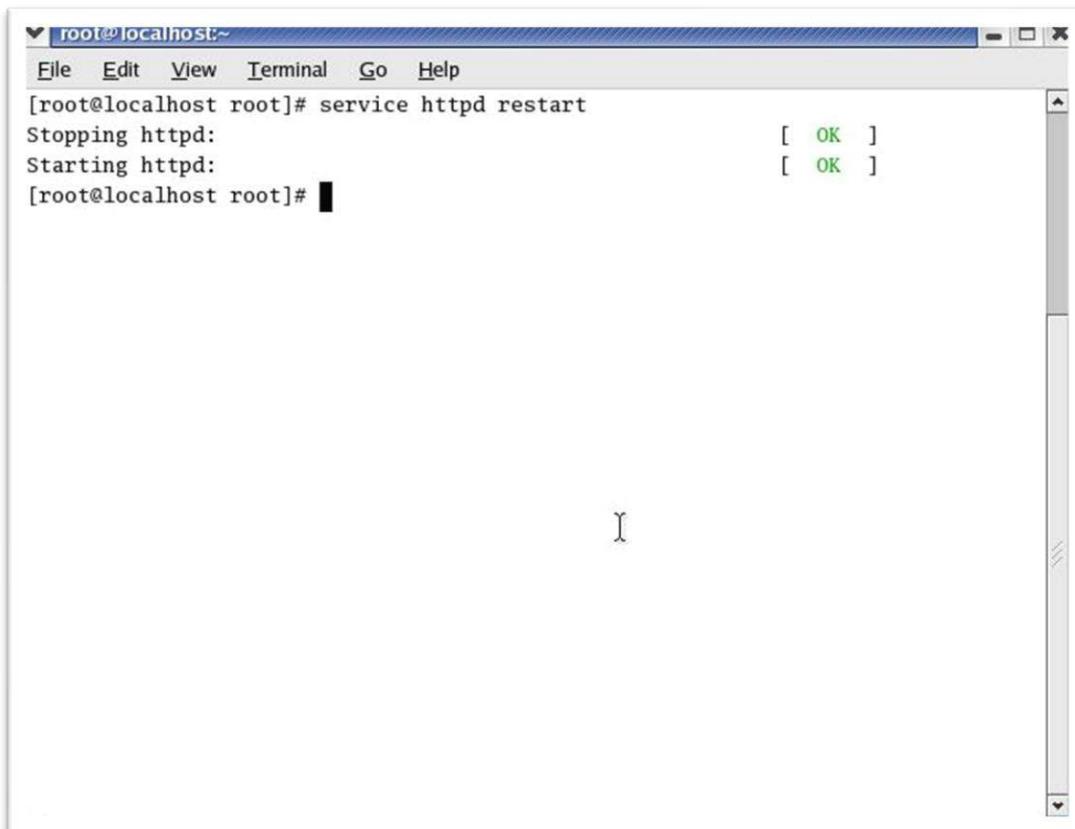


A screenshot of a terminal window titled "root@localhost:~". The window has a menu bar with File, Edit, View, Terminal, Go, and Help. The main area displays the output of the command "ip addr show". The output shows two interfaces: "lo" (loopback) and "eth0" (ethernet). The "lo" interface has an IP address of 127.0.0.1/8. The "eth0" interface has an IP address of 192.168.245.130/24.

```
[root@localhost root]# ip addr show
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:0c:29:75:25:ad brd ff:ff:ff:ff:ff:ff
    inet 192.168.245.130/24 brd 192.168.245.255 scope global eth0
[root@localhost root]#
```

Configuring your Web Server

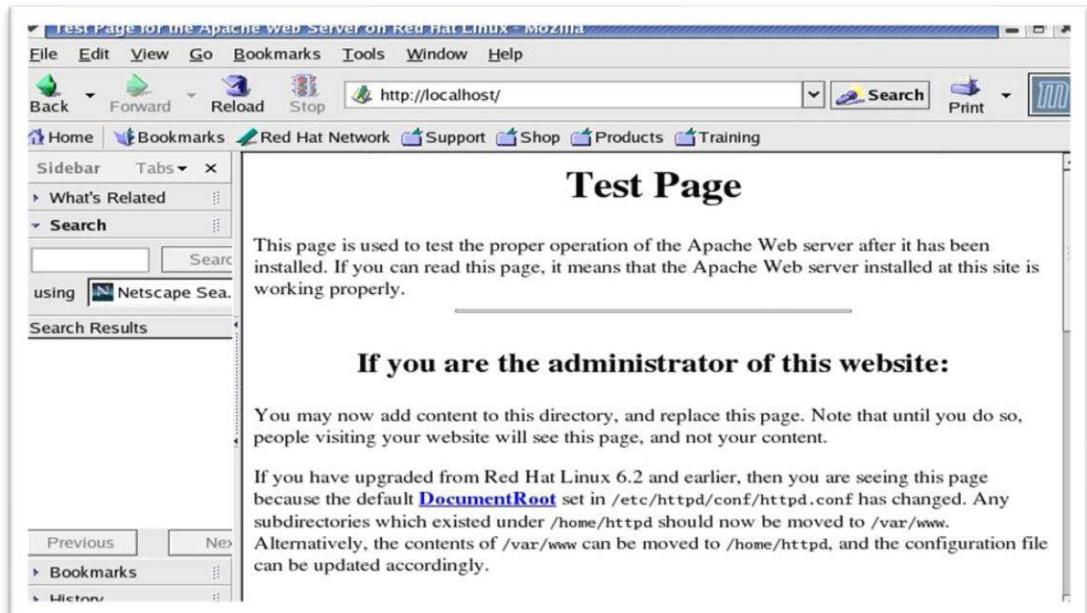
Again we need to restart the httpd service by writing service httpd restart.



A screenshot of a terminal window titled "root@localhost:~". The window has a menu bar with File, Edit, View, Terminal, Go, and Help. The main area displays the output of the command "service httpd restart". The output shows the service stopping and then starting successfully, indicated by green brackets [OK].

```
[root@localhost root]# service httpd restart
Stopping httpd:                                     [  OK  ]
Starting httpd:                                     [  OK  ]
[root@localhost root]#
```

Once this is done. Again open `http://localhost`

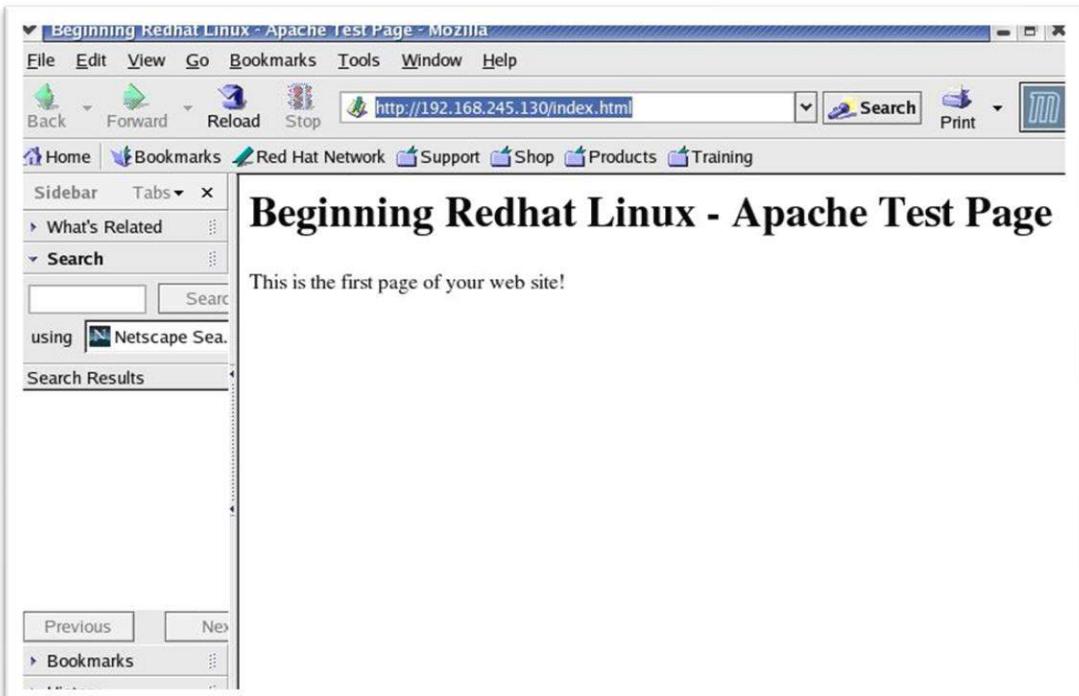
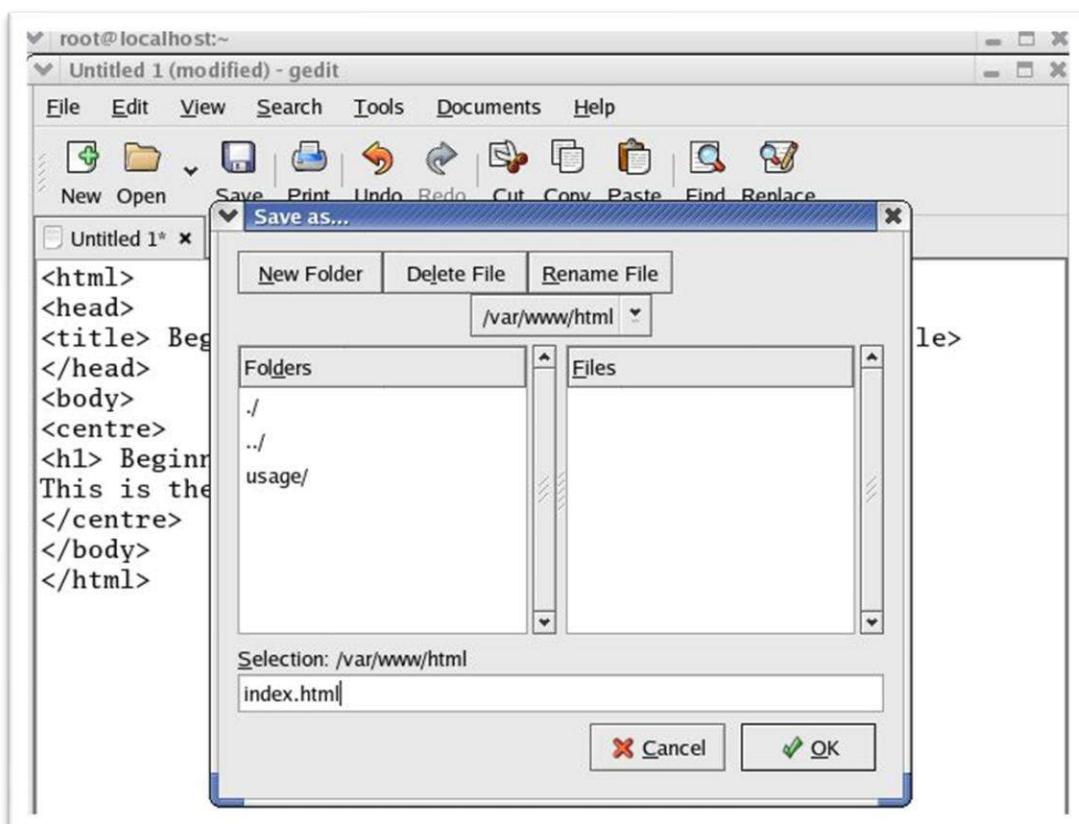


Setting up Your First Web Page

- This involves creating a simple HTML web page, and saving it to a location on the hard disk that is used by the web server to store published web pages.
- Then, when a user requests the page, the web server will be able to respond by retrieving it from this location and sending it to the requestor.
- Launch an editor (gedit).

A screenshot of a gedit text editor window titled "Untitled 1 (modified) - gedit". The title bar shows "root@localhost:~". The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar includes icons for New, Open, Save, Print, Undo, Redo, Cut, Copy, Paste, Find, and Replace. The main text area contains the following HTML code:

```
<html>
<head>
<title> Beginning Redhat Linux - Apache Test Page </title>
</head>
<body>
<centre>
<h1> Beginning Redhat Linux - Apache Test Page </h1>
This is the first page of your web site!
</centre>
</body>
</html>
```



Summary

- A web server is required When we want to publish web pages on the Internet or on an intranet.

- The standard protocol for internet is Hypertext Transfer Protocol (HTTP).
- All web servers listen for HTTP requests on port 80.
- Apache's popularity is due not only of its open source pedigree, but also to its highly competitive levels of performance, functionality, stability, flexibility, and security.
- There are two ways to start RPM's graphical interface: One is: Main Menu | System Settings | Add/Remove Applications and another is \$ redhat-config-packages.
- The important package httpd_manual contains the documentation for the Apache web server. After installation, you can access this documentation from the command line by typing man httpd.
- The /etc/httpd/httpd.conf file is Apache's main configuration file.
- A web server is an application that does two things: It listens for page requests and when it receives a page request, it examines the request and responds with the page that was requested.

Keywords

- **Web server:** web server performs is to display the content of the website, which it does by storing, then processing, and eventually delivering the webpages to the user who has requested it.
- **Static Web Server:** The static content is the one that is fixed. The static web server contains HTTP software and computer. This is static because the server sends hosted files as is present to the browser.
- **Dynamic Web Server:** The dynamic web browser will have the webserver and the software like the database and the applications server. This is dynamic because the application server is used to update the files hosted before these are sent to the browser.
- **Apache Web Server:** Apache is a modular server – the core server provides the basic functionality, with extended features available in various modules. This makes it very flexible and easy to configure, because you need to configure only the modules you need.
- **HTTP:** Hypertext Transfer Protocol (HTTP) is an application-layer protocol for transmitting hypermedia documents, such as HTML. It was designed for communication between web browsers and web servers, but it can also be used for other purposes.

Self Assessment

1. When we are talking about the web servers, how many packages are available?
 - A. 15
 - B. 16
 - C. 17
 - D. 18
2. Which package contains the documentation of httpd web server?
 - A. httpd-manual
 - B. hwcrypto
 - C. php
 - D. php-image

3. We can install the web server package if we have logged in as
 - A. Root
 - B. Normal user
 - C. Either root or normal user
 - D. None of the above

4. When we are installing web server, which service should be started from service configuration dialog?
 - A. echo
 - B. httpd
 - C. cups
 - D. None of the above

5. The configuration file is modified by searching
 - A. ServerAdmin
 - B. ServerName
 - C. Both of the above
 - D. None of the above

6. Where do we provide the ip address of machine when the modification of configuration file is done?
 - A. ServerAdmin
 - B. ServerName
 - C. Both of the above
 - D. None of the above

7. The important packages for Apache web server are
 - A. httpd-manual
 - B. mod_ssl
 - C. php
 - D. All of the above mentioned

8. Which command is used to check the ip address of the Linux machine?
 - A. ip addr show
 - B. internet address show
 - C. ip address show
 - D. None of the above mentioned

9. Why do we require web server on a Linux machine?
 - A. To test the site that is under development
 - B. To test a private website only available in private network
 - C. Both of the above
 - D. None of the above

10. With which way we can open RPM's graphical interface?
 - A. Main Menu | System Settings | Add/Remove Applications
 - B. \$ redhat-config-packages
 - C. Either of these mentioned
 - D. None of the above

11. HTTP stands for
 - A. Host text transfer protocol
 - B. Hypertext transfer protocol
 - C. High text transfer protocol
 - D. None of the above

12. The commercial web servers are
 - A. Zeus
 - B. Microsoft
 - C. SunOne
 - D. All of the mentioned

13. Which of these features belongs to Apache web server?
 - A. Portability
 - B. Scalability
 - C. Security
 - D. All of the above mentioned

14. What is the task of a web server?
 - A. Listens for a page request.
 - B. Examines and responds with the page requested
 - C. Both of the above
 - D. None of the above

15. Which one is the standard protocol of internet?
 - A. HTTP
 - B. PTTH
 - C. PTHT
 - D. None of the above

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. C | 2. A | 3. A | 4. B | 5. C |
| 6. B | 7. D | 8. A | 9. C | 10. C |
| 11. B | 12. D | 13. D | 14. C | 15. A |

Review Questions:

1. What is a web server? What is the need of installing web server on RedHat Linux machine? Give some examples of commercial and open source web servers.
2. What is Apache web server? Write some features of it in detail.
3. Write in detail how can we install Apache in RedHat Linux machine.
4. What are the important packages and configuration files of Apache web service. Give details.
5. How can we start and test Apache web server? Write in detail about its configuration.

**Further Readings**

Sandip Bhattacharya,Pancrazio De Mauro,Shishir Gundavaram,Mark Mamone,Kalip Sharma,Deepak Thomas, and Simon Whiting, Beginning Red Hat Linux 9, Wiley Publishing, Inc.

**Web Links**

<https://www.sumologic.com/blog/apache-web-server-introduction/>

Unit 12: File Server Configuration

CONTENTS

- Objectives
- Introduction
- 12.1 Installing the vsftpd FTP Server
- 12.2 Installing the vsftpd FTP Server
- 12.3 Starting your FTP Server
- 12.4 Testing Your FTP Server
- 12.5 Using an FTP Client to Test Anonymous Read Access
- 12.6 Configuring an Anonymous FTP Server for File Upload
- 12.7 Using an FTP Client to Test Anonymous Write Access
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to

- Understand the FTP protocol
- Understand the relevance and use of FTP
- How to start FTP server
- How to test FTP server
- How to use FTP client to test anonymous read access

Introduction

FTP stands for File Transfer Protocol. At its core, the file transfer protocol is a way to connect two computers to one another in the safest possible way to help transfer files between two or more points. To put it simply, it's the means by which files are securely shared between parties. FTP servers are the solutions used to facilitate file transfers across the internet. If you send files using FTP, files are either uploaded or downloaded to the FTP server. When you're uploading files, the files are transferred from a personal computer to the server. When you're downloaded files, the files are transferred from the server to your personal computer. TCP/IP (Transmission Control Protocol/Internet Protocol), or the language the internet uses to execute commands, is used to transfer files via FTP. So, the main points in consideration are:

- If you want to enable other users to download files from a location on your server's hard disk, and/or to upload files to that location, then one solution is to install an FTP server.
- You can think of an FTP server essentially as an area of disk space that is used for storing files, plus the software and configuration required to allow other users to upload and download files.

- When users want to upload or download from your FTP server, they use a program called an FTP client.

These communications between FTP server and FTP client take place using the File Transfer Protocol (FTP). FTP is a TCP protocol that is designed specifically for the transfer of files over a network, and it's one of the oldest Internet protocols still in widespread use.

Relevance of FTP

The availability of so many different FTP client programs, and the fact that many operating systems come with FTP software pre-installed, are indications of how relevant FTP still is today. FTP also is more efficient at large file transfers than HTTP and requires a password for access. FTP keeps a log of data transmission, where HTTP does not. All these components combined create a system that the common user might not notice, but which is incredibly important for professionals working with IT. FTP and its derivatives are the best choices when dealing with the manipulation of larger quantities of data. Rather than simply giving one-way access, FTP services allow users an enormous amount of control, and this can be extremely beneficial to individuals and business which update regularly.

Security of FTP

FTP is generally considered to be an insecure protocol because it relies on clear-text usernames and passwords for authentication and does not use encryption. Data sent via FTP is vulnerable to sniffing, spoofing, and brute force attacks, among other basic attack methods. It is not considered a secure protocol, because communication between the FTP client and server are unencrypted. Consequently, Secure FTP (SFTP) is also becoming popular (and, indeed, is part of the openssh package that comes with Red Hat Linux 9). It's also possible to configure your FTP server in other ways, for example by forcing users to log in, or by using access control lists (ACLs) to allow different rights to different groups of users.



Did you know?

The FTP was not built to be secure. The communication between the FTP client and server are unencrypted.

Anonymous FTP Access

- Anonymous FTP is called *anonymous* because you don't need to identify yourself before accessing files. In fact, many FTP servers still allow anonymous FTP access, which means that the FTP server allows any user to access its disk space and download its files.
- Anonymous FTP access is used mostly to enable users to access freely available documents and files via the Internet without access control.

Despite the security issues, FTP remains popular – it's fast and easy to use, and it is the Internet standard protocol for file transfer.

FTP Servers in the Red Hat Linux Distribution

There are few FTP servers available in Red Hat Linux distribution. These are:

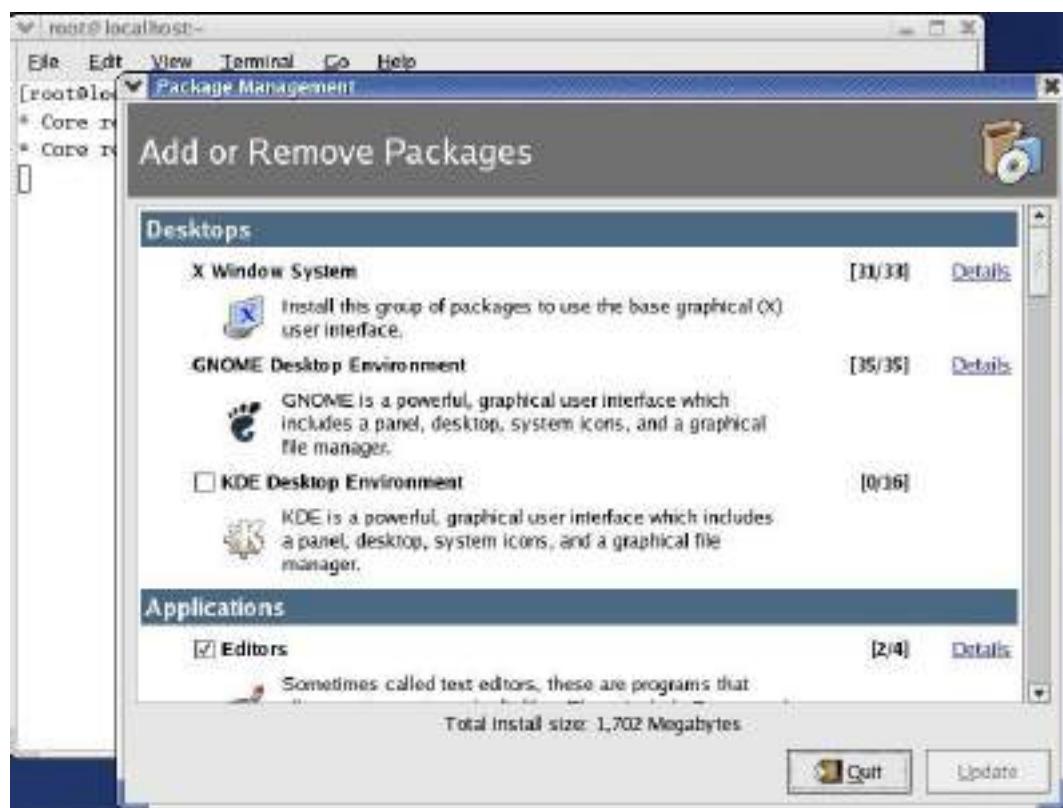
FTP Server	Remarks
vsftpd	It is a simplified FTP server implementation. It is designed to be a very secure FTP server and can also be configured to allow anonymous access.
TUX	It is a kernel-based, threaded, extremely high-performance HTTP server, which also has FTP capabilities. TUX is perhaps the best in terms of performance but offers less functionality than other FTP server software. TUX is installed by default with Red Hat Linux 9.

wu-ftp	It is a highly configurable and full-featured FTP daemon, which was popular in earlier versions of Red Hat Linux but has since given way to the more security-conscious vsftpd.
gssftpd	It is a kerberized FTP daemon, which means that it is suitable for use with the Kerberos authentication system.

12.1 Installing the vsftpd FTP Server

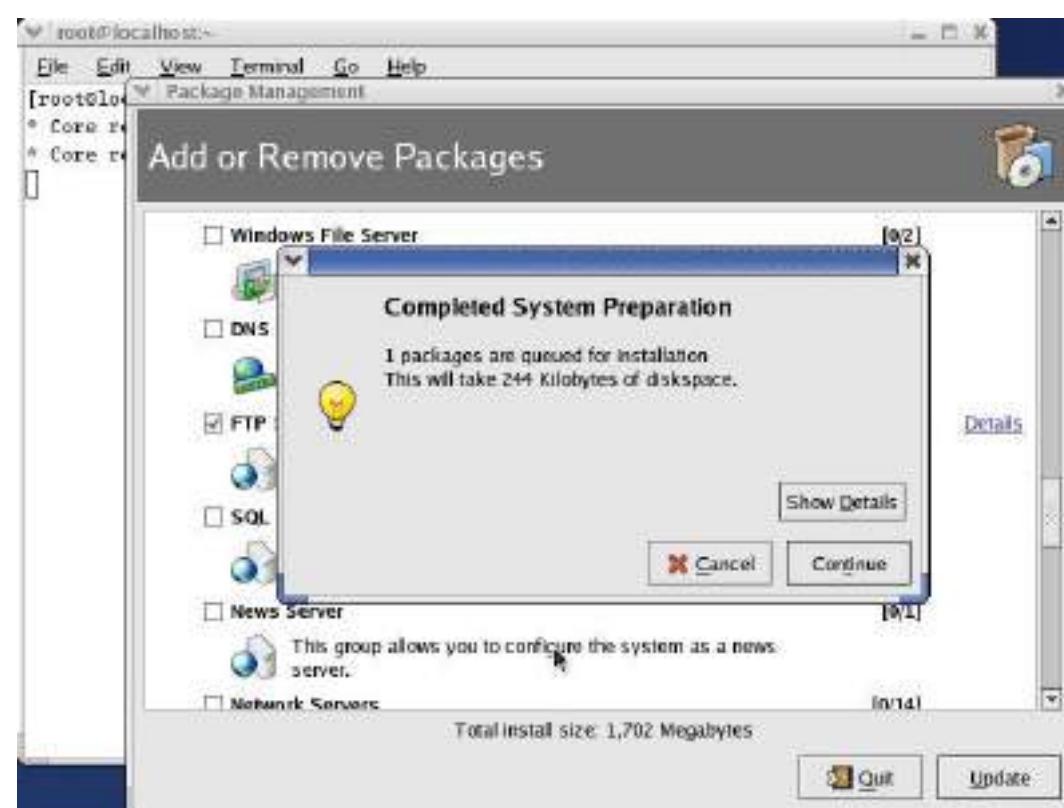
The easiest way to install the vsftpd FTP Server package is via the RPM GUI tool. There are two ways by which we can open the RPM GUI tool. These are:

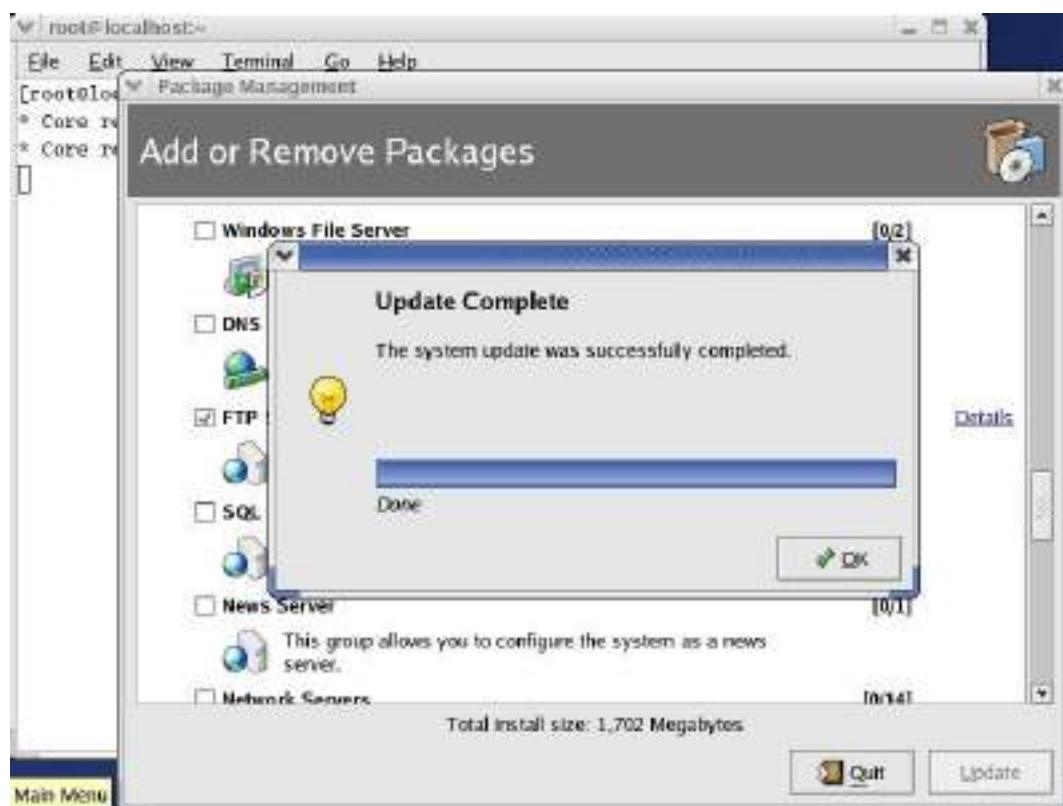
- Main Menu | System Settings | Add/Remove Applications
- \$ redhat-config-packages



12.2 Installing the vsftpd FTP Server

For installation of FTP server, we will see the category 'Servers'. In this we will click on FTP server. By clicking on details link, we can see the packages available in that. As in FTP server we have just one package. After clicking on update button, we can



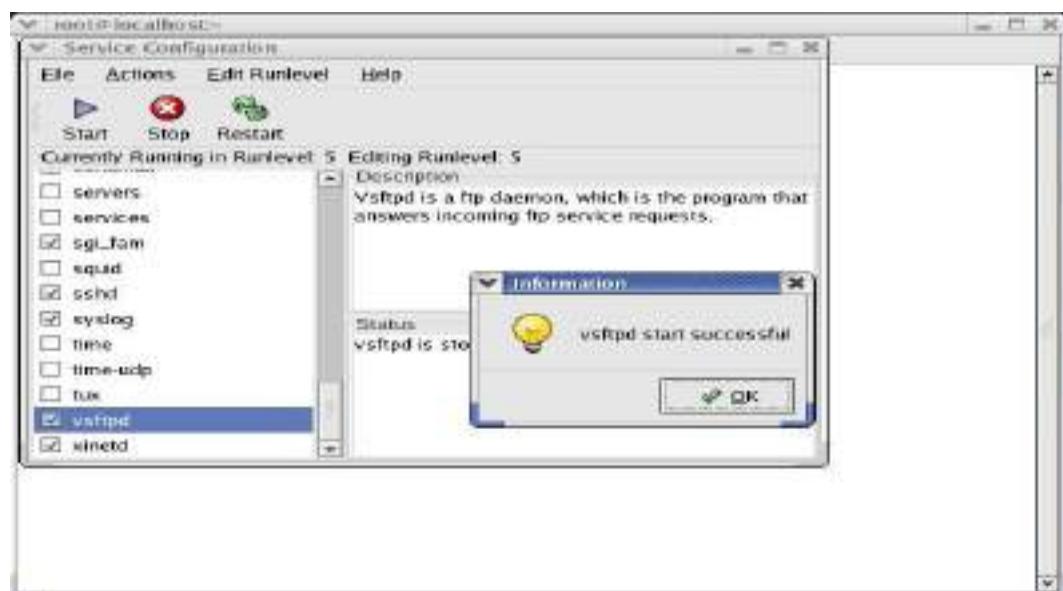


12.3 Starting your FTP Server

When the installation is done. We need to start the FTP Service, we can use the Service Configuration tool. There are again two ways to start FTP server. These are:

- Main Menu | System Settings | Server Settings | Services
- \$ redhat-config-services

Again, you'll be prompted for the root password, unless you're already logged on as root. After logging in as a root, it will open service configuration dialog.



Linux and Shell Scripting

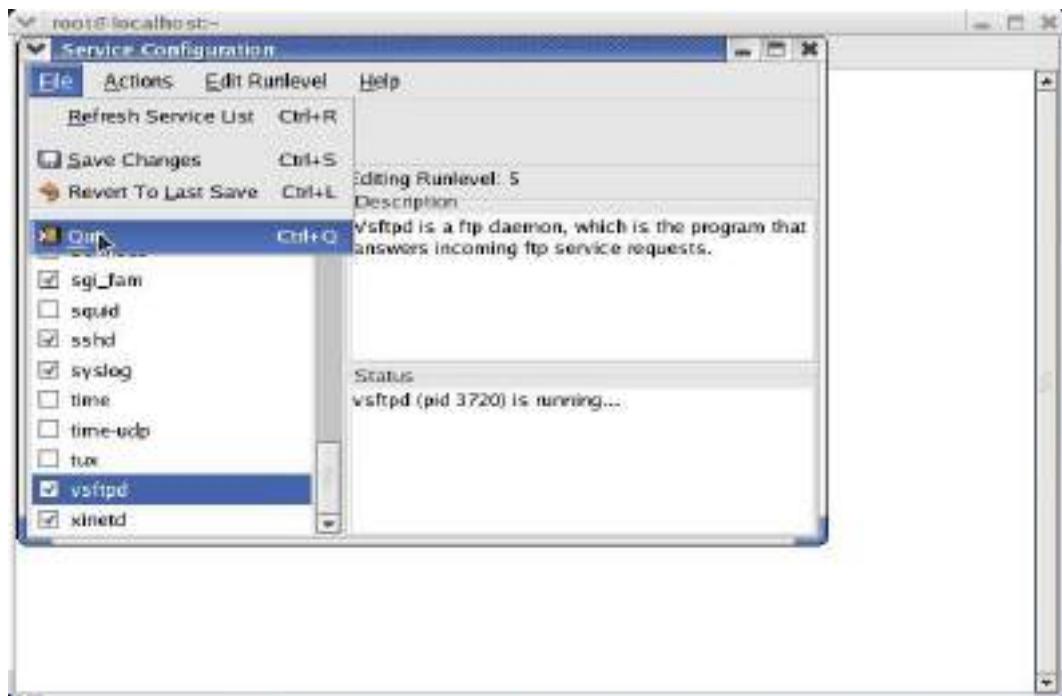
Here we need to start the service. By clicking on start button, it will start the service. After it, we need to save the changes and quit. It's also possible to start and stop these FTP services from the command line, using the service command to start and stop the vsftpd script:

```
# service vsftpd start
Starting vsftpd: [ OK ]
# service vsftpd stop
Stopping vsftpd: [ OK ]
```

Again, if you run the script without an option, the resulting usage message reveals all the available options:

```
# service vsftpd
```

Usage: vsftpd {start | stop | restart | condrestart | status}



12.4 Testing Your FTP Server

After setting up the FTP server, we need to test it. From the client side, we will test the server whether it is working fine or not. From a command line, issue the `ftp` command to start an FTP session, naming your FTP server as the server that you want to connect to. You should get a Name login prompt like the one shown above – this is enough to confirm to us that the vsftpd server is running. Press `Ctrl-C` to terminate this FTP session and return to the command line.

```
[root@localhost root]# ftp 192.168.245.129
Connected to 192.168.245.129 (192.168.245.129).
220 (vsFTPD 1.1.3)
Name (192.168.245.129:root):
```

Configuring an Anonymous FTP Server for File Download

Anonymous users cannot read from just any directory on your Linux server. By default, the vsftpd package creates a directory tree starting at /var/ftp, and enables 'anonymous read access' to this directory and the directory structure beneath it. For this we'll adopt the role of one of these users, and run a client FTP session to access the FTP server, examine the contents of the FTP site, and download a copy of the test file.

Setting up the FTP Server

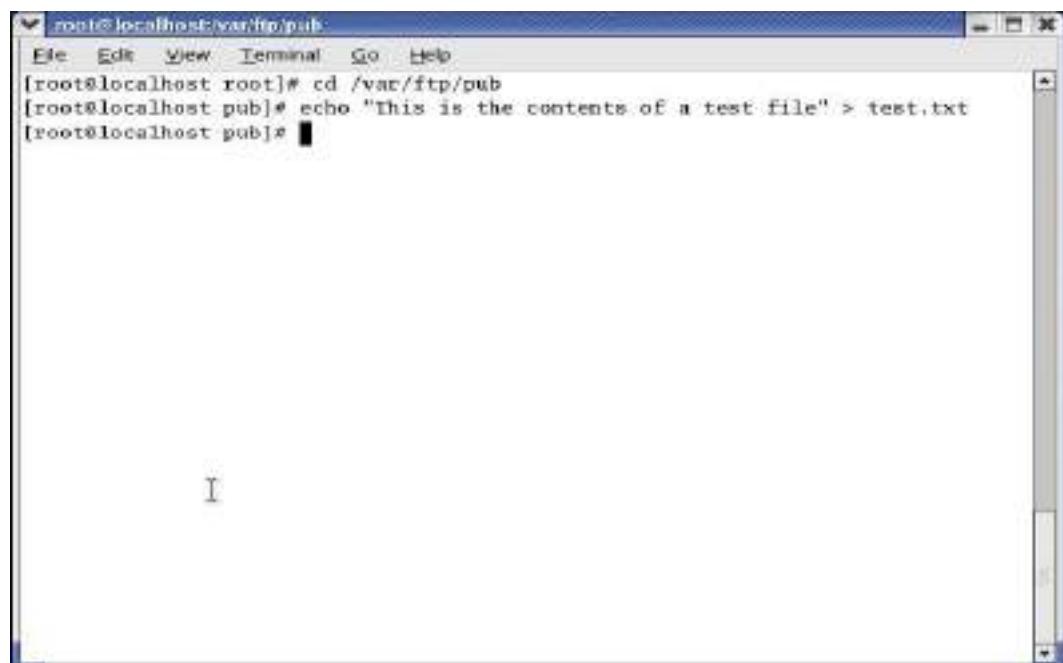
All we need to do here is place some test content somewhere under the /var/ftp directory, so that other users can access it. The owner of the /var/ftp is the root account, and by default is the only one with permission to write to the directory. For this we need to enter as a root user, so use a command line to switch to the root user:

```
$ su -
```

Password:

Then you can place whatever content you want under the /var/ftp directory. For example, you can easily use a command such as echo to create a simple test file:

```
# cd /var/ftp/pub
# echo "This is the contents of a test file!" > test.txt
```



The screenshot shows a terminal window titled 'root@localhost:/var/ftp/pub'. The window contains the following text:

```
File Edit View Terminal Go Help
[root@localhost root]# cd /var/ftp/pub
[root@localhost pub]# echo "This is the contents of a test file!" > test.txt
[root@localhost pub]#
```

12.5 Using an FTP Client to Test Anonymous Read Access

Now you can test for anonymous read access, by using an FTP client to try to grab a copy of this test file via an FTP connection. You can use any FTP client, and you can test from a Windows or Linux machine – provided the client machine can see the FTP server across a network. You can even use your Linux server as a client if you have only one machine. For example, in both Windows and Linux you can use the ftp program at the command line. In the following, we'll use the ftp program as FTP client to connect to the FTP server, examine the contents of the FTP site, and then download the file test.txt:

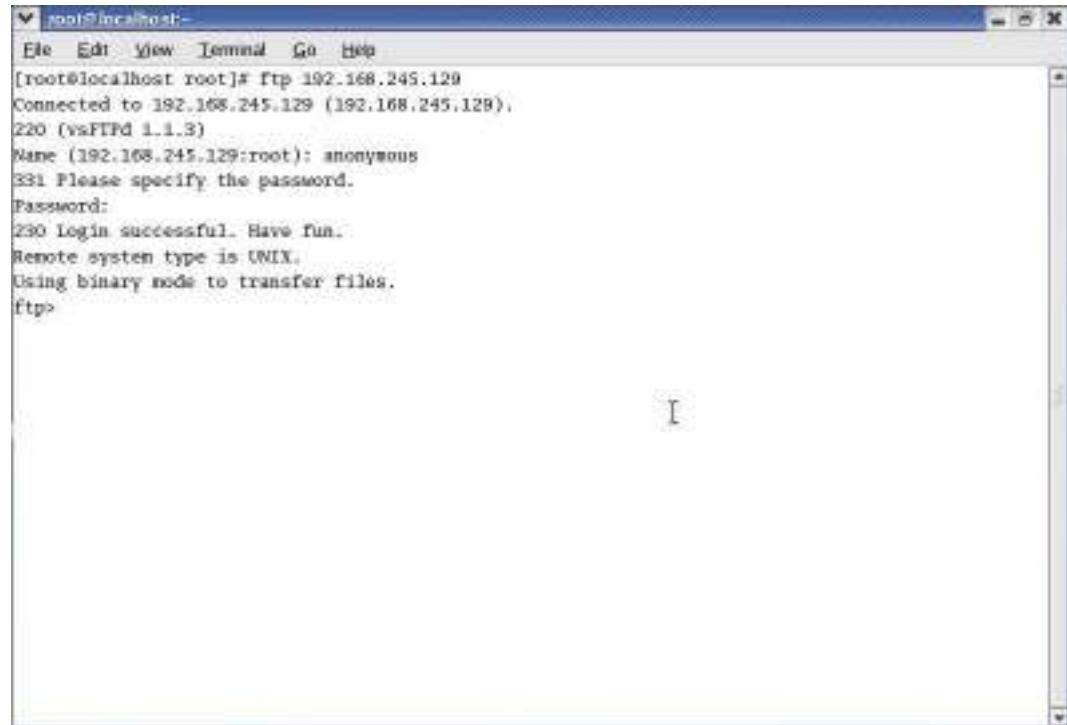
- 1) Start by connecting to the FTP server. When you're prompted for a username, specify anonymous (as shown below) or ftp to indicate that you want anonymous access:

```
$ ftp 192.168.245.129
Connected to 192.168.245.129 (192.168.245.129).
220 (vsFTPd 1.1.3)
```

Linux and Shell Scripting

```
Name (192.168.0.99:none): anonymous
331 Please specify the password.
Password:
230 Login successful. Have fun.
Remote system type is UNIX.

Using binary mode to transfer files .
```



A screenshot of a terminal window titled "root@localhost:~". The window shows an FTP session. The user has connected to the server at 192.168.245.129. The server is running vsFTPD 1.1.3. It prompts for a name (anonymous) and password, both of which are left blank. The server responds with a successful login message and states that the remote system type is UNIX. The user then types "Using binary mode to transfer files." and ends the command with a carriage return. The prompt "ftp>" is visible at the bottom.

2) Now, we can start to examine the contents of the FTP site that are available to users with anonymous access. For example, here we'll use the ls command to examine the contents of the FTP root directory which happens to be the directory /var/ftp on the server:

```
ftp>ls
220 Entering Passive Mode (192,168,245,129,29,204)
150 Here comes the directory listing.
drwxr--r--    2 0      0       4096   Apr13  15:38    pub
226 Directory send OK.
```

This shows that the root directory contains just one subdirectory, called pub. Now we'll use cd to change to this directory, and we'll list its contents.

Unit 12: File Server Configuration

```
[root@localhost root]# ftp 192.168.245.129
Connected to 192.168.245.129 (192.168.245.129).
220 (vsFTPd 1.1.3)
Name (192.168.245.129:root): anonymous
331 Please specify the password.
Password:
230 Login successful. Have fun.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
227 Entering Passive Mode (192.168.245.129,29,204)
150 Here comes the directory listing.
drwxr-xr-x 2 0 0 4096 Apr 13 15:38 pub
226 Directory send OK.
ftp>
```

```
[root@localhost root]# ftp 192.168.245.129
Connected to 192.168.245.129 (192.168.245.129).
220 (vsFTPd 1.1.3)
Name (192.168.245.129:root): anonymous
331 Please specify the password.
Password:
230 Login successful. Have fun.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
227 Entering Passive Mode (192.168.245.129,29,204)
150 Here comes the directory listing.
drwxr-xr-x 2 0 0 4096 Apr 13 15:38 pub
226 Directory send OK.
ftp> cd pub
250 Directory successfully changed.
ftp> ls -al
227 Entering Passive Mode (192.168.245.129,96,213)
150 Here comes the directory listing.
drwxr-xr-x 2 0 0 4096 Apr 13 15:38 .
drwxr-xr-x 3 0 0 4096 Apr 13 14:36 ..
-rw-r--r-- 1 0 0 96 Apr 13 15:38 test.txt
226 Directory send OK.
ftp>
```

3) Now, we'll attempt to download the test.txt file we've just located. To do this, we'll use the get command:

```
ftp> get test.txt
local: test.txt remote: test.txt
227 Entering Passive Mode (192, 168, 245, 129, 33, 9)
150 Opening BINARY mode data connection for test.txt (36 bytes).

226 File send OK.
```

Linux and Shell Scripting

```

root@localhost:~#
File Edit View Terminal Go Help
331 Please specify the password.
Password:
230 Login successful. Have fun.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
227 Entering Passive Mode (192,168,245,129,29,204)
150 Here comes the directory listing.
drwxr-xr-x 2 0 0 4096 Apr 13 15:38 pub
226 Directory send OK.
ftp> cd pub
250 Directory successfully changed.
ftp> ls -al
227 Entering Passive Mode (192,168,245,129,96,213)
150 Here comes the directory listing.
drwxr-xr-x 2 0 0 4096 Apr 13 15:38 .
drwxr-xr-x 3 0 0 4096 Apr 13 14:36 ..
-rw-r--r-- 1 0 0 36 Apr 13 15:38 test.txt
226 Directory send OK.
ftp> get test.txt
local: test.txt remote: test.txt
227 Entering Passive Mode (192,168,245,129,96,9)
150 Opening BINARY mode data connection for test.txt (36 bytes).
226 File send OK.
36 bytes received in 2.4e-05 secs (1.5e+03 Kbytes/sec)
ftp>

```

4) Finally, we'll end the session:

```

ftp> bye
221 Goodbye.
$
```

```

root@localhost:~#
File Edit View Terminal Go Help
230 Login successful. Have fun.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
227 Entering Passive Mode (192,168,245,129,29,204)
150 Here comes the directory listing.
drwxr-xr-x 2 0 0 4096 Apr 13 15:38 pub
226 Directory send OK.
ftp> cd pub
250 Directory successfully changed.
ftp> ls -al
227 Entering Passive Mode (192,168,245,129,96,213)
150 Here comes the directory listing.
drwxr-xr-x 2 0 0 4096 Apr 13 15:38 .
drwxr-xr-x 3 0 0 4096 Apr 13 14:36 ..
-rw-r--r-- 1 0 0 36 Apr 13 15:38 test.txt
226 Directory send OK.
ftp> get test.txt
local: test.txt remote: test.txt
227 Entering Passive Mode (192,168,245,129,96,9)
150 Opening BINARY mode data connection for test.txt (36 bytes).
226 File send OK.
36 bytes received in 2.4e-05 secs (1.5e+03 Kbytes/sec)
ftp> bye
221 Goodbye.
[root@localhost root]#

```

12.6 Configuring an Anonymous FTP Server for File Upload

Anonymous FTP users can write only to the directories that we allow them to write to. By default, vsftpd does not allow users to upload to the FTP server at all; we must first configure the server to allow anonymous users write access to some directory. So, we'll set up the FTP server for anonymous write access first; then we'll test it again using an FTP client.

Setting up the FTP Server for Anonymous Write Access

Unit 12: File Server Configuration

There are four steps here. We'll need to create the folder, set the appropriate permissions, and then enable uploading in the FTP server configuration:

1. First, we need to create a writeable directory. Again, you'll need the root account for this.

Let's create a directory called /upload (in the /var/ftp/pub directory):

```
# cd /var/ftp/pub
# mkdir upload
```

2. Next, we need to set the permission of the upload directory so that it allows write-only access to anonymous FTP users (so that they can write to the directory but not download from it – this restricts file sharing among FTP users). To do this, we'll first use the chgrp command to change the group associated with the upload directory:

```
# chgrp ftp upload
```

Now, the owner of the folder is still root, but the directory's group is ftp – the set of FTP users. Now we'll use the chmod command to assign read/write/execute access to the owner, write/access only to the group, and deny access to other users:

```
# chmod -R u=rwx, g=rx, o=rxw upload
```

3. Finally, we must configure the vsftpd server to allow anonymous upload. To do this, we simply edit the configuration file, /etc/vsftpd/vsftpd.conf. Open this file using gedit (or your favorite text editor), and locate the following lines:

```
# Uncomment this to allow the anonymous FTP user to upload files. This only
# has an effect if the above global write enable is activated. Also, you will
# obviously need to create a directory writable by the FTP user.
#anon_upload_enable=YES
```

Just remove the leading # character in the last line, and save the file:
anon_upload_enable=YES

4. Finally, restart the vsftpd service by using the Restart button in the Server Configuration dialog, or typing the following at the command line:

```
# service vsftpd restart
```

12.7 Using an FTP Client to Test Anonymous Write Access

So, let's test our configuration with another simple session on our FTP client:

1. Connect to the client and log in (using the username anonymous or ftp) as you did before:

```
$ ftp 192.168.0.99
```

```
Connected to 192.168.0.99 (192.168.0.99).
```

```
220 (vsFTPd 1.1.3)
```

```
Name (192.168.0.99:none): anonymous
```

```
331 Please specify the password.
```

```
Password:
```

```
230 Login successful. Have fun.
```

```
Remote system type is UNIX.
```

```
Using binary mode to transfer files.
```

2. Change directory to the pub/upload directory. Try to list its contents – you'll find that you can't,

because that's the way we configured the permissions on the upload directory:

```
ftp> cd /pub/upload
```

```
250 Directory successfully changed.
```

```
ftp> ls
```

```
227 Entering Passive Mode (192, 168, 0, 99, 95, 148)
```

```
150 Here comes the directory listing.
```

```
226 Transfer done (but failed to open directory).
```

3. However, you can upload a file. To prove it, use the put command to upload a simple file like this:

```
ftp> put uploadtest.txt
local: uploadtest.txt remote: uploadtest.txt
227 Entering Passive Mode (192,168,0,99,133,229)
150 Ok to send data.
226 File receive OK.
40 bytes send in 0.000101 secs (2.1e+02 Kbytes/sec)
```

4. That's it. Now you can close the FTP session:

```
ftp> bye
221 Goodbye.
#
#
```

Summary

- If you want to enable other users to download files from a location on your server's hard disk, and/or to upload files to that location, then one solution is to install an FTP server.
- FTP is not considered a secure protocol, because communication between the FTP client and server are unencrypted.
- The easiest way to install the vsftpd FTP Server package is via the RPM GUI tool. One way is: Main Menu | System Settings | Add/Remove Applications and another is \$ redhat-config-packages.
- Press Ctrl-C to terminate this FTP session and return to the command line.
- FTP is a TCP protocol that is designed specifically for the transfer of files over a network, and it's one of the oldest Internet protocols still in widespread use.

Keywords

- **FTP Client:** When users want to upload or download from your FTP server, they use a program called an FTP client.
- **File Transfer Protocol:** These communications between FTP server and FTP client take place using the File Transfer Protocol (FTP).
- **vsftpd:** It is a simplified FTP server implementation. It is designed to be a very secure FTP server and can also be configured to allow anonymous access.
- **FTP Server:** FTP servers are the solutions used to facilitate file transfers across the internet. If you send files using FTP, files are either uploaded or downloaded to the FTP server.
- **TUX:** It is a kernel-based, threaded, extremely high-performance HTTP server, which also has FTP capabilities. TUX is perhaps the best in terms of performance but offers less functionality than other FTP server software. TUX is installed by default with Red Hat Linux 9.

Self Assessment

1. FTP is
 - A. Easy to use
 - B. Free
 - C. Internet standard protocol for file transfer
 - D. All of the above mentioned

2. How can we open the RPM's GUI tool?
 - A. Main Menu | System Settings | Add/ Remove Applications
 - B. \$ redhat-config-packages
 - C. By using either of these ways
 - D. None of the above

3. FTP stands for
 - A. File transfer protocol
 - B. First transfer protocol
 - C. First temperature protocol
 - D. None of the above

4. Who is the owner of /var/ftp?
 - A. All normal users
 - B. Only root
 - C. Owner can be anyone
 - D. None of the above

5. Who can install FTP in the system?
 - A. Only root
 - B. Normal user
 - C. Any of the above
 - D. None of the above

6. Why FTP client program is used?
 - A. To upload the files to FTP server
 - B. To download the files from FTP server
 - C. Both upload and download
 - D. None of the above

7. Which utility is used to change the directories?
 - A. cd
 - B. changedir
 - C. chdirectory
 - D. None of the above

8. Which key combination is used to terminate the FTP session?
 - A. CTRL-A
 - B. CTRL-B
 - C. CTRL-C
 - D. CTRL-D

9. Why FTP is not considered secure?
 - A. Communications are fake
 - B. Communications are unencrypted

- C. Communications are available to everyone
 - D. None of the above
10. FTP server can be considered as
- A. Area of disk space used for storing files
 - B. Software to allow access
 - C. Configuration files to allow access
 - D. All of the above mentioned
11. FTP is a
- A. TCP protocol
 - B. HTTP protocol
 - C. SMTP protocol
 - D. None of the above
12. SFTP stands for
- A. Second file transfer protocol
 - B. Secure file transfer protocol
 - C. Steamed file transfer protocol
 - D. None of the above
13. In FTP server, the software and configuration files are required for
- A. Giving the user access for download
 - B. Giving the user access for upload
 - C. Both mentioned above
 - D. None of the above
14. What indicates the relevance of FTP today?
- A. Availability of different FTP client programs
 - B. Many OS come with FTP preinstalled
 - C. Both above mentioned
 - D. None of the above
15. Which of these are FTP servers?
- A. vsftpd
 - B. TUX
 - C. Both of the above
 - D. None of the above

Answers for Self Assessment

- | | | | | |
|------|------|------|------|-------|
| 1. D | 2. C | 3. A | 4. B | 5. A |
| 6. C | 7. A | 8. C | 9. B | 10. D |

11. A 12. B 13. C 14. C 15. C

Review Questions

1. What is a FTP server? Write the FTP servers available in Red Hat Linux Distribution.
2. Write the installation procedure of vsftpd FTP server. How can we start the FTP server?
3. After installation and testing of FTP server, how can we use it? Explain various issues encountered in this.
4. How can we configure anonymous FTP server for file download?
5. How can we configure anonymous FTP server for file upload?



Further Readings

Sandip Bhattacharya, Pancrazio De Mauro, Shishir Gundavaram, Mark Mamone, Kalip Sharma, Deepak Thomas and Simon Whiting, Beginning Red Hat Linux 9, Wiley Publishing, Inc.



Web Links

<https://www.techopedia.com/definition/26108/ftp-server>

Unit 13: Samba Servers

CONTENTS

- Objectives
- Introduction
- 13.1 Installing SAMBA
- 13.2 Starting and Stopping the Samba Service
- 13.3 Samba Configuration Files and Utilities
- 13.4 Samba Configuration with SWAT
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to

- Understand the SAMBA server
- Installing, starting and stopping the server
- SAMBA configuration with SWAT, starting SWAT service
- Adding SAMBA user
- Creating and configuring SAMBA share

Introduction

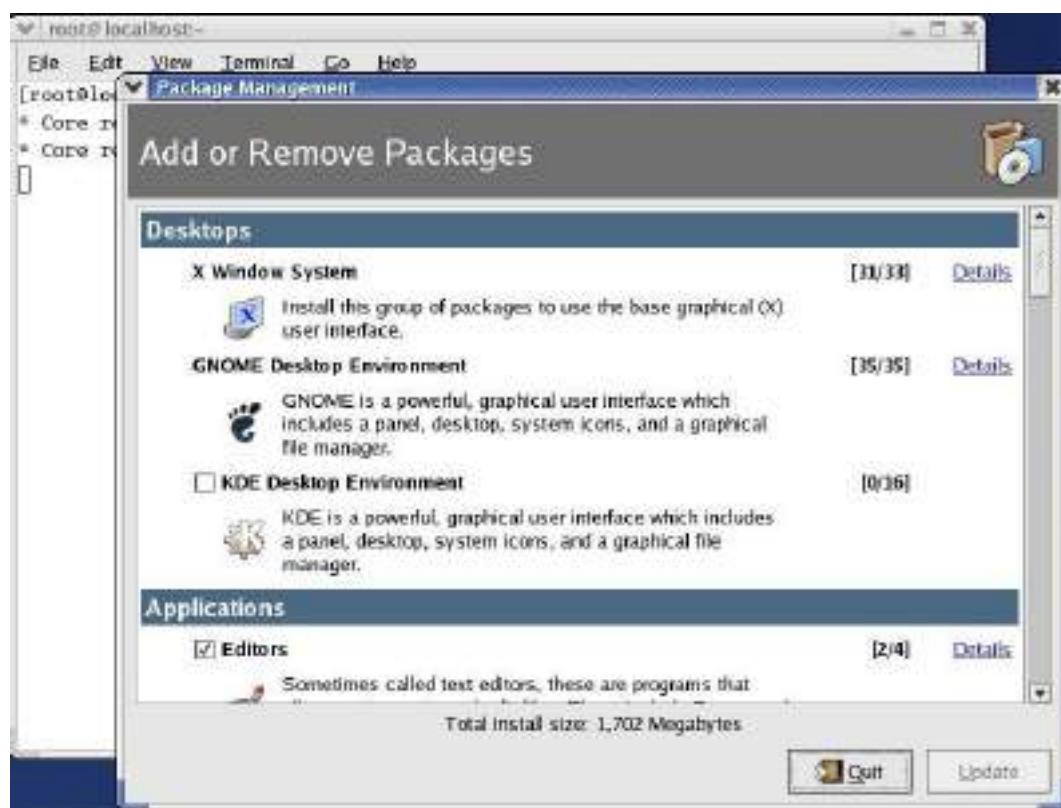
Samba is an implementation of the Windows SMB and CIFS protocols on Unix. The Samba project was started by Andrew Tridgell, who wanted to mount his Unix server disk space onto a DOS PC. When he'd solved the problem, Tridgell discovered that what he'd built was an implementation of the SMB (servermessage block) protocol – a protocol for sharing files and other resources. Tridgell named his implementation

Samba, and published version 1.0 in early 1992. Since that time, the Samba project has grown tremendously, and today there is still Samba development going on in the open-source community. So, Samba is a collection of programs that make it possible to share files and printers between computers equipped to use the SMB protocol – Windows by default, Linux/Unix with Samba, and (more recently) MacOS X. Samba is freely available under GNU General Public License and is included as part of the Red HatLinux 9 distribution.

13.1 Installing SAMBA

The installation of SAMBA Server suite can be done via the RPM GUI tool. One of the way of this is Main Menu | System Settings | Add/Remove Applications and another way is by using the terminal and writing the command \$ redhat-config-packages.

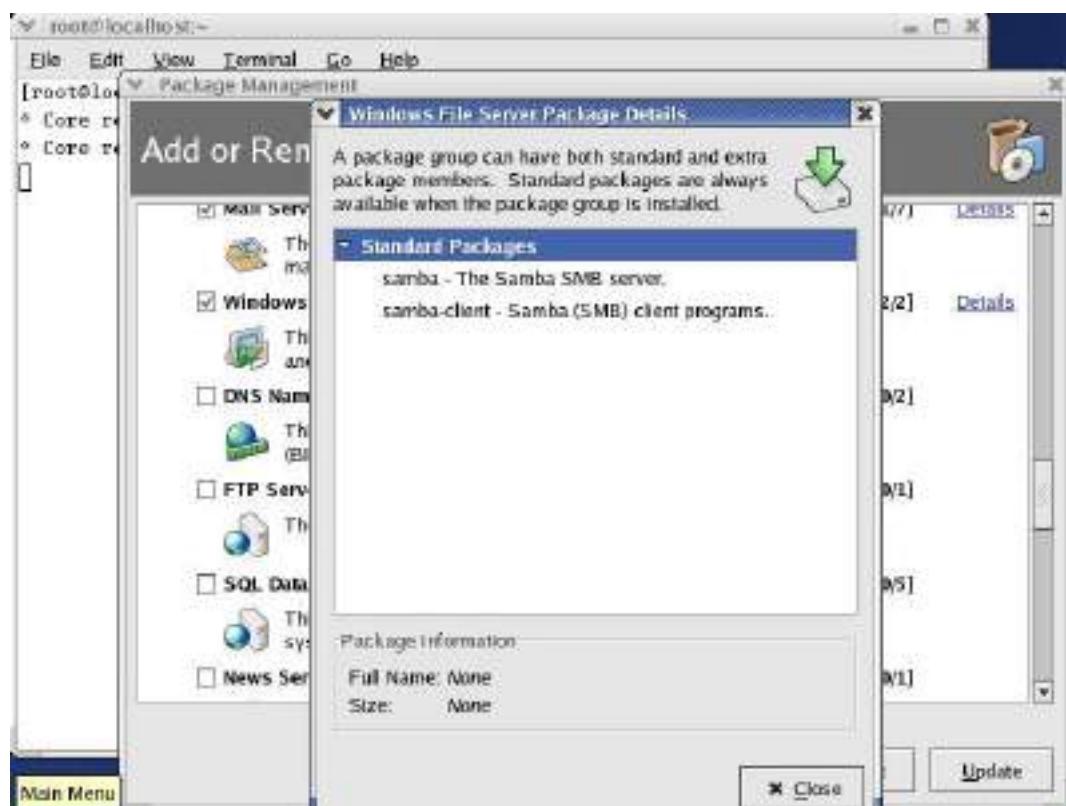
Linux and Shell Scripting



In package management window, we will look for the servers category. In servers, we will go for Windows File Server.



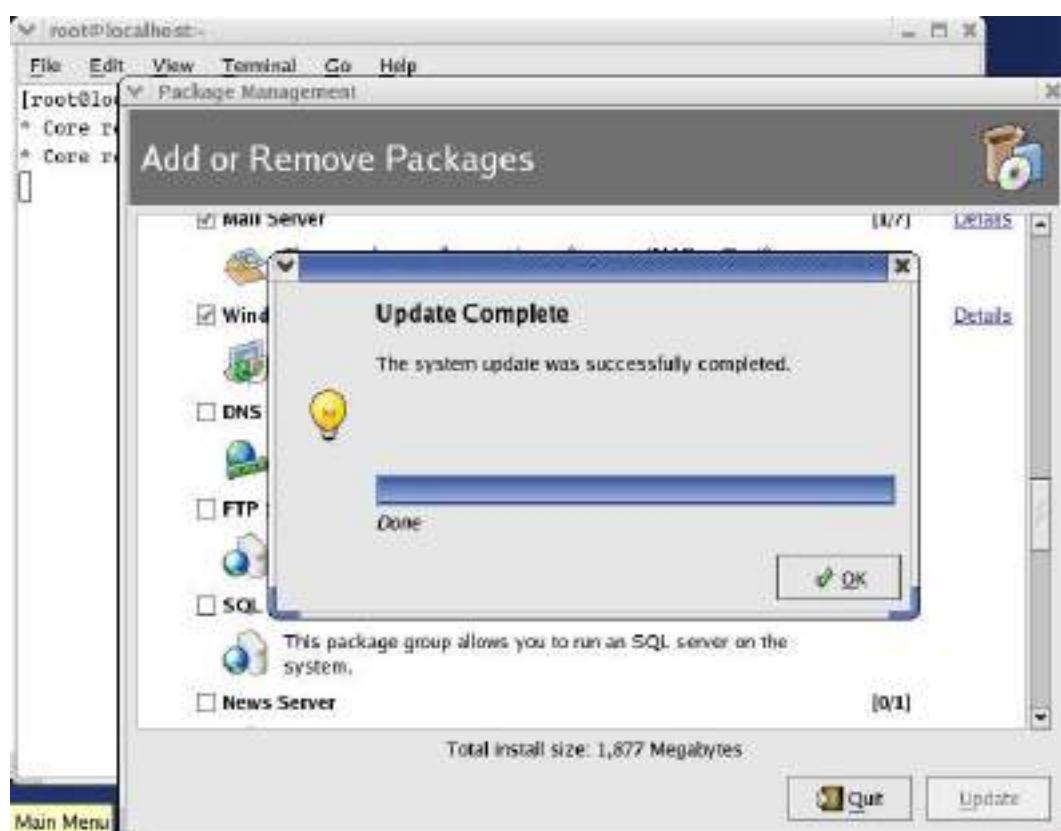
By clicking on the details, we can see the packages available in this. There are two standard packages available in this. These are: samba which is the Samba SMB server and samba-client which is SMB client programs.



After clicking on update button, the packages will be installed. The popup will show the required diskspace. After clicking on continue, it will queue the packages for installation.



After few seconds, the update will complete. Click on OK.



13.2 Starting and Stopping the Samba Service

There are several ways to start and stop the Samba service. Once again, we can do so via the Service Configuration GUI tool. To launch the tool, select Main Menu | System Settings | Server Settings | Services or type the following command at the command line: \$ redhat-config-services. Click on smb in this. Now the service is stopped. We need to start it by clicking on the start button.



It's also a good idea to check the checkbox, to configure the samba service to start automatically whenever you boot up the system. For example, if you ever have to perform an emergency reboot on your file server, then the "automatic start" configuration means that the file server is immediately available to users after the reboot. When you've done this, select File | Save Changes to save your new setting.

Alternatively, you can also stop and start smb service at the command line, using the service command to run the /etc/rc.d/init.d/smb script we mentioned earlier. Typing the script name at the command line like this reveals the possible usages:

```
# servicesmb
```

```
Usage: /etc/rc.d/smb {start|stop|restart|reload|status|condrestart}
```

As you can see, it works in much the same way as the httpd and vsftpd scripts we've seen in earlier sections of this chapter. So, to start the service we'd type this:

```
# servicesmb start
```

Starting SMB services:	[OK]
------------------------	--------

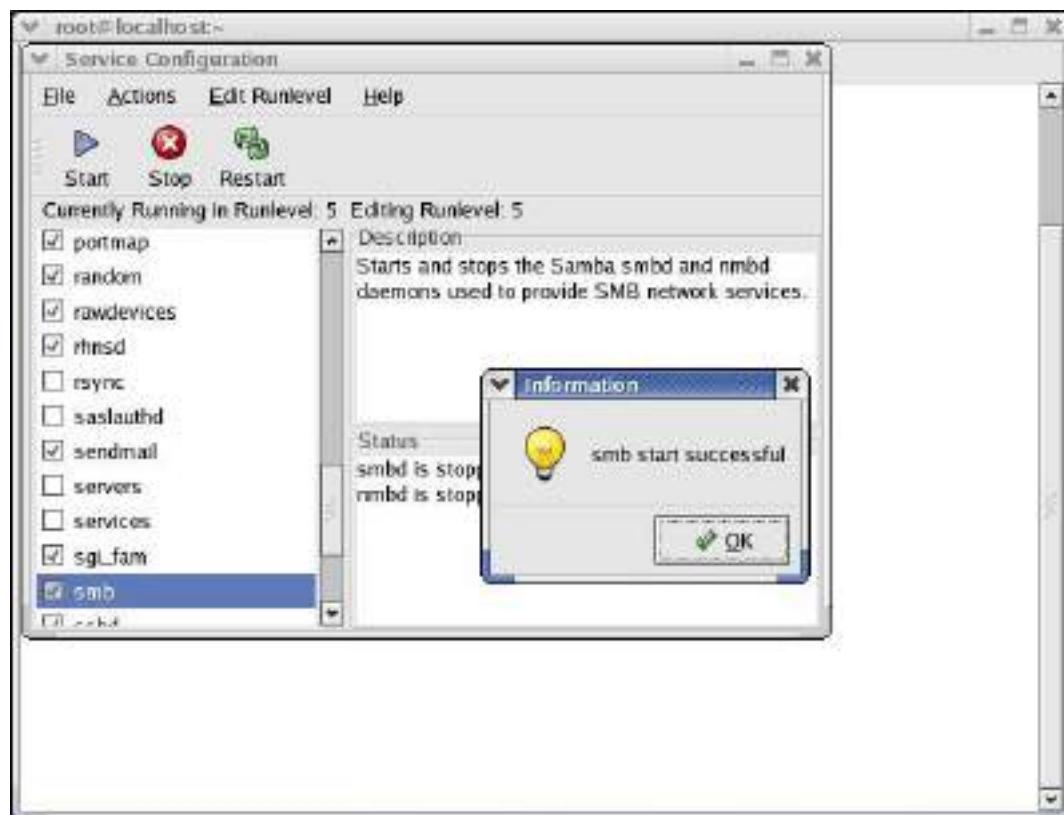
Starting NMB services:	[OK]
------------------------	--------

This command starts both SMB and NMB (NetBIOS name server), which are both services related to Samba. To stop the service, we'd type this:

```
# servicesmb stop
```

Shutting down SMB services:	[OK]
-----------------------------	--------

Shutting down NMB services:	[OK]
-----------------------------	--------



13.3 Samba Configuration Files and Utilities

Samba's configuration files are contained in the directory /etc/samba.

Configuration File	Description
smb.conf	This is the main configuration file for Samba.
lmhosts	This contains Samba's NetBIOS-to-IP address mappings.
secrets.tdb	This is the Samba secrets database. It stores private information such as the local SID and machine trust password. It is generated by machine and cannot be read in a text editor.
smbusers	This is a text file that maps your Linux system's users to various SMB-specific usernames.
smbpasswd	This is an encrypted password file. This file doesn't exist when you first install Samba, but is created when you add Samba users.

Some of the most important utilities provided by Samba. They're all contained in the directory /usr/bin.

Program	Purpose
smbclient	This is an FTP-like client, used to access SMB/CIFS resources on a file server.
smbadduser	This is a script, used for adding Samba users. It updates the smbusers and smbpasswd files.
smbpasswd	This changes a Samba user's SMB password. It is similar to the Unix passwd command
smbmount	This is used to mount an SMB filesystem.
smbumount	This is used to unmount an SMB file system.
smbstatus	This lists the current Samba connections.
testparm	This checks the smb.conf configuration file for correctness.
nmblookup	This is used to query NetBIOS names and map them to IP addresses in a network using NetBIOS over TCP/IP.

In addition, we'll also make use of the script /etc/rc.d/init.d/smb, which we use to start and stop the Samba fileservice.

13.4 Samba Configuration with SWAT

Perhaps the easiest way to configure Samba is by using the Samba Web Administration Tool(SWAT). SWAT is a web-based interface, which means that you can use it to configure and manage your Samba server through a web browser – if you want, you can even do it remotely across a network or even across the Internet.

Installing SWAT

- 1) Launch a terminal window, and switch to the root user account by using this command:
\$ su-
- 2) Insert Red Hat Linux 9 distribution Disk 2. Change to the directory on the CD that contains the RPM package files: # cd /mnt/cdrom/RedHat/RPMS
- 3) Use the ls command to find out the exact version of samba–swat contained on the disk. For example: # ls samba–swat*.rpm

```
samba–swat–2.2.7a–6.i386.rpm
```

If this command doesn't find any matches, then remove the disk, replace it with Disk 1 or Disk 2, and return to Step 2.

- 4) Install the samba–swat package you've just found, by using the rpm command:
rpm –ivh samba–swat–2.2.7a–6.i386.rpm

Starting the SWAT Service

Starting the SWAT service is a two step process:

- 1) Launch gedit or your favorite text editor, and open the file /etc/xinetd.d/swat. This is the configuration file for the SWAT service, and it looks like this:

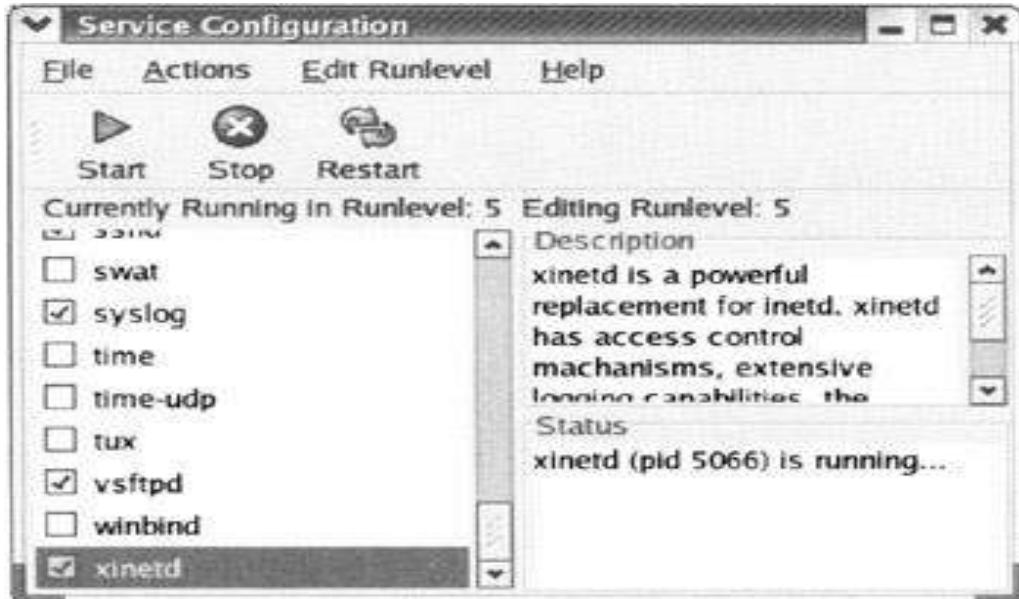
```
# default: off
# description: SWAT is the Samba Web Admin Tool. Use swat \
# to configure your Samba server. To use SWAT, \
# connect to port 901 with your favorite web browser.

service swat
{
    port = 901
    socket_type = stream
    wait = no
    only_from = 127.0.0.1
    user = root
    server = /usr/sbin/swat
    log_on_failure += USERID
    disable = yes
}
```

Change disable value, like this: disable = no. Save the file, and close your text editor.

- 2) Restart the xinetd service. (We must do this because SWAT runs as an xinetd service.

To do this, first launch the Service Configuration GUI tool by selecting Main Menu | System Settings | Server Settings | Services. Then locate and select the service called xinetd, and click Restart. You'll get a dialog to confirm that the restart was successful; then you can exit the tool by clicking on Quit.



Using SWAT for the First Time

Now we can test SWAT. Open a browser on the Linux server and type in the URL `http://localhost:901`. You'll be prompted for a username and password – use the system's root username and password. Then you should see the SWAT HOME page:



There are also eight buttons listed across the top of this page, which provide access to SWAT's various utilities:

- **HOME** – the home page of SWAT, and the first page that you see when you fire up the SWAT interface.
- **GLOBALS** – for setting the global variable values of the /etc/samba/smb.conf configuration file.
- **SHARES** – for creating and deleting Samba shares, and setting Samba parameters.
- **PRINTERS** – for creating and deleting Samba printer shares and setting printer parameters.
- **WIZARD** – like GLOBALS, this is also for setting various values in /etc/samba/smb.conf.
- **STATUS** – for viewing the Samba server's status, and starting and stopping Samba-related services
- **VIEW** – for viewing the content of the /etc/samba/smb.conf configuration file
- **PASSWORD** – for adding, removing, enabling, and disabling Samba users, and for setting Samba users' passwords

Adding a Samba User

To grant a system account access to the Samba services, you can use SWAT's PASSWORD feature. Once you've logged into SWAT using the root username and password, click the PASSWORD button, and you'll see the following screen:



In the Server Password Management section, enter the name of an existing account on the system, and supply a password. Then click the Add New User button. This will add an entry to Samba's smbpasswd configuration file, to indicate that this user has access to Samba's services.

Creating and Configuring a Samba Share

- 1) Create a directory, which we'll call /share, to be used for the file server. With root permission, you can do this at the command line using this command: # mkdir /share
- 2) If you haven't done so already, use the Mozilla browser (or your favorite web browser) to browse to the SWAT home page at http://localhost:901, and log in using the root user account.
- 3) Click on the GLOBALS toolbar icon. In the Base Options section, use the Workgroup field to enter the name of the workgroup that you want your server to appear in when clients use it. (If you haven't set up a workgroup, then this is probably the default value,

- WORKGROUP). You should also name the service, by entering a value in the server string field – it can be any string that you want your Samba clients to see.
- 4) Now click on SHARES toolbar button at the top of the screen. We'll share our /share directory by giving it a share name; let's call it linuxbox-share. You should first check the entries in the drop-down list on this page, to ensure that your chosen share name hasn't already been used for a different share; then, enter the share name in the Create Share field.
 - 5) Now click on Create Share button to create the share. This will present you with another screen in which you can specify the properties of your share. You will certainly need to set the path field to the path of your share directory (in this example, it's /share).
 - 6) Here, we've set the read only field to No to make it a writeable share; the browsable field to Yes, to allow the contents of the /share directory to be visible; and the available field to Yes, to "enable" the share (that is, make it available to users). We've also added a comment to remind us what the share is for. When you're done, click on the Commit Changes button to commit these settings to the Samba configuration file.
 - 7) Restart the Samba service, so that the configuration changes can take effect. To do this, click on STATUS button and then click on the Restart smbd button to restart the service. Wait for the page to reload, and then click on the Restart nmbd button to restart that service too.

Summary

- Samba is an implementation of the Windows SMB and CIFS protocols on Unix.
- Samba is freely available under GNU General Public License and is included as part of the Red Hat Linux 9 distribution.
- There are two ways to install the SAMBA Server suite via the RPM GUI tool: Main Menu | System Settings | Add/Remove Applications and \$ redhat-config-packages.
- There are two standard packages available in this: samba and samba-client.
- There are two ways to start the SAMBA Service, we can use the Service Configuration tool. These are: Main Menu | System Settings | Server Settings | Services and \$ redhat-config-services.
- If you ever have to perform an emergency reboot on your file server, then the "automatic start" configuration means that the file server is immediately available to users after the reboot.

Keywords

- **Samba:** It is a collection of programs that make it possible to share files and printers between computers equipped to use the SMB protocol – Windows by default and Linux/Unix with Samba.
- **Windows File Server:** This package group allows you to share files between Linux and MS windows system.
- **smb.conf:** This is the main configuration file for Samba.
- **Smbclient:** This is an FTP-like client, used to access SMB/CIFS resources on a file server.
- **SWAT:** It is a web-based interface, which means that you can use it to configure and manage your Samba server through a web browser – if you want, you can even do it remotely across a network or even across the Internet.

- **Xinetd:** xinetd is an open-source super-server daemon which runs on many Unix-like systems and manages Internet-based connectivity.

Self Assessment

1. In package group “Windows File Server”, which standard packages are available?
 - A. samba
 - B. samba-client
 - C. Both of the above
 - D. None of the above
2. Which of these is the main configuration file of SAMBA?
 - A. secrets.tdb
 - B. smb.conf
 - C. samba.conf
 - D. imhosts
3. Starting of SWAT service is _____ step process.
 - A. One
 - B. Two
 - C. Three
 - D. Four
4. SAMBA server on UNIX is an implementation of
 - A. Windows SMB
 - B. CIFS protocol
 - C. Both of the above
 - D. None of the above
5. Which server group will be chosen for installing SAMBA server?
 - A. DNS name server
 - B. FTP server
 - C. Mail server
 - D. Windows File server
6. Which of these commands will stop the smb service?
 - A. # servicesmb stop
 - B. # servicesmb end
 - C. # service stop smb
 - D. # service end smb
7. Which of these is unmount an SMB file system?
 - A. smbunmount

- B. smbmount
 - C. smbmount
 - D. None of the above
8. For configuration of SAMBA, which service is required?
- A. SWAT
 - B. FTP
 - C. TCP
 - D. None of the above
9. SWAT service will run as _____ service.
- A. FTP
 - B. TCP
 - C. xinted
 - D. None of the above
10. Which of these utility is used to create a directory?
- A. crtdir
 - B. mkdir
 - C. dircreate
 - D. None of the above
11. Because of the SAMBA server, it is possible to
- A. Share files between computers
 - B. Share printers between computers
 - C. Both of the above
 - D. None of the above
12. SAMBA server on UNIX is an implementation of
- A. Windows SMB
 - B. CIFS protocol
 - C. Both of the above
 - D. None of the above
13. How can we start the RPM GUI tool?
- A. Main Menu | System Settings | Add/Remove Applications
 - B. \$ redhat-config-packages
 - C. By using either of the way mentioned above
 - D. Something other than this
14. Which service needs to be started for SAMBA server through service configuration?
- A. smbd
 - B. nmbd

- C. Both of the above mentioned
- D. None of the above

- 15. By using which way, we can start the SAMBA service?
 - A. Main Menu | System Settings | Server Settings | Services
 - B. \$ redhat-config-services
 - C. By using either of the way mentioned above
 - D. None of the above

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. C | 2. B | 3. B | 4. C | 5. D |
| 6. A | 7. B | 8. A | 9. C | 10. B |
| 11. C | 12. C | 13. C | 14. C | 15. C |

Review Questions

1. What is a SAMBA server? What are the different ways to start RPM GUI tool for installation of SAMBA server?
2. What are the standard packages for SAMBA servers? How can we install the SAMBA server in the Linux system?
3. How can we start and stop the SAMBA service?
4. What are the different configuration files and utilities of SAMBA server? Explain.
5. What is SWAT? How can we install SWAT?
6. How can we start the SWAT service? How to use the SWAT service for first time?
7. How can we create and configure a SAMBA share?



Further Readings

Sandip Bhattacharya, Pancrazio De Mauro, Shishir Gundavaram, Mark Mamone, Kalip Sharma, Deepak Thomas and Simon Whiting, Beginning Red Hat Linux 9, Wiley Publishing, Inc.



Web Links

<https://linux.die.net/man/8/swat>

Unit 14: Network File Systems

CONTENTS

Objectives

Introduction

14.1 Setting Up an NFS Client

14.2 Setting Up an NFS Server

14.3 Testing the Server Setup

14.4 Automount: Automatically Mounts Directory Hierarchies

Summary:

Keywords:

Self Assessment

Answers for Self Assessment

Review Questions:

Further Readings

Objectives

After studying this unit, you will be able to:

- Understand NFS
- Plan NFS installation
- Configure NFS server and client
- Use automount service
- Examine NFS security

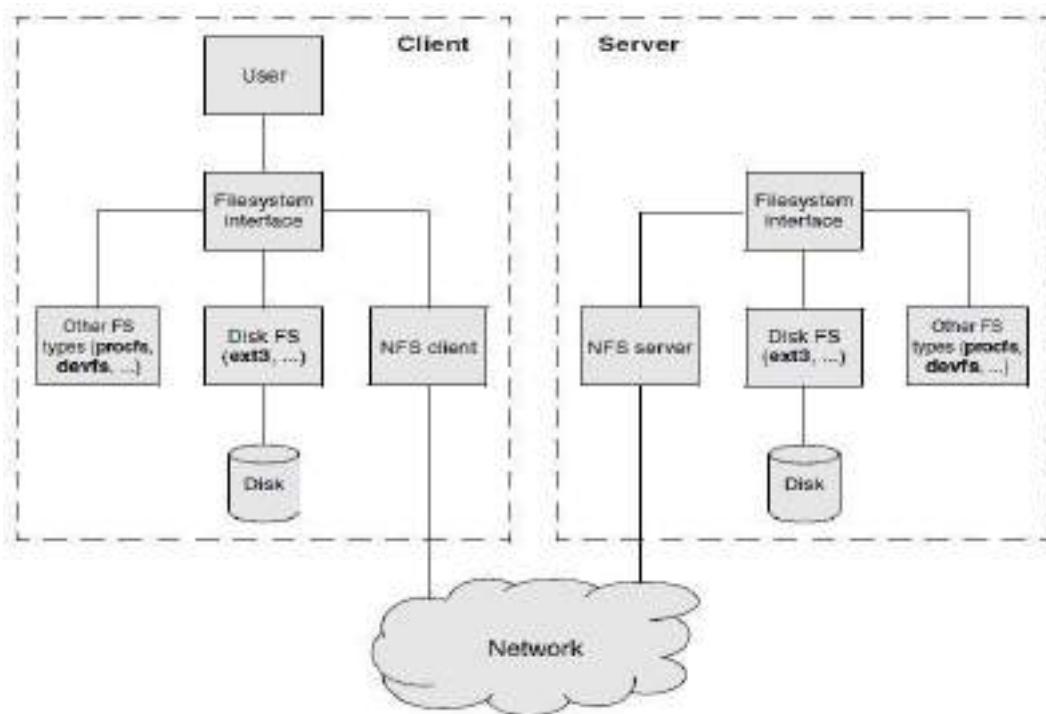
Introduction

NFS stands for Network Filesystem protocol. It is a UNIX de facto standard originally developed by Sun Microsystems. It allows a server to share selected local directory hierarchies with client systems on a heterogeneous network. NFS runs on UNIX, DOS, Windows, VMS, Linux, and more. Files on the remote computer (the fileserver) appear as if they are present on the local system (the client). The physical location of a file is irrelevant to an NFS user.

NFS reduces storage needs and system administration workload. As an example, each system in a company traditionally holds its own copy of an application program. To upgrade the program, the administrator needs to upgrade it on each system. NFS allows you to store a copy of a program on a single system and give other users access to it over the network. This scenario minimizes storage requirements by reducing the number of locations that need to maintain the same data. In addition to boosting efficiency, NFS gives users on the network access to the same data (not just application programs), thereby improving data consistency and reliability. By consolidating data, NFS reduces administrative overhead and provides a convenience to users. The flow of data from a client to server can be seen in the figure given below.

An NFS directory hierarchy appears to users and application programs as just another directory hierarchy. By looking at it, you cannot tell that a given directory holds a remotely mounted NFS directory hierarchy and not a local ext3 filesystem. The NFS server translates commands from the client into operations on the server's filesystem.

- 1) **Diskless systems:** In many computer facilities, user files are stored on a central fileserver equipped with many large-capacity disk drives and devices that quickly and easily make backup copies of the data. A diskless system boots from a fileserver (netboots), a CD, or a floppy diskette and loads system software from a fileserver. The Linux Terminal Server Project (ltsp.org) Web site says it all: "Linux makes a great platform for deploying diskless workstations that boot from a network server. The LTSP is all about running thin client computers in a Linux environment." Because a diskless workstation does not require a lot of computing power, you can give older, retired computers a second life by using them as diskless systems.



- 2) **Netboot/PXE:** You can netboot systems that are appropriately set up. Fedora/RHEL includes the PXE (Preboot Execution Environment) server package for netbooting Intel systems. Older systems sometimes use tftp (Trivial File Transfer Protocol) for netbooting. Non-Intel architectures have historically included netboot capabilities, which Fedora/RHEL also supports. You can build the Linux kernel so that it mounts root (/) using NFS. Given the many ways to set up a system, the one you choose depends on what you want to do. See the Remote-Boot mini-HOWTO for more information.
- 3) **Dataless systems:** Another type of Linux system is a dataless system, in which the client has a disk but stores no user data (only Linux and the applications are kept on the disk). Setting up this type of system is a matter of choosing which directory hierarchies are mounted remotely.

df: shows where directory hierarchies are mounted: The df utility displays a list of the directory hierarchies available on the system, along with the amount of disk space, free and used, on each. The -h (human) option makes the output more intelligible. Directory hierarchy names that are prepended with hostname: are available through NFS.

- 4) **Errors:** Sometimes you may lose access to remote files. For example, a network problem or a remote system crash may make these files temporarily unavailable. When you try to access a remote file in these circumstances, you get an error message, such as NFS server speedy not responding. When the local system can contact the remote server again, you see another message, such as NFS server speedy OK. Setting up a stable network and server (or not using NFS) is the best defense against these kinds of problems.
- 5) **Security:** NFS is based on the trusted-host paradigm and therefore has all the security shortcomings that plague other services based on this paradigm. In addition, NFS is not encrypted. Because of these issues, you should implement NFS on a single LAN segment only, where you can be (reasonably) sure that systems on a LAN segment are what they claim to be. Make sure a firewall blocks NFS traffic from outside the LAN and never use NFS over the Internet.

14.1 Setting Up an NFS Client

The Prerequisites for setting up an NFS client is to install the following packages:

- nfs-utils
- system-config-nfs (optional)

Under RHEL, the portmap utility must be running to enable reliable file locking. Under FEDORA, this function is served by rpcbind.

JumpStart I: Mounting a Remote Directory Hierarchy

To set up an NFS client, mount the remote directory hierarchy the same way you mount a local directory hierarchy.

- **mount: Mounts a Remote Directory Hierarchy**

There are various assumption for mounting a remote directory hierarchy. These are: (1) Speedy is on the same network as the local system and is sharing /home and /export with the local system.

- (2) The /export directory on speedy holds two directory hierarchies that you want to mount: /export/progs and /export/oracle.
- (3) The example mounts speedy's /home directory on /speedy.home on the local system, /export/progs on /apps, and /export/oracle on /oracle.

First use mkdir to create the directories that are the mount points for the remote directory hierarchies:

- **# mkdir/speedy.home /apps /oracle**

You can mount any directory from an exported directory hierarchy. In this example, speedy exports /export and the local system mounts /export/progs and /export/oracle. The following commands manually mount the directory hierarchies one time:

```
# mount speedy/home/speedy.home
# mount -o ro,nosuid speedy/export/progs /apps
# mount -o ro speedy:/export/oracle /oracle
```

The first command mounts the /home directory hierarchy from speedy on the local directory /speedy.home. The second and third commands use the -o ro option to force a readonly mount. The second command adds the nosuid option, which forces setuid executables in the mounted directory hierarchy to run with regular permissions on the local system. If you receive the error mount: RPC: Program not registered, it may mean NFS is not running on the server. By default, directory hierarchies are mounted read-write, assuming the NFS server is exporting them with read-write permissions.

- **nosuid option:**

If a user could run a setuid program, that user has the power of Superuser. This ability should be limited. Unless you know that a user will need to run a program with setuid permissions from a mounted directory hierarchy, always mount a directory hierarchy with the nosuid option. For example, you would need to mount a directory hierarchy with setuid privileges when a diskless workstation has its root partition mounted using NFS.

- **nodev option**

Mounting a device file creates another potential security hole. Although the best policy is not to mount untrustworthy directory hierarchies, it is not always possible to implement this policy. Unless a user needs to use a device on a mounted directory hierarchy, mount directory hierarchies with the nodev option, which prevents character and block special files on the mounted directory hierarchy from being used as devices.

- **fstab file**

If you mount directory hierarchies frequently, you can add entries for the directory hierarchies to the **/etc/fstab** file.

```
$ cat /etc/fstab
...
speedy:/home          /speedy.home    nfs - 0 0
speedy:/export/progs   /apps         nfsr,nosuid 0 0
speedy:/export/oracle  /oracle        nfs r 0 0
```

A file that is mounted using NFS is always type nfs on the local system, regardless of what type it is on the remote system. Typically, you do not run fsck on or back up an NFS directory hierarchy. The entries in the third, fifth, and sixth columns of fstab are usually nfs (filesystem type), 0 (do not back up this directory hierarchy with dump), and 0 (do not run fsck on this directory hierarchy). The options for mounting an NFS directory hierarchy differ from those for mounting an ext3 or other type of filesystem.

- **umount: Unmounts a Remote Directory Hierarchy**

Use umount to unmount a remote directory hierarchy the same way you would unmount a local filesystem

mount: Mounts a Directory Hierarchy

The mount utility associates a directory hierarchy with a mount point (a directory). You can use mount to mount an NFS (remote) directory hierarchy.

Attribute Caching

File attributes, which are stored in a file's inode, provide information about a file, such as file modification time, size, links, and owner. File attributes do not include the data stored in a file. Typically file attributes do not change very often for an ordinary file; they change even less often for a directory file. Even the size attribute does not change with every write instruction: When a client is writing to an NFS-mounted file, several write instructions may be given before the data is transferred to the server

ac (noac) (attribute cache): It permits attribute caching (default). The noac option disables attribute caching. Although noac slows the server, it avoids stale attributes when two NFS clients actively write to a common directory hierarchy.

acdirmax=n (attribute cache directory file maximum): The n is the number of seconds, at a maximum, that NFS waits before refreshing directory file attributes (default is 60 seconds).

acdirmi=n (attribute cache directory file minimum): The n is the number of seconds, at a minimum, that NFS waits before refreshing directory file attributes (default is 30 seconds).

acregmax=n (attribute cache regular file maximum): The n is the number of seconds, at a maximum, that NFS waits before refreshing regular file attributes (default is 60 seconds).

acregmin=n (attribute cache regular file minimum): The n is the number of seconds, at a minimum, that NFS waits before refreshing regular file attributes (default is 3 seconds).

actimeo=n (attribute cache timeout): This option sets acregmin, acregmax, acdirmmin, and acdirmmax to *n seconds* (without this option, each individual option takes on its assigned or default value).

Error Handling

Various options control what NFS does when the server does not respond or when an I/O error occurs. To allow for a mount point located on a mounted device, a missing mount point is treated as a timeout.

- **fg (bg) (foreground):** The option fg retries failed NFS mount attempts in the foreground (default). The bg (background) option retries failed NFS mount attempts in the background.
- **hard (soft) :** This option displays server not responding on the console on a major timeout and keeps retrying (default). The soft option reports an I/O error to the calling program on a major timeout. In general, it is not advisable to use soft. As the mount man page says of soft, "Usually it just causes lots of trouble."
- **nointr (intr) (no interrupt):** This option does not allow a signal to interrupt a file operation on a hard mounted directory hierarchy when a major timeout occurs (default). The intr option allows this type of interrupt.
- **retrans=n (retransmission value):** After n minor timeouts, NFS generates a major timeout (default is 3). A major timeout aborts the operation or displays server not responding on the console, depending on whether hard or soft is set.
- **retry=n (retry value):** The number of minutes that NFS retries a mount operation before giving up (default is 10,000).
- **timeo=n (timeout value):** The n is the number of tenths of a second that NFS waits before retransmitting following an RPC, or minor, timeout (default is 7). The value is increased at each timeout to a maximum of 60 seconds or until a major timeout occurs. On a busy network, in case of a slow server, or when the request passes through multiple routers/gateways, increasing this value may improve performance.

Miscellaneous Options

- **lock (nolock):** Permits NFS locking (default). The nolock option disables NFS locking (does not start the lockd daemon) and is useful with older servers that do not support NFS locking.
- **mounthost=name:** The name of the host running mountd, the NFS mount daemon.
- **mountport=n:** The port used by mountd.
- **nodev (no device):** Causes mounted device files not to function as devices.
- **port=n:** The port used to connect to the NFS server (defaults to 2049 if the NFS daemon is not registered with rpcbind/portmap). When n=0 (default), NFS queries rpcbind/portmap on the server to determine the port.
- **rsize=n (read block size)** The number of bytes read at one time from an NFS server. The default block size is 4096.
- **wsize=n (write block size):** The number of bytes written at one time to an NFS server. The default block size is 4096.

- **tcp:** Use TCP in place of the default UDP protocol for an NFS mount. This option may improve performance on a congested network; however, some NFS servers support UDP only.
- **udp:** Use the default UDP protocol for an NFS mount.

Improving Performance:

- **hard/soft:** Several parameters can affect the performance of NFS, especially over slow connections such as a line with a lot of traffic or one controlled by a modem. If you have a slow connection, make sure hard is set (this is the default) so that timeouts do not abort program execution.
- **Block size:** One of the easiest ways to improve NFS performance is to increase the block size—that is, the number of bytes NFS transfers at a time. The default of 4096 is low for a fast connection using modern hardware. Try increasing **rsize and wsize to 8192** or higher. Experiment until you find the optimal block size. Unmount and mount the directory hierarchy each time you change an option.
- **Timeouts:** NFS waits the amount of time specified by the timeo option for a response to a transmission. If it does not receive a response in this amount of time, it sends another transmission. The second transmission uses bandwidth that, over a slow connection, may slow things down further. You may be able to increase performance by increasing **timeo**.

/etc/fstab: Mounts Directory Hierarchies Automatically

The /etc/fstab file lists directory hierarchies that the system mounts automatically as it comes up. This example line from fstab mounts grape's /gc1 filesystem on the /grape.gc1 mount point:

```
grape/gc1      /grape.gc1      nfsrsize=8192,wsize=8192 0      0
```

A mount point should be an empty, local directory. Files in a mount point are hidden when a directory hierarchy is mounted on it. The type of a filesystem mounted using NFS is always nfs, regardless of its type on the local system. You can increase the rsize and wsize options to improve performance. This example from fstab mounts a filesystem from speedy:

```
speedy/export  /speedy.export  nfstimeo=50,hard      0      0
```

Because the local system connects to speedy over a slow connection, timeo is increased to 5 seconds (50 tenths of a second). This example from fstab shows a remote-mounted home directory. Because speedy is a local server and is connected via a reliable, high-speed connection, timeo is decreased and rsize and wsize are increased substantially:

```
speedy/export/home /home nfstimeo=4,rsize=16384,wsize=16384      0      0
```

14.2 Setting Up an NFS Server

The prerequisites for setting up NFS server is to install the following packages:

nfs-utils

system-config-nfs (optional)

After this run chkconfig to cause nfs to start when the system enters multiuser mode:

```
# /sbin/chkconfig nfs on
```

The nfs can be started by:

```
# /sbin/service nfs start
```

The nfsinit script starts mountd, nfsd, and rquotad. RHEL Under RHEL, the portmap daemon must be running to enable reliable file locking.

JumpStart II: Configuring an NFS Server Using system-config-nfs:

To display the NFS Server Configuration window, enter the command system-config-nfs or select Main Menu: System | Administration | Server Settings | NFS. From this window you can generate an /etc/exports file, which is almost all there is to setting up an NFS server. The system-config-nfs utility allows you to specify which directory hierarchies are shared and how they are shared using NFS. Each exported hierarchy is called a share. To add a share, click Add on the toolbar. To modify a share, highlight the share and click Properties on the toolbar. Clicking Add displays the Add NFS Share window, while clicking Properties displays the Edit NFS Share window. The Add/Edit NFS Share window has three tabs: Basic, General Options, and User Access. These are given as:

- **Basic tab:** You can specify the pathname of the root of the shared directory hierarchy, the names or IP addresses of the systems (hosts) that the hierarchy will be shared with, and whether users from the specified systems will be able to write to the shared files. The selections in the other two tabs correspond to options that you can specify in the /etc/exports file.
- **General Options tab:** In this tab, make the following changes:

Allow connections from ports 1024 and higher: **insecure**

Allow insecure file locking: **no_auth_nlm** or **insecure_locks**

Disable subtree checking: **no_subtree_check**

Sync write operations on request: **sync**

Force sync of write operations immediately: **no_wdelay**

Hide filesystems beneath: **nohide**

Export only if mounted: **mountpoint**

- **User Access tab:** In this tab, make the following changes:

Treat remote root user as local root: **no_root_squash**

Treat all client users as anonymous users: **all_squash**

Local user ID for anonymous users: **anonuid**

Local group ID for anonymous users: **anongid**

After making the changes you want, click OK to close the Add/Edit NFS Share window and click OK again to close the NFS Server Configuration window. There is no need to restart any daemons.

Exporting a Directory Hierarchy

Exporting a directory hierarchy makes the directory hierarchy available for mounting by a client on the network. "Exported" does not mean "mounted": When a directory hierarchy is exported, it is placed in the list of directory hierarchies that can be mounted by other systems. An exported directory hierarchy may be mounted (or not) at any given time. A server holds three lists of exported directory hierarchies:

- **/etc/exports** – Access control list for exported directory hierarchies. The system administrator can modify this file by editing it or by running system-config-nfs.

- **/var/lib/nfs/xtab** – Access control list for exported directory hierarchies. Initialized from /etc/exports when the system is brought up. Read by mountd when a client asks to mount a directory hierarchy. Modified by exportfs as directory hierarchies are mounted and unmounted by NFS.
- **Kernel's export table** – List of active exported directory hierarchies. The kernel obtains this information from /var/lib/nfs/xtab. You can display this table by giving the command cat /proc/fs/nfs/exports. The /etc/exports file is the access control list for exported directory hierarchies that NFS clients can mount; it is the only file you need to edit to set up an NFS server. The exports file controls the following aspects:
 - 1) Which clients can access files on the server?
 - 2) Which directory hierarchies on the server each client can access
 - 3) How each client can access each directory hierarchy
 - 4) How client usernames are mapped to server usernames
 - 5) Various NFS parameters

Each line in the exports file has the following format:

export-point client1(options) [client2(options) ...]

where export-point is the absolute pathname of the root directory of the directory hierarchy to be exported, client1-n is the name of one or more clients or is one or more IP addresses, separated by SPACES, that are allowed to mount the export-point. You can either use system-config-nfs to make changes to exports or you can edit this file directly. The following simple exports file gives grape read and write access and gives speedy readonly access to the files in /home:

```
# cat /etc(exports
/home grape(rw,no_subtree_check)
/home speedy(ro,no_subtree_check)
```

General options

- **auth_nlm (no_auth_nlm) or secure_locks (insecure_locks)**: This causes the server to require authentication of lock requests (using the NLM [NFS Lock Manager] protocol). Use no_auth_nlm for older clients when you find that only files that anyone can read can be locked.
- **mountpoint[=path]**: It allows a directory to be exported only if it has been mounted. This option prevents a mount point that does not have a directory hierarchy mounted on it from being exported and prevents the underlying mount point from being exported.
- **nohide (hide)**: When a server exports two directory hierarchies, one of which is mounted on the other, a client must mount both directory hierarchies explicitly to access both. When the second (child) directory hierarchy is not explicitly mounted, its mount point appears as an empty directory and the directory hierarchy is hidden. The nohide option causes the underlying second directory hierarchy to appear when it is not explicitly mounted, but this option does not work in all cases.
- **ro (rw) (readonly)**: It permits only read requests on an NFS directory hierarchy. Use rw to permit read and write requests.
- **secure (insecure)**: It requires that NFS requests originate on a privileged port so that a program without root permissions cannot mount a directory hierarchy. This option does not guarantee a secure connection.

- **subtree_check (no_subtree_check):** It checks subtrees for valid files. Assume that you have an exported directory hierarchy that has its root below the root of the filesystem that holds it (that is, an exported subdirectory of a filesystem). When the NFS server receives a request for a file in that directory hierarchy, it performs a subtree check to confirm the file is in the exported directory hierarchy.
- **sync (async) (synchronize):** It specifies that the server is to reply to requests only after disk changes made by the request are written to disk. The **async option** specifies that the server does not have to wait for information to be written to disk and can improve performance, albeit at the cost of possible data corruption if the server crashes or the connection is interrupted.
- **Wdelay (no_wdelay) (write delay):** Causes the server to delay committing write requests when it anticipates that another, related request follows, thereby improving performance by committing multiple write requests within a single operation. The **no_wdelay option** does not delay committing write requests and can improve performance when the server receives multiple, small, unrelated requests.

User ID Mapping Options

Each user has a UID number and a primary GID number on the local system. The local /etc/passwd and /etc/group files map these numbers to names. When a user makes a request of an NFS server, the server uses these numbers to identify the user on the remote system. It raises several issues: the user may not have the same ID numbers on both systems and may therefore have owner access to files of another user. You may not want the root user on the client system to have owner access to root-owned files on the server. You may not want a remote user to have owner access to some important system files that are not owned by root (such as those owned by bin).

Owner access means that the remote user can execute, remove, or – worse – modify the file. NFS gives you two ways to deal with these cases:

- You can use the **root_squash** option to map the ID number of the root user on a client to the nfsnobody user on the server.
- You can use the **all-squash** option to map all NFS users on the client to nfsnobody on the server

NIS and NFS: When you use NIS for user authorization, users automatically have the same UIDs on both systems. If you are using NFS on a large network, it is a good idea to use a directory service such as LDAP or NIS for authorization. Without such a service, you must synchronize the passwd files on all the systems manually.

root_squash (no_root_squash): It maps requests from root on a remote system so that they appear to come from the UID for nfsnobody, an unprivileged user on the local system, or as specified by a nonuid. It does not affect other sensitive UIDs such as bin. The **no_root_squash** option turns off this mapping so that requests from root appear to come from root.

no_all_squash (all_squash): It does not change the mapping of users making requests of the NFS server. The **all_squash** option maps requests from all users, not just root, on remote systems to appear to come from the UID for nfsnobody, an unprivileged user on the local system, or as specified by a nonuid. This option is useful for controlling access to exported public FTP, news, and other directories.

anonuid=un and anongid=gn: It sets the UID or the GID of the anonymous account to un or gn, respectively. NFS uses these accounts when it does not recognize an incoming UID or GID and when instructed to do so by root_squash or all_squash.

showmount:

It displays NFS status information. Without any options, the showmount utility displays a list of systems that are allowed to mount local directories. To display information for a remote system, give the name of the remote system as an argument. You typically use showmount to display a list of directory hierarchies that a server is exporting. The information that showmount provides may not be complete, however, because it depends on mountd and trusts that remote servers are reporting accurately. In the following example, bravo and grape can mount local directories, but you do not know which ones:

```
# /usr/sbin/showmount
```

Hosts on localhost:

bravo.tcorp.com

grape.tcorp.com

If showmount displays an error such as RPC: Program not registered, NFS is not running on the server. Start NFS on the server with the nfsinit script

-a (all): It tells which directories are mounted by which remote systems. This information is stored in /etc/exports.

```
# /usr/sbin/showmount -a
```

All mount points on localhost:

bravo.tcorp.com:/home

grape.tcorp.com:/home

-e (exports): It displays a list of exported directories.

```
# /usr/sbin/showmount -e
```

Export list for localhost:

/home bravo.tcorp.com,grape.tcorp.com

exportfs: Maintains the List of ExportedDirectory Hierarchies

The exportfs utility maintains the kernel's list of exported directory hierarchies. Without changing /etc/exports, exportfs can add to or remove from the list of exported directory hierarchies. An exportfs command has the following format:

```
/usr/sbin/exportfs [options] [client:dir ...]
```

where options is one or more options (as detailed in the next section), client is the name of the system that dir is exported to, and dir is the absolute pathname of the directory at the root of the directory hierarchy being exported. The system executes the following command when it comes up (it is in the nfsinit script). This command reexports the entries in /etc/exports and removes invalid entries from /var/lib/nfs/xtab so that /var/lib/nfs/xtab is synchronized with /etc/exports:

```
# exportfs -r
```

Options: There are various options which we can use:

- **-a (all):** Exports directory hierarchies specified in /etc/exports. This option does not unexport entries you have removed from exports (that is, it does not remove invalid entries from /var/lib/nfs/xtab); use -r to perform this task.

- **-i (ignore):** Ignores /etc/exports; uses what is specified on the command line only.
- **-o (options):** Specifies options. You can specify options following -o the same way you do in the exports file. For example, exportfs -i -o ro speedy:/home/sam exports /home/sam on the local system to speedy for readonly access.
- **-r (reexport):** Reexports the entries in /etc/exports and removes invalid entries from /var/lib/nfs/xtab so that /var/lib/nfs/xtab is synchronized with /etc/exports.
- **-u (unexport)** Makes an exported directory hierarchy no longer exported. If a directory hierarchy is mounted when you unexport it, you will see the message Stale NFS file handle if you try to access the directory hierarchy from the remote system.
- **-v (verbose):** Provides more information. Displays export options when you use exportfs to display export information.

14.3 Testing the Server Setup

From the server, run the nfsinit script with an argument of status. If all is well, the system displays something like the following:

```
#/sbin/service nfs status
rpc.mountd (pid 15795) is running...
nfsd (pid 15813 15812 15811 15810 15809 15808 15807 15806) is running...
rpc.rquotad (pid 15784) is running...
```

Next, from the server, use rpcinfo to make sure NFS is registered with rpcbind/portmap:

```
$/usr/sbin/rpcinfo -p localhost | grep nfs
100003 2 udp 2049 nfs
100003 3 udp 2049 nfs
```

Repeat the preceding command from the client, replacing localhost with the name of the server. The results should be the same. Finally, try mounting directory hierarchies from remote systems and verify access.

14.4 Automount: Automatically Mounts Directory Hierarchies

With distributed computing, when you log in on any system on the network, all of your files, including startup scripts, are available. In a distributed computing environment, all systems are commonly able to mount all directory hierarchies on all servers: Whichever system you log in on, your home directory is waiting for you. As an example, assume that /home/alex is a remote directory hierarchy that is mounted on demand. When you issue the command ls /home/alex, autoofs goes to work: It looks in the /etc/auto.home map, finds that alex is a key that says to mount bravo:/export/home/alex, and mounts the remote directory hierarchy. Once the directory hierarchy is mounted, ls displays the list of files you want to see. If you give the command ls /home after this mounting sequence, ls shows that alex is present within the /home directory. The df utility shows that alex is mounted from bravo.

Prerequisites

The prerequisites is to install the following package: **autoofs**

Run chkconfig to cause autoofs to start when the system enters multiuser mode:

```
#/sbin/chkconfig autoofs on
```

Start autoofs:

```
# /sbin/service autofs start
```

An **autofs directory hierarchy** is like any other directory hierarchy, but remains unmounted until it is needed, at which time the system mounts it automatically (demand mounting). The system unmounts an autofs directory hierarchy when it is no longer needed – by default after five minutes of inactivity. Automatically mounted directory hierarchies are an important part of administrating a large collection of systems in a consistent way. The automount daemon is particularly useful when an installation includes many servers or many directory hierarchies. It also helps to remove server-server dependencies. When you boot a system that uses traditional fstab-based mounts and an NFS server is down, the system can take a long time to come up as it waits for the server to time out. Similarly, when you have two servers, each mounting directory hierarchies from the other, and both systems are down, both may hang as they are brought up and each tries to mount a directory hierarchy from the other. This situation is called a server-server dependency. The automount facility gets around these issues by mounting a directory hierarchy from another system only when a process tries to access it. When a process attempts to access one of the directories within an unmounted autofs directory hierarchy, the kernel notifies the automount daemon, which mounts the directory hierarchy. You must give a command, such as cd /home/alex, that accesses the autofs mount point (in this case /home/alex) to create the demand that causes automount to mount the autofs directory hierarchy so you can see it. Before you issue the cd command, alex does not appear to be in /home. The main file that controls the behavior of automount is /etc/auto.master. Example:

```
# cat /etc/auto.master
/free1  /etc/auto.misc  --timeout      60
/free2  /etc/auto.misc2 --timeout      60
```

The auto.master file has three columns.

- The first column names the parent of the autofs mount point—the location where the autofs directory hierarchy is to be mounted (/free1 and /free2 in the example are not mount points but will hold the mount points when the directory hierarchies are mounted).
- The second column names the files, called map files, that store supplemental configuration information.
- The optional third column holds mount options for map entries that do not specify an option.

Although the map files can have any names, one is traditionally named auto.misc. Following are the two map files specified in auto.master:

```
# cat /etc/auto.misc
sam -fstype=ext3 :/dev/sda8

# cat /etc/auto.misc2
helen -fstype=ext3 :/dev/sda9
```

Before the new setup can work, you must create directories for the parents of the mount points (/free1 and /free2 in the preceding example) and start (or restart) the automount daemon using the autofsinit script. The following command displays information about configured and active autofs mount points:

```
# /sbin/service autofs status
```

Summary:

- NFS runs on UNIX, DOS, Windows, VMS, Linux, and more. Files on the remote computer (the fileserver) appear as if they are present on the local system (the client). The physical location of a file is irrelevant to an NFS user.

- NFS is based on the trusted-host paradigm and therefore has all the security shortcomings that plague other services based on this paradigm.
- To display the NFS Server Configuration window, enter the command system-config-nfs or select Main Menu: System | Administration | Server Settings | NFS.
- A file that is mounted using NFS is always type nfs on the local system, regardless of what type it is on the remote system.
- The mount utility associates a directory hierarchy with a mount point (a directory). You can use mount to mount an NFS (remote) directory hierarchy.
- A server holds three lists of exported directory hierarchies. These are: /etc/exports, /var/lib/nfs/xtab and kernel's export table.

Keywords:

- **NFS:** NFS stands for Network Filesystem protocol. It is a UNIX de facto standard originally developed by Sun Microsystems. It allows a server to share selected local directory hierarchies with client systems on a heterogeneous network.
- **NFS Server:** The NFS server translates commands from the client into operations on the server's filesystem.
- **df utility:** The df utility displays a list of the directory hierarchies available on the system, along with the amount of disk space, free and used, on each.
- **portmap utility:** The portmap utility must be running to enable reliable file locking.
- **umount:** It unmounts a remote directory hierarchy. Use umount to unmount a remote directory hierarchy the same way you would unmount a local filesystem.
- **Share:** The system-config-nfs utility allows you to specify which directory hierarchies are shared and how they are shared using NFS. Each exported hierarchy is called a share.
- **exportfs:** The exportfs utility maintains the kernel's list of exported directory hierarchies. Without changing /etc/exports, exportfs can add to or remove from the list of exported directory hierarchies

Self Assessment

1. The problem in NFS security is
 - A. NFS is encrypted
 - B. NFS is not encrypted
 - C. NFS does not respond
 - D. None of the above

2. Which of these options disables a signal to interrupt a file operation on hardmounted directory hierarchy?
 - A. intr
 - B. nointr
 - C. unintr
 - D. None of the above

3. Which of these options displays a list of exported directories?
 - A. -a
 - B. -e
 - C. -list
 - D. None of the above

4. In FEDORA, which utility is used for setting up an NFS client?
 - A. portmap
 - B. rpcbind
 - C. setupnfs
 - D. None of the above

5. By default block size in wsize and rsize is _____
 - A. 128
 - B. 256
 - C. 1468
 - D. 4096

6. Which of these options are used to mount a remote directory hierarchy?
 - A. mount
 - B. automount
 - C. Both of the above
 - D. None of the above

7. Which utility displays a list of directory hierarchies available on the system?
 - A. df
 - B. dir
 - C. dirhier
 - D. None of the above

8. Which of these options disables the attribute caching?
 - A. ac
 - B. noac
 - C. unac
 - D. None of the above

9. Which of these options permits only read access on an NFS directory hierarchy?
 - A. r
 - B. ro
 - C. rw
 - D. or

10. NFS
 - A. Reduces the storage requirement

- B. Boosts efficiency
 - C. Reduces administration workload
 - D. All of the above
11. If a user can run a setuid program, that user
- A. Is a normal user
 - B. Has the power of a superuser
 - C. Hides the identity
 - D. None of the above
12. Kernel's export table consists of
- A. Active exported directory hierarchies
 - B. Inactive exported directory hierarchies
 - C. Both active and inactive
 - D. None of the above
13. NFS stands for
- A. Network FileSystem
 - B. Not a FileSystem
 - C. New Filesystem
 - D. None of the above
14. In RHEL, which utility is used for setting up an NFS client?
- A. portmap
 - B. rpcbind
 - C. setupnfs
 - D. None of the above
15. The NFS performance can be improved by _____
- A. Increasing the block size
 - B. Decreasing the block size
 - C. Block size should remain constant
 - D. None of the above

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. B | 2. B | 3. B | 4. B | 5. D |
| 6. C | 7. A | 8. B | 9. B | 10. D |
| 11. B | 12. C | 13. A | 14. A | 15. A |

Review Questions:

1. What is NFS? Explain the flow of data from client to server with the help of a diagram.
2. Explain the various features of NFS.
3. How to set up an NFS client? Explain.
4. Explain the utility which is used to mount a directory hierarchy with various options available with it.
5. What is error handling? Also explain various options available with it.
6. How to set up an NFS server?
7. How to export a directory hierarchy? Explain the lists associated with this.
8. How to test the server setup? Explain autoofs.



Further Readings

Mark G Sobell, A Practical Guide to Fedora and RedHat Enterprise Linux, Fifth Edition,
Prentice Hall



Web Links

<https://cloud.netapp.com/blog/azure-anf-blg-linux-nfs-server-how-to-set-up-server-and-client>

LOVELY PROFESSIONAL UNIVERSITY

Jalandhar-Delhi G.T. Road (NH-1)

Phagwara, Punjab (India)-144411

For Enquiry: +91-1824-300360

Fax.: +91-1824-506111

Email: odl@lpu.co.in

Advanced Data Structures

DECAP770

**Edited by
Balraj Kumar**



LOVELY
PROFESSIONAL
UNIVERSITY



L O V E L Y
P R O F E S S I O N A L
U N I V E R S I T Y

Advanced Data Structures

Edited By:
Balraj Kumar

CONTENT

Unit 1:	Introduction	1
	<i>Ashwani Kumar, Lovely Professional University</i>	
Unit 2:	Arrays vs Linked Lists	18
	<i>Ashwani Kumar, Lovely Professional University</i>	
Unit 3:	Stacks	43
	<i>Ashwani Kumar, Lovely Professional University</i>	
Unit 4:	Queues	57
	<i>Ashwani Kumar, Lovely Professional University</i>	
Unit 5:	Search Trees	73
	<i>Ashwani Kumar, Lovely Professional University</i>	
Unit 6:	Tree Data Structure 1	87
	<i>Ashwani Kumar, Lovely Professional University</i>	
Unit 7:	Tree Data Structure 2	102
	<i>Ashwani Kumar, Lovely Professional University</i>	
Unit 8:	Heaps	125
	<i>Ashwani Kumar, Lovely Professional University</i>	
Unit 9:	More on Heaps	139
	<i>Ashwani Kumar, Lovely Professional University</i>	
Unit 10:	Graphs	163
	<i>Ashwani Kumar, Lovely Professional University</i>	
Unit 11:	More on Graphs	180
	<i>Ashwani Kumar, Lovely Professional University</i>	
Unit 12:	Hashing Techniques	202
	<i>Ashwani Kumar, Lovely Professional University</i>	
Unit 13:	Collision Resolution	215
	<i>Ashwani Kumar, Lovely Professional University</i>	
Unit 14:	More on Hashing	229
	<i>Ashwani Kumar, Lovely Professional University</i>	

Unit 01: Introduction

CONTENTS

- Objectives
- Introduction
- 1.1 Data Structure
- 1.2 Data Structure Operations
- 1.3 Abstract Data Type
- 1.4 Algorithm
- 1.5 Characteristics of an Algorithm
- 1.6 Types of Algorithms
- 1.7 Algorithm Complexity
- 1.8 Asymptotic Notations
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Question
- Further Readings

Objectives

After studying this unit, you will be able to:

- Describe basic concepts of data structure
- Learn Algorithm and its complexity
- Know Abstract data type
- Data structure types

Introduction

The static representation of a linear ordered list using an array wastes resources and, in some situations, causes overflows. We no longer want to pre-allocate memory to any linear list; instead, we want to allocate memory to elements as they are added to the list. This necessitates memory allocation that is dynamic.

Semantically data can exist in either of the two forms – atomic or structured. In most of the programming problems data to be read, processed and written are often related to each other. Data items are related in a variety of different ways. Whereas the basic data types such as integers, characters etc. can be directly created and manipulated in a programming language, the responsibility of creating the structured type data items remains with the programmers themselves. Accordingly, programming languages provide mechanism to create and manipulate structured data items.

A data structure is a type of storage that is used to organize and store data. It is a method of organizing data on a computer so that it may be easily accessible and modified.

It's critical to choose the correct data format for your project based on your requirements and project. If you wish to store data sequentially in memory, for example, you can use the Array data structure.

1.1 Data Structure

A data structure is a set of data values along with the relationship between the data values. Since, the operations that can be performed on the data values depend on what kind of relationships exists among them, we can specify the relationship amongst the data values by specifying the operations permitted on the data values. Therefore, we can say that a data structure is a set of values along with the set of operations permitted on them. It is also required to specify the semantics of the operations permitted on the data values, and this is done by using a set of axioms, which describes how these operations work, and therefore a data structure is made of:

1. A set of data values.
2. A set of functions specifying the operations permitted on the data values.
3. A set of axioms describing how these operations work.

Hence, we conclude that a data structure is a triple (D,F,A), where

1. D is a set of data values
2. F is a set of functions
3. A is a set of axioms

A triple (D, F, A) is referred to as an abstract data structure because it does not tell anything about its actual implementation. It does not tell anything about how these values will be physically represented in the computer memory and these functions will be actually implemented.

Therefore, every abstract data structure is required to be implemented, and the implementation of an abstract data structure requires mapping of the abstract data structure to be implemented into the data structure supported by the computer. For example, if the abstract data structure to be implemented is integer, then it can be implemented by mapping into bits which is a data structure supported by hardware. This requires that every integer data value is to be represented using suitable bit patterns and expressing the operations on integer data values in terms of operations for manipulating bits.

Data Structure mainly two types:

1. Linear type data structure
2. Non-linear type data structure

Linear data structure: A linear data structure traverses the data elements sequentially, in which only one data element can directly be reached. Ex: Arrays, Linked Lists

Arrays: An array is a collection of similar type of data items and each data item is called an element of the array.

The data type of the element may be any valid data type like char, int, float or double. — The individual elements of the array age are: — age[0], age[1], age[2], age[3], age[98], age[99].

Linked List: Linked list is a linear data structure which is used to maintain a list in the memory.

It can be seen as the collection of nodes stored at non-contiguous memory locations. Each node of the list contains a pointer to its adjacent node

Stack: Stack is a linear list in which insertion and deletions are allowed only at one end, called top.

A stack is an abstract data type, can be implemented in most of the programming languages. It is named as stack because it behaves like a real-world stack, for example: - piles of plates or deck of cards etc.

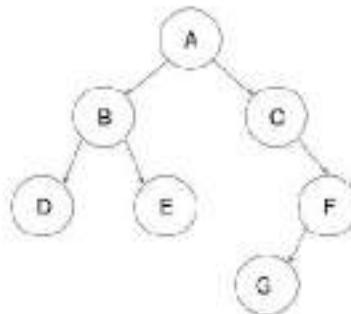
Queue is a linear list in which element can be inserted only at one end called rear and deleted only at other end called front.

It is abstract data structure, similar to stack. It is open at both end therefore if follows first-in-first-out (FIFO) technique for storing the data items.

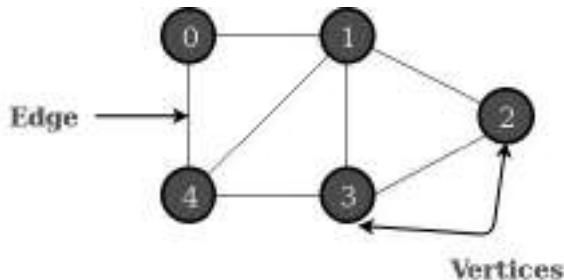
Non-linear data structure: Every data item is attached to several other data items in a way that is specific for reflecting relationships. The data items are not arranged in a sequential structure.

Ex: Trees, Graphs.

Trees: Trees are multilevel data structures with a hierarchical relationship among its elements known as nodes.



Graphs: Graphs can be defined as the pictorial representation of the set of elements (represented by vertices) connected by the links known as edges



Basic Terminology

Data: Data can be defined as an elementary value or the collection of values, for example, student's name and its id are the data about the student.

Group Items: Data items which have subordinate data items are called Group item, for example, name of a student can have first name and the last name.

Record: Record can be defined as the collection of various data items, for example, if we talk about the student entity, then its name, address, course and marks can be grouped together to form the record for the student.

Field: A File is a collection of various records of one type of entity, for example, if there are 60 students in the class, then there will be 20 records in the related file where each record contains the data about each student.

Need of Data Structures

As applications are getting complex and amount of data is increasing day by day, there may arise many problems:

Processor speed: To handle very large amount of data, high speed processing is required, but as the data is growing day by day to the billions of files per entity, processor may fail to deal with that much amount of data.

Data Search: Consider an inventory size of 100 items in a store, If our application needs to search for a particular item, it needs to traverse 100 items every time, results in slowing down the search process.

Multiple requests: If thousands of users are searching the data simultaneously on a web server, then there are the chances that a very large server can be failed during that process. To solve these problems data structures are used.

Basic Concept of Data

The memory (also called storage or core) of a computer is simply a group of bits (switches). At any instant of the computer's operation any particular bit in memory is either 0 or 1 (off or on).

The setting or state of a bit is called its value and that is the smallest unit of information. A set of bit values form data.

Some logical properties can be imposed on the data. According to the logical properties data can be segregated into different categories. Each category having unique set of logical properties is known as data type.

Data type are of two types:

1. Simple data type or elementary item like integer, character.
2. Composite data type or group item like array, structure, union.

Data structures are of two types:

1. *Primitive Data Structures*: Data can be structured at the most primitive level, where they are directly operated upon by machine-level instructions. At this level, data may be character or numeric, and numeric data may consist of integers or real numbers.

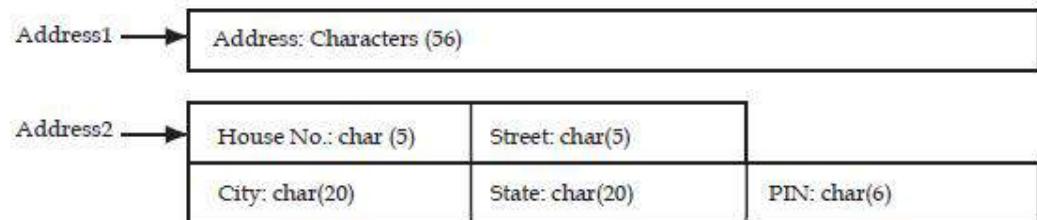
2. *Non-primitive Data Structures*: Non-primitive data structures can be classified as arrays, lists, and files.

An array is an ordered set which contains a fixed number of objects. No deletions or insertions are performed on arrays i.e. the size of the array cannot be changed. At best, elements may be changed.

A list, by contrast, is an ordered set consisting of a variable number of elements to which insertions and deletions can be made, and on which other operations can be performed. When a list displays the relationship of adjacency between elements, it is said to be linear; otherwise, it is said to be non-linear.

A file is typically a large list that is stored in the external memory of a computer. Additionally, a file may be used as a repository for list items (records) that are accessed infrequently.

From a real world perspective, very often we have to deal with structured data items which are related to each other. For instance, let us consider the address of an employee. We can take address to be one variable of character type or structured into various fields, as shown below:



As shown above Address1 is unstructured address data. In this form you cannot access individual items from it. You can at best refer to the entire address at one time. While in the second form, i.e., Address2, you can access and manipulate individual fields of the address - House No., Street, PIN etc. Given hereunder are two instances of the address1 and address2 variables.

1.2 Data Structure Operations

The data appearing in our data structure is processed by means of certain operations. The particular data structure that one chooses for a given situation depends largely on the frequency with which specific operations are performed. The following four operations play a major role:

1. Traversing: Accessing each record exactly once so that certain items in the record may be processed. (This accessing or processing is sometimes called 'visiting' the records.)
2. Searching: Finding the location of the record with a given key value, or finding the locations of all records, which satisfy one or more conditions.

3. Inserting: Adding new records to the structure.

4. Deleting: Removing a record from the structure.

Sometimes two or more data structure of operations may be used in a given situation; e.g., we may want to delete the record with a given key, which may mean we first need to search for the location of the record.

1.3 Abstract Data Type

Before we move to abstract data type let us understand what data type is. Most of the languages support basic data types viz. integer, real, character etc. At machine level, the data is stored as strings containing 1's and 0's. Every data type interprets the string of bits in different ways and gives different results. In short, data type is a method of interpreting bit patterns.

Every data type has a fixed type and range of values it can operate on. For example, an integer variable can hold values between the min and max values allowed and carry out operations like addition, subtraction etc. For character data type, the valid values are defined in the character set and the operations performed are like comparison, conversion from one case to another etc.

There are fixed operations, which can be carried out on them. We can formally define data types as a formal description of the set of values and operations that a variable of a given type may take. That was about the inbuilt data types. One can also create user defined data types, decide the range of values as well as operations to be performed on them. The first step towards creating a user defined data type or a data structure is to define the logical properties. A tool to specify the logical properties of a data type is Abstract Data Type.

Data abstraction can be defined as separation of the logical properties of the organization of programs' data from its implementation. This means that it states what the data should be like.

It does not consider the implementation details. ADT is the logical picture of a data type; in addition, the specifications of the operations required to create and manipulate objects of this data type.

While defining an ADT, we are not concerned with time and space efficiency or any other implementation details of the data structure. ADT is just a useful guideline to use and implement the data type.

An ADT has two parts:

1. Value definition
2. Operation definition.

Value definition is again divided into two parts:

1. Definition clause
2. Condition clause

As the name suggests the definition clause states the contents of the data type and condition clause defines any condition that applies to the data type. Definition clause is mandatory while condition clause is optional.

In operation definition, there are three parts:

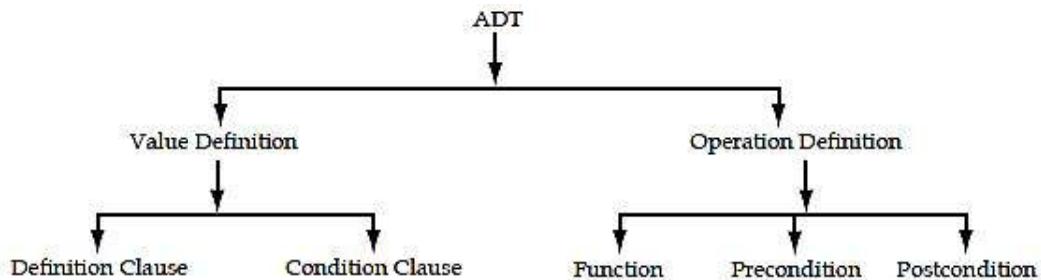
1. Function
2. Precondition
3. Postcondition

The *function* clause defines the role of the operation. If we consider the addition operation in integers the function clause will state that two integers can be added using this function. In general, precondition specifies any restrictions that must be satisfied before the operation can be applied.

This clause is optional. If we consider the division operation on integers then the precondition will state that the divisor should not be zero. So any call for divide operation, which does not satisfy this condition, will not give the desired output.

Precondition specifies any condition that may apply as a pre-requisite for the operation definition. There are certain operations that can be carried out if certain conditions are satisfied. For example, in case of division operation the divisor should never be equal to zero. Only if this condition is satisfied the division operation is carried out. Hence, this becomes a precondition. In that case & (ampersand) should be mentioned in the operation definition.

Postcondition specifies what the operation does. One can say that it specifies the state after the operation is performed. In the addition operation, the post condition will give the addition of the two integers.



Component of ADT

As an example, let us consider the representation of integer data type as an ADT. We will consider only two operations addition and division.

Value Definition

1. Definition clause: The values must be in between the minimum and maximum values specified for the particular computer.
2. Condition clause: Values should not include decimal point.

Operations

1. add (a, b)

Function: add the two integers a and b.

Precondition: no precondition.

Postcondition: output = a + b

2. Div (a, b)

Function: Divide a by b.

Precondition: b != 0

Postcondition: output = a/b.

There are two ways of implementing a data structure viz. static and dynamic. In static implementation, the memory is allocated at the compile time. If there are more elements than the specified memory then the program crashes. In dynamic implementation, the memory is allocated as and when required during run time.

Any type of data structure will have certain basic operations to be performed on its data like insert, delete, modify, sort, search etc. depending on the requirement. These are the entities that decide the representation of data and distinguish data structures from each other.

Let us see why user defined data structures are essential. Consider a problem where we need to create a list of elements. Any new element added to the list must be added at the end of the list and whenever an element is retrieved, it should be the last element of the list. One can compare this to a pile of plates kept on a table. Whenever one needs a plate, the last one on the pile is taken and if a plate is to be added on the pile, it will be kept on the top. The description wants us to implement a stack. Let us try to solve this problem using arrays.

We will have to keep track of the index of the last element entered in the list. Initially, it will be set to -1. Whenever we insert an element into the list, we will increment the index and insert the value into the new index position. To remove an element, the value of current index will be the output

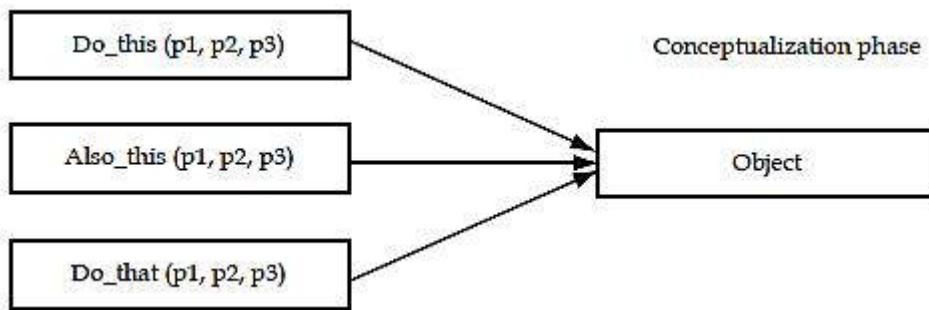
and the index will be decremented by one. In the above representation, we have satisfied the insertion and deletion conditions.

Using arrays we could handle our data properly, but arrays do allow access to other values in addition to the top most one. We can insert an element at the end of the list but there is no way to ensure that insertion will be done only at the end. This is because array as a data structure allows access to any of its values. At this point we can think of another representation, a list of elements where one can add at the end, remove from the end and elements other than the top one are not accessible. As already discussed, this data structure is called as STACK. The insertion operation is known as push and removal as pop. You can try to write an ADT for stacks.

Another situation where we would like to create a data structure is while working with complex numbers. The operations add, subtract division and multiplication will have to be created as per the rules of complex numbers. The ADT for complex numbers is given below. Only addition and multiplication operations are considered here, you can try to write the remaining operations.

Abstract Data Type (ADT)

1. A framework for an object interface
2. What kind of stuff it'd be made of (no details)?
3. What kind of messages it would receive and kind of action it'll perform when properly triggered?



From this we figure out

1. Object make-up (in terms of data)
2. Object interface (what sort of messages it would handle?)
3. How and when it should act when triggered from outside (public trigger) and by another object friendly to it?

These concerns lead to an ADT – a definition for the object.

An Abstract Data Type (ADT) is a set of data items and the methods that work on them.

An implementation of an ADT is a translation into statements of a programming language, of the declaration that defines a variable to be of that ADT, plus a procedure in that language for each operation of the ADT. An implementation chooses a data structure to represent the ADT; each data structure is built up from the basic data types of the underlying programming language.

Thus, if we wish to change the implementation of an ADT, only the procedures implementing the operations would change. This change would not affect the users of the ADT.

Although the terms 'data type', 'data structure' and 'abstract data type' sound alike, they have different meanings. In a programming language, the data type of a variable is the set of values that the variable may assume. For example, a variable of type Boolean can assume either the value true or the value false, but no other value. An abstract data type is a mathematical model, together with various operations defined on the model. As we have indicated, we shall design algorithms in terms of ADTs, but to implement an algorithm in a given programming language.

we must find some way of representing the ADTs in terms of the data types and operators supported by the programming language itself. To represent the mathematical model underlying an ADT, we use data structures, which are a collection of variables, possibly of several data types, connected in various ways.

The cell is the basic building block of data structures. We can picture a cell as a box that is capable of holding a value drawn from some basic or composite data type. Data structures are created by giving names to aggregates of cells and (optionally) interpreting the values of some cells as representing relationships or connections (e.g., pointers) among cells.

1.4 Algorithm

Algorithm is set of rules/ instructions that step-by-step define how a work is to be executed upon in order to get the expected results.

systematic procedure that produces in a finite number of steps the answer to a question or the solution of a problem.

Computer algorithms work via input and output. They take the input and apply each step of the algorithm to that information to generate an output.

E.g. a search engine is an algorithm that takes a search query as an input and searches its database for items relevant to the words in the query. It then outputs the results.

Financial companies use algorithms in areas such as loan pricing, stock trading, asset-liability management, and many automated functions. For example, algorithmic trading, known as algo trading, is used for deciding the timing, pricing, and quantity of stock orders. Also referred to as automated trading or black-box trading, algo trading uses computer programs to buy or sell securities at a pace not possible for humans.

Computer algorithms make life easier by trimming the time it takes to manually do things. In the world of automation, algorithms allow workers to be more proficient and focused. Algorithms make slow processes more proficient. In many cases, especially in automation, algos can save companies money.

1.5 Characteristics of an Algorithm

- Well defined Input and output
- Clear and Unambiguous
- Finite-ness
- Feasible
- Language Independent

Input and output should be defined precisely.

Each step in the algorithm should be clear and unambiguous.

Algorithms should be most effective among many different ways to solve a problem.

An algorithm shouldn't include computer code. Instead, the algorithm should be written in such a way that it can be used in different programming languages.

The algorithm must be finite, i.e. it should not end up in an infinite loops or similar.

The algorithm must be simple, generic and practical, such that it can be executed upon will the available resources. It must not contain some future technology, or anything.

The Algorithm designed must be language-independent, i.e. it must be just plain instructions that can be implemented in any language, and yet the output will be same, as expected.

1.6 Types of Algorithms

Algorithms are categorized based on the concepts that they use to accomplish a task.

- Divide and conquer algorithms
- Brute force algorithms
- Greedy algorithms
- Backtracking algorithms
- Randomized algorithms



Example:

Step 1: Start

Step 2: Declare variables num1, num2 and sum.

Step 3: Read values num1 and num2.

Step 4: Add num1 and num2 and assign the result to sum.

Sum = num1+num2

Step 5: Display sum

Step 6: Stop

1.7 Algorithm Complexity

Space Complexity

Time Complexity

Space Complexity: Space complexity of an algorithm refers to the amount of memory that this algorithm requires to execute and get the result. This can be for inputs, temporary operations, or outputs.

Fixed Part: This refers to the space that is definitely required by the algorithm. For example, input variables, output variables, program size, etc.

Variable Part: This refers to the space that can be different based on the implementation of the algorithm. For example, temporary variables, dynamic memory allocation, recursion stack space, etc.

Time Complexity: Time complexity of an algorithm refers to the amount of time that this algorithm requires to execute and get the result. This can be for normal operations, conditional if-else statements, loop statements, etc.

Constant time part: Any instruction that is executed just once comes in this part. For example, input, output, if-else, switch, etc.

Variable Time Part: Any instruction that is executed more than once, say n times, comes in this part. For example, loops, recursion, etc.

1.8 Asymptotic Notations

To measure the efficiency of an algorithm asymptotic analysis is used.

The efficiency of an algorithm depends on the amount of time, storage and other resources required to execute the algorithm.

Performance of algorithm is change with different type of inputs.

The study of change in performance of the algorithm with the change in the order of the input size is defined as asymptotic analysis.

Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.

Types of asymptotic notations

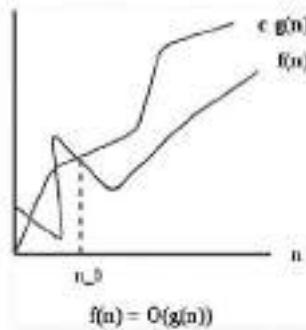
There are three major asymptotic notations

- Big-O notation
- Omega notation
- Theta notation

Big-O notation represents the upper bound of the running time of an algorithm. It gives the worst-case complexity of an algorithm.

$O(n)$ is useful when we only have an upper bound on the time complexity of an algorithm.

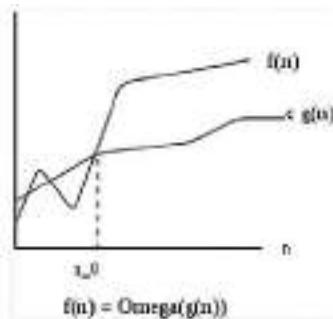
It is widely used to analyses an algorithm as we are always interested in the worst-case scenario.



$$\begin{aligned} O(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \\ \text{such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \} \end{aligned}$$

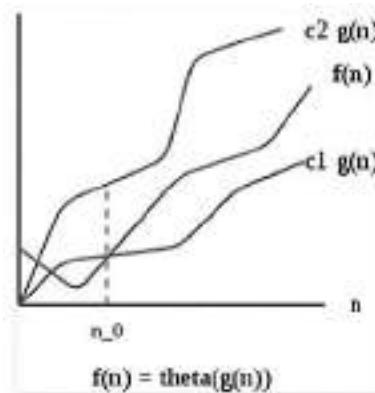
Omega notation represents the lower bound of the running time of an algorithm. It provides the best-case complexity of an algorithm.

Omega Notation can be useful when we have lower bound on time complexity of an algorithm. Omega notation is the least used notation among all three.



$$\begin{aligned} \Omega(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and} \\ n_0 \text{ such that } 0 \leq c^*g(n) \leq f(n) \text{ for all } n \geq n_0 \} \end{aligned}$$

Theta notation encloses the function from above and below. It represents the upper and the lower bound of the running time of an algorithm, it is used for analysing the average-case complexity of an algorithm.



$$\begin{aligned} \Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2 \text{ and } n_0 \text{ such} \\ \text{that } 0 \leq c_1^*g(n) \leq f(n) \leq c_2^*g(n) \text{ for all } n \geq n_0 \} \end{aligned}$$

Properties of Asymptotic Notations

If $f(n)$ is $O(g(n))$ then $a*f(n)$ is also $O(g(n))$; where a is a constant.

General Properties

If $f(n)$ is $O(g(n))$ then $a*f(n)$ is also $O(g(n))$; where a is a constant.

Transitive Properties

If $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$ then $f(n) = O(h(n))$

Reflexive Properties

If $f(n)$ is given then $f(n)$ is $O(f(n))$

Symmetric Properties

If $f(n)$ is $\Theta(g(n))$ then $g(n)$ is $\Theta(f(n))$

Transpose Symmetric Properties

If $f(n)$ is $O(g(n))$ then $g(n)$ is $\Omega(f(n))$

Summary

- Data Structure is method or technique to data organization, management, and storageformat in the computer so we can perform operations on the stored data more efficiently.
- Data structure is a combination of one or more basic data types to form a single addressable data type.
- An algorithm is a finite set of instructions which, when followed, accomplishes a particular task, the termination of which is guaranteed under all cases, i.e. the termination is guaranteed for every input.
- The instructions must be unambiguous and the algorithm must produce the output within a finite number of executions of its instructions.
- Abstract data type (ADT) is a mathematical model with a collection of operations defined on that model. Although the terms 'data type', 'data structure' and 'abstract data type' sound alike, they have different meanings.

Keywords

- *Data*: Data can be defined as an elementary value or the collection of values, for example, student's name and its id are the data about the student.
- *Group Items*: Data items which have subordinate data items are called Group item, for example, name of a student can have first name and the last name.
- *Linear Data Structure*: A linear data structure traverses the data elements sequentially, in whichonly one data element can directly be reached.
- *Non-linear Data Structure*: Every data item is attached to several other data items in a way thatis specific for reflecting relationships. The data items are not arranged in a sequential structure.
- *Searching*: Finding the location of the record with a given key value, or finding the locations ofall records, which satisfy one or more conditions.
- *Traversing*: Accessing each record exactly once so that certain items in the record may beprocessed.

Self Assessment

1. Which is type of data structure.

- A. Primitive
 - B. Non-primitive
 - C. Both primitive and non-primitive
 - D. None of above
2. Which of the following is linear data structure?
- A. Trees
 - B. Arrays
 - C. Graphs
 - D. None of these
3. Which of the following is non-linear data structure?
- A. Array
 - B. Linked lists
 - C. Stacks
 - D. None of these
4. User defined data type is also called?
- A. Primitive
 - B. Identifier
 - C. Non-primitive
 - D. None of these
5. Stack is based on which principle
- A. FIFO
 - B. LIFO
 - C. Push
 - D. None of the Above
6. What are the characteristics of an Algorithm.
- A. Clear and Unambiguous
 - B. Finite-ness
 - C. Feasible
 - D. All of above
7. A procedure for solving a problem in terms of action and their order is called as
- A. Program instruction
 - B. Algorithm
 - C. Process
 - D. Template
8. Algorithm can be represented as

- A. Pseudocode
 - B. Flowchart
 - C. None of the above
 - D. Both Pseudocode and Flowchart
9. What are the different types of Algorithms?
- A. Brute force algorithms
 - B. Greedy algorithms
 - C. Backtracking algorithms
 - D. All of these
10. Which is algorithm complexity.
- A. Space Complexity
 - B. Time Complexity
 - C. Both space and time complexity
 - D. None of aboveone of these
11. Which one is asymptotic notations?
- A. Big-O notation
 - B. Omega notation
 - C. Theta notation
 - D. All of above
12. Big-O Notation represents...
- A. Space complexity
 - B. Upper bound of the running time of an algorithm
 - C. Lower bound of the running time of an algorithm
 - D. None of above
13. Omega Notation (Ω -notation) represents....
- A. Upper bound of the running time of an algorithm
 - B. Space complexity
 - C. Lower bound of the running time of an algorithm
 - D. None of above
14. Which is property of Asymptotic Notations?
- A. Reflexive
 - B. Symmetric
 - C. Transpose Symmetric
 - D. All of these
15. Abstract Data Type having.

- A. Value definition
- B. Operation definition
- C. Both value and operation definition
- D. None of above.

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. C | 2. B | 3. D | 4. C | 5. B |
| 6. D | 7. B | 8. D | 9. D | 10. C |
| 11. D | 12. B | 13. C | 14. D | 15. C |

Review Question

1. Define data structure and its application.
2. What are the advantages of data structure?
3. Discuss abstract data type.
4. What is significance of space and time complexity in algorithm.
5. Explain different types of algorithms.
6. Discuss Asymptotic notations with example.
7. Define record and file.



Further Readings

- Data Structures and Algorithms; Shi-Kuo Chang; World Scientific.
- Data Structures and Efficient Algorithms, Burkhard Monien, Thomas Ottmann, Springer.
- Mark Allen Weis: Data Structure & Algorithm Analysis in C Second Edition. Addison-Wesley publishing
- Thomas H. Cormen, Charles E. Leiserson & Ronald L. Rivest: Introduction to Algorithms. Prentice-Hall of India Pvt. Limited, New Delhi
- Timothy A. Budd, Classic Data Structures in C++, Addison Wesley.

Unit 02: Arrays vs Linked Lists

CONTENTS

- Objectives
- Introduction
- 2.1 Arrays
- 2.2 Types of Arrays
- 2.3 Types of Array Operations
- 2.4 Linked list
- 2.5 Types of linked list
- Summary
- Keywords
- Self-Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to:

- Learn basic concepts of arrays
- Understand the basics of linked list
- Describe the types of array operations
- Discuss the operations of linked lists

Introduction

A data structure consists of a group of data elements bound by the same set of rules. The data elements also known as members are of different types and lengths. We can manipulate data stored in the memory with the help of data structures. The study of data structures involves examining the merging of simple structures to form composite structures and accessing definite components from composite structures. An array is an example of one such composite data structure that is derived from a primitive data structure.

An array is a set of similar data elements grouped together. Arrays can be one-dimensional or multidimensional. Arrays store the entries sequentially. Elements in an array are stored in continuous locations and are identified using the location of the first element of the array.

2.1 Arrays

An array is a data type, much like a variable as both array and variable hold information. However, unlike a variable, an array can hold several pieces of data called elements. Arrays can hold any type of data, which includes string, integers, Boolean, and so on. An array can also handle other variables as well as other arrays. An integer index identifies the individual elements of an array.

Arrays are allocated the memory in a strictly contiguous fashion. The simplest array is one-dimensional array which is a list of variables of same data type. An array of one-dimensional arrays

[Subject]

is called a two-dimensional array; array of two-dimensional arrays is three-dimensional array and so on.

The members of the array can be accessed using positive integer values (indicating their order in the array) called subscript or index.

200	120	-78	100	0
-----	-----	-----	-----	---

a[0] a[1] a[2] a[3] a[4]

The description of this array is listed below:

Name of the array : a

Data type of the array : integer

Number of elements : 5

Valid index values : 0, 1, 2, 3, 4

Value stored at the location a[0] : 200

Value stored at the location a[1] : 120

Value stored at the location a[2] : -78

Value stored at the location a[3] : 100

Value stored at the location a[4] : 0

Initializing an Array

We can initialize an array by assigning values to the elements during declaration. We can access the element by specifying its index. While initializing an array, the initial values are given sequentially separated by commas and enclosed in braces.



Example:

Consider the elements 10, 20, 30, and 40. The array can be represented as:

a[4]={10, 20, 30, 40}

The elements can be stored in an array as shown below:

a[0] = 10

a[1] = 20

a[2] = 30

a[3] = 40

The element 20 can be accessed by referencing a[1].

Now, consider n number of elements in an array. Hence, to access any element within the array, we use a[i], where i is the value between 0 to n-1.

The corresponding code used in C language to read n number of integers in an array is:

```
for(i= 0; i<n; i++)
{
    scanf("%d",&a[i]);
}
```

Array Initialization in its Declaration

A variable is initialized in its declaration.



Example:

```
int value = 10;
```

Here, the value 10 is called an initializer.

Similar to a variable, we can initialize an array at the time of its declaration. The following example shows an array initialization.



Example:

```
int a[5] = {10, 11, 12, 13, 14};
```

In this declaration, a[0] is initialized to 10, a[1] is initialized to 11, and so on. There must be at least one

initial value between braces. If the number of initialized array elements is lesser than the declared size,

then the remaining array elements are assigned the value 0.

If we provide all the array elements during initialization, it is not necessary to specify the array size. The compiler automatically counts the number of elements and reserves the space in the memory for the array.



Example:

```
int a[] = {10, 20, 30, 40};
```

Here the compiler reserves four spaces for array a.

2.2 Types of Arrays

The elements in an array are referred either by a single subscript or by two or more subscripts. Hence, the arrays are of two types namely, one-dimensional array and multidimensional array, based on the subscript referred. A two-dimensional array is also a type of multidimensional array. When the array is referred by a single subscript, then it is known as one-dimensional array or linear array. When the array is referred by two subscripts, it is known as a two-dimensional array. Some programming languages allow more than two or three subscripts and these arrays are known as multidimensional arrays.

According to the number of subscripts required to access an array element, arrays can be of

following types:

1. One-dimensional array
2. Multi-dimensional array

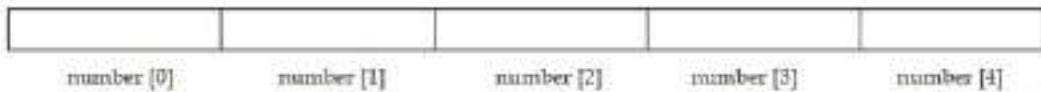
Linear Array

A linear or one-dimensional array is a structured collection of elements (often called array elements). It can be accessed individually by specifying the position of each element by an index value.

Example: If we want to store a set of five numbers by an array variable number. Then it will be accomplished in the following way:

```
int number [5];
```

This declaration will reserve five contiguous memory locations capable of storing an integer type value each, as shown below:



Now let us see how individual elements of linear array are accessed. The syntax for accessing an array component is:

ArrayName[IndexExpression]

The IndexExpression must be an integer value. The integer value can be of char, short int, long int, or

Boolean value because these are integral data types. The simplest form of index expression is a constant.



Example:

If we consider an array number[25], then,

number[0] specifies the 1st component of the array

number[1] specifies the 2nd component of the array

number[2] specifies the 3rd component of the array

number[3] specifies the 4th component of the array

number[4] specifies the 5th component of the array

.

.

number[23] specifies the 2nd

To store and print values from the number array, we can perform the following:

```
for(int i=0; i< 25; i++)
{
    number[i]=i; // Storing a number in each array element
    printf("%d", number[i]); //Printing the value
}
```

Multidimensional Array

Multidimensional arrays are also known as "arrays of arrays." Programming languages often need to store and manipulate two or more dimensional data structures such as, matrices, tables, and so on. When programming languages use two subscripts they are known as two-dimensional arrays. One subscript denotes a row and the other denotes a column.

The declaration of two-dimension array is as follows:

data_type array_name[row_size][column_size];



Example:

int m[5][10]

Here, m is declared as a two dimensional array having 5 rows (numbered from 0 to 4) and 10 columns (numbered from 0 to 9). The first element of the array is m[0][0] and the last row last column is m[4][9]

Now let us discuss a three-dimensional array. A three-dimensional array is considered as an array of two-dimensional arrays.



Example:

A three dimensional array is created as follows:

```
int bigArray[ ][ ][ ] = new int [10][10][4];
```

This will create an array named bigArray containing 400 integers. We can access any element of this array by using 3 indices.



Example:

Suppose we want to assign a value 312 to the element at position 3 down, 7 across, and 2 in, then we write it as:

```
bigArray [2][6][1] = 312;
```

Initialization of Multidimensional Arrays

Like the one dimension arrays, two-dimensional arrays are also initialized by declaring a list of initial values enclosed in braces.



Example:

```
int table[2][3]={0,0,0,1,1,1};
```

The table array initializes the elements of first row to 0 and the second row to

1. The initialization is done row by row. The above statement can be equivalently written as:

```
int table[2][3]={ {0,0,0}, {1,1,1} }
```

Three or four-dimensional arrays are more complicated. They can also be initialized by declaring a list of initial values enclosed in braces.



Example:

```
int table[3][3][3]={1,2,3,4,5 6,7,8,.....27 };
```

This will create an array named table containing 27 integers. We can access any element of this array by using 3 indices.

The method to access table[1][1][1], is as shown below:

The values for array - table[3][3][3] are as follows:

```
{1, 2, 3}
```

```
{4, 5, 6}
```

```
{7, 8, 9}
```

```
{10, 11, 12}
```

```
{13, 14, 15}
```

```
{16, 17, 18}
```

```
{19, 20, 21}
```

```
{22, 23, 24}
```

```
{25, 26, 27}
```

The values in the array can be accessed using three for loops. The loop contains three variables i, j, and k respectively. This is as shown below:

```
for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
        for(k=0;k<3;k++)
    {
        printf("%d\t",table[i][j][k]);
    }
}
```

```

    }
    printf("\n");
}
}

printf("%d", table[1][1][1]);

```

For every iteration of the i, j and k loops, the values printed are:

```

[0][0][0] = 1
[0][0][1] =2
[0][0][2] =3
[1][1][1] =14

```

2.3 Types of Array Operations

The operations performed on an array, are

1. Adding operation
2. Sorting operation
3. Searching operation
4. Traversing operation

Adding Operation

Adding elements into an array is known as insertion. The insertion of data elements is done at the end of an array. This is possible only if there is enough space in the array to add the additional elements. The elements can also be inserted in the middle of the array. Here, the average half of the array elements is moved to the next location to empty the block of memory, and to accommodate the new element.

Algorithm for Inserting an Element into an Array

Let a be an array of size N and I be the array index. Algorithm to insert an element in the MthPosition of the array a is as follows

1. Start
2. read a[N], I<-0
3. repeat for I=N to M (Decrement I by one)
4. a[I+1]<- a[I]
5. a[M]<-ELEMENT
6. M<-M+1
7. Stop

The below program illustrates the concept of inserting an element into a one-dimensional array.



Example:

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int n, i, data, po_indx, a[50]; //Variable declaration

```

```

clrscr();
printf("Enter number of elements in the array\n");
/*Get the number of elements to be added to the array from the user*/
scanf("%d", &n);
printf("\nEnter %d elements\n\n", n); //Print the number of elements
for(i=0;i<n;i++) //Iterations using for loop
scanf("%d",&a[i]); //Accepting the values in the array
printf("\nEnter a data to be inserted\n");
scanf("%d",&data); //Reads the data added by user
printf("\nEnter the position of the item \n");
scanf("%d",&po_indx); //Reads the position where the data is inserted
/* Checking if the position is greater than the size of the array*/
if(po_indx-1>n)
printf("\nposition not valid\n"); //If the condition is true this will be printed
else //If the condition is false the 'else' part will get executed
{
for(i=n;i>=po_indx;i--) //Iterations using for loop
a[i]=a[i-1]; //Value of a[i-1] is assigned to a[i]
/*Value of data will be assigned to [po_indx-1] position*/
a[po_indx-1]=data;
n=n+1; //Incrementing the value of n
printf("\nArray after insertion\n"); //Print the array list after insertion
for(i=0;i<n;i++) //Use for loop and
printf("%d\t",a[i]); //Print the final array after insertion
}
getch(); //Display characters on screen
}

```

Output:

Enter number of elements in the array

5

Enter 5 elements

15 20 32 45 62

Enter a data to be inserted

77

Enter the position of the item

2

Array after insertion

15 77 20 32 45 62

In this example:

1. First, the header files are included using #include directive.

2. Then, the index, array, and the variables are declared.
3. The program accepts the number of elements in the array.
4. Using a for loop, the values are accepted and stored in the array.
5. Then, the program accepts the data along with the position where it needs to be inserted.
6. If the position to be inserted is greater than the number of elements ($po_idx-1 > n$) then the program displays "position is not valid". Otherwise, the program by means of a for loop, checks whether $i \geq po_idx$ is true and assigns the $a[i-1]$ value to $a[i]$.
7. Then data is assigned to $a[po_idx-1]$.
8. Then, the program increments the number of elements and prints the array after insertion.
9. getch() prompts the user to press a key and the program terminates.

Sorting Operation

Sorting operation arranges the elements of a list in a certain order. Efficient sorting is important for optimizing the use of other algorithms that require sorted lists to work correctly.

Sorting an array efficiently is quite complicated. There are different sorting algorithms to perform the task of sorting, but here we will discuss only Bubble Sort.

Bubble Sort

Bubble sort is a simple sorting technique when compared to other sorting techniques. The bubble sort algorithm starts from the very first element of the data set. In order to sort elements in the ascending order, the algorithm compares the first two elements of the data set. If the first element is greater than the second, then the numbers are swapped.

This process is carried out for each pair of adjacent elements to the end of the data set until no swaps occur on the last pass. This algorithm's average and worst case performance is $O(2n)$ as it is rarely used to sort large, unordered data sets.

Bubble sort can always be used to sort a small number of items where efficiency is not a high priority. Bubble sort may also be effectively used to sort a partially sorted list.

Algorithm for Sorting an Array

Let A be an array containing data with N elements. This algorithm sorts the elements in A as follows:

1. Start
2. Repeat Steps 3 and 4 for $K = 1$ to $N-1$
3. Set PTR := 1 [Initializes pass pointer PTR]
4. Repeat while $PTR \leq N - K$: [Executes pass]
 - If $A[PTR] > A[PTR+1]$, then:
 - Interchange $A[PTR]$ and $A[PTR + 1]$
 - [End of If structure]
 - Set PTR := PTR+1
 - [End of inner loop]
5. Exit

In the algorithm, there is an inner loop, which is controlled by the variable PTR, and an index K controls the outer loop. K is used as a counter and PTR is used as an index.

The below program illustrates the concept of sorting an array using bubble sort.



Example:

```
#include <stdio.h>
#include <conio.h>

int A[8] = {55, 22, 2, 43, 12, 8, 32, 15}; //Declaring the array with 8 elements
int N = 8; //Size of the array
void BUBBLE (void); //BUBBLE Function declaration
void main()
{
    int i; //Variable declaration
    clrscr();
    /*Printing the values in the array*/
    printf("\n\nValues present in array A =");
    for (i=0; i<8; i++) //Iterations using for loop
        printf(" %d, ", A[i]); //Printing the array
    BUBBLE(); //BUBBLE function is called
    /*Printing the values from the array after sorting*/
    printf("\n\nValues present in the array after sorting =");
    for (i=0; i<8; i++) //Iterations
        printf(" %d, ", A[i]); // Printing the array after sorting
    getch(); // waits for a key to be pressed
}

void BUBBLE(void) //BUBBLE Function definition
{
    int K, PTR, TEMP; //Declaration variables
    for(K=0; K <= (N-2); K++) //Iterations
    {
        PTR = 0; //Assign 0 to variable PTR
        while(PTR <= (N-K-1-1)) //Checking if PTR <= (N-K-1-1)
        {
            /* Checking if the element at A[PTR] is greater than A[PTR+1]*/
            if(A[PTR] > A[PTR+1])
            {
                TEMP = A[PTR];
                A[PTR] = A[PTR+1];
                A[PTR +1] = TEMP;
            }
            /*Increment the array index*/
            PTR = PTR+1;
        }
    }
}
```

[Subject]

```
}
```

Output:

Values present in A[8] = 55, 22, 2, 43, 12, 8, 32, 15

Values present in A[8] after sorting = 2, 8, 12, 15, 22, 32, 43, 55

In this example:

1. First, the header files are included using #include directive.
2. Then, the array A is declared globally along with the array elements and the size.
3. Then, inside the main function the variable i is declared.
4. The values in the array are printed using a for loop.
5. Next, the Bubble function is called. The sorting operation is carried out and values present in the array are printed.
6. getch() prompts the user to press a key. Then the program terminates.
7. In The BUBBLE function the variables K, PTR and TEMP are declared as integers.
8. PTR is set to 0.
9. Within the while loop the adjacent array elements are compared. If the element at a lower position is greater than the element at the next position, both the elements are interchanged.
10. The array index is then incremented.

Searching Operation

Searching is an operation used for finding an item with specified properties among a collection of items. In a database, the items are stored individually as records, or as elements of a search space addressed by a mathematical formula or procedure. The mathematical formula or procedure may be the root of an equation containing integer variables.

Search operation is closely related to the concept of dictionaries. Dictionaries are a type of data structure that support operations such as, search, insert, and delete.

Computer systems are used to store large amounts of data. From these large amount of data, individual records are retrieved based on some search criterion. The efficient storage of data is an important issue to facilitate fast searching.

There are many different searching techniques or algorithms. The selection of algorithm depends on the way the information is organized in memory. Now, we will discuss linear searching technique.

Algorithm for Linear search

Let A be a linear array with N elements and ITEM be the given item of information. The search algorithm will find the location LOC of ITEM in A or sets LOC := 0 if the search fails. The algorithm is as follows:

1. Start
2. [Insert ITEM at the end of A.] Set A[N+1] := ITEM
3. [Initialize counter.] Set LOC := 1
4. [Search for ITEM.]
 - (a) Repeat while A[LOC] ≠ ITEM:
 - (b) Set LOC := LOC + 1
- [End of loop]
5. [Successful?] If LOC = N + 1, then: Set LOC := 0

6. Exit

Traversing Operation

Traversing an array refers to moving in inward and outward direction to access each element in an array. To traverse an array, one can use for loop. The array elements are accessed using an array index or a pointer of type similar to that of array elements. To access the elements using a pointer, the pointer must be initialized with the base address of the array. Traversing operation also involves printing the elements in an array.

Algorithm for Traversal Operation

Let X be an array of size N. You need to traverse through the array and perform the required operations on each element of the array. Let the required operation be OP. Here, i is the array index and the lower bound starts with 0. The algorithm for traversing a given array is as follows:

1. Start
2. read X[N], i=0
3. repeat for I = 0, 1, 2.....N
OP on X[i]
4. Stop



Example:

```
#include<stdio.h>
#include<conio.h>
#define SIZE 20 //Define array size
void main()
{
float sum(float[], int); //Function declaration
float x[SIZE], Sum_total=0.0;
int i, n; //Variable declaration
clrscr();
printf("Enter the number of elements in array\n");
scanf(" %d", &n); //Reads the data added by user
printf("Enter %d elements:\n", n); //Printing the values in the array
for(i=0; i<n; i++) //Iterations using for loop
/* Input the elements of the array (Traverse operation)*/
scanf(" %f", &x[i]);
printf("The elements of array are:\n\n"); //Printing the elements of the array
for(i=0; i<n; i++) //Iterations using for loop
/*print the elements of array in floating point form(Traverse operation)*/
printf(" %.2f\t", x[i]);
/*Call the function sum and store the value returned in Sum_total*/
Sum_total = sum(x, n);
/*Printing the sum*/
printf("\n\nSum of the given array is: %.2f\n", Sum_total);
getch(); //wait until a key is pressed
}
```

```

float sum(float x[], int n) //Function declaration
{
    int i; //Variable declaration
    float total=0.0; //the variable total is set to 0.0
    for(i=0; i<n; i++) //Iterations
        total+=x[i]; //each element x[i] is added to the value of total
    return(total); //Returning the total value
}

```

Output:

Enter the number of elements in array

5

Enter 5 elements

14 15 16 17 18

The elements of array are:

14.00 15.00 16.00 17.00 18.00

Sum of the given array is: 80.00

In this example:

1. First, the header files are included using #include directive.
2. Using the #define directive, the array size, SIZE, is set to 20.
3. In the main() function, the function sum and the variables are declared.
4. A for loop is used accept the elements of the array.
5. The next for loop prints the elements of the array.
6. The program calls the sum() function to add all the elements of the array. The value returned by the sum() function is stored in the variable Sum_total.
7. The program then prints the sum of the elements of the array.
8. getch() prompts the user to press a key to exit the program.
9. The function sum that accepts two arguments and returns a float value is defined. The function sum does the following steps:
 - (a) Initializes an integer variable i.
 - (b) Initializes a float variable total and assigns 0.0 to it.
 - (c) Adds the elements of the array using a for loop and the result is stored in total.
 - (d) Finally, it returns the value of total.

2.4 Linked list

Linked lists are the most common data structures. They are referred to as an array of connected objects where data is stored in the pointer fields. Linked lists are useful when the number of elements to be stored in a list is indefinite.

Concept of Linked Lists

An array is represented in memory using sequential mapping, which has the property that elements are fixed distance apart. But this has the following disadvantage. It makes insertion or

deletion at any arbitrary position in an array a costly operation, because this involves the movement of some of the existing elements.

When we want to represent several lists by using arrays of varying size, either we have to represent each list using a separate array of maximum size or we have to represent each of the lists using one single array. The first one will lead to wastage of storage, and the second will involve a lot of data movement.

So we have to use an alternative representation to overcome these disadvantages. One alternative is a linked representation. In a linked representation, it is not necessary that the elements be at a fixed distance apart. Instead, we can place elements anywhere in memory, but to make it a part of the same list, an element is required to be linked with a previous element of the list. This can be done by storing the address of the next element in the previous element itself. This requires that every element be capable of holding the data as well as the address of the next element. Thus every element must be a structure with a minimum of two fields, one for holding the data value, which we call a data field, and the other for holding the address of the next element, which we call link field.

Therefore, a linked list is a list of elements in which the elements of the list can be placed anywhere in memory, and these elements are linked with each other using an explicit link field, that is, by storing the address of the next element in the link field of the previous element.

This program uses a strategy of inserting a node in an existing list to get the list created. An insert function is used for this. The insert function takes a pointer to an existing list as the first parameter, and a data value with which the new node is to be created as a second parameter, creates a new node by using the data value, appends it to the end of the list, and returns a pointer to the first node of the list. Initially the list is empty, so the pointer to the starting node is NULL.

Therefore, when insert is called first time, the new node created by the insert becomes the start node. Subsequently, the insert traverses the list to get the pointer to the last node of the existing list, and puts the address of the newly created node in the link field of the last node, thereby appending the new node to the existing list. The main function reads the value of the number of nodes in the list. Calls iterate that many times by going in a while loop to create the links with the specified number of nodes.

2.5 Types of linked list

Single linked list

Double linked list

Circular linked list

A doubly-linked list is a linked data structure that consists of a set of sequentially linked records called nodes. Each node contains two fields, called links, that are references to the previous and to the next node in the sequence of nodes.

In the single linked list each node provides information about where the next node is in the list. It faces difficulty if we are pointing to a specific node, then we can move only in the direction of the links. It has no idea about where the previous node lies in memory. The only way to find the node which precedes that specific node is to start back at the beginning of the list. The same problem arises when one wishes to delete an arbitrary node from a single linked list. Since in order to easily delete an arbitrary node one must know the preceding node. This problem can be avoided by using Doubly Linked List, we can store in each node not only the address of next node but also the address of the previous node in the linked list. A node in Doubly Linked List has three fields

1. Data
2. Previous Link
3. Next Link



Implementation of Doubly Linked List

Structure of a node of Doubly Linked List can be defined as:

```
struct node
{
    int data;
    struct node *llink;
    struct node *rlink;
}
```

Circular Linked List

Circular Linked List is another remedy for the drawbacks of the Single Linked List besides Doubly Linked List. A slight change to the structure of a linear list is made to convert it to circular linked list; link field in the last node contains a pointer back to the first node rather than a Null.

Representation of Linked List

Because each node of an element contains two parts, we have to represent each node through a structure.

While defining linked list we must have recursive definitions:

```
struct node
{
    int data;
    struct node * link;
}
```

Here, link is a pointer of struct node type i.e. it can hold the address of variable of struct node type. Pointers permit the referencing of structures in a uniform way, regardless of the organization of the structure being referenced. Pointers are capable of representing a much more complex relationship between elements of a structure than a linear order.

Initialization:

```
main()
{
    struct node *p, *list, *temp;
    list = p = temp = NULL;
    .
    .
    .
}
```



Example:

Program:

```
# include <stdio.h>
# include <stdlib.h>
struct node
{
    int data;
    struct node *link;
```

```
};

struct node *insert(struct node *p, int n)
{
    struct node *temp;
    /* if the existing list is empty then insert a new node as the
       starting node */
    if(p==NULL)
    {
        p=(struct node *)malloc(sizeof(struct node)); /* creates new
           node data value passes
           as parameter */
        if(p==NULL)
        {
            printf("Error\n");
            exit(0);
        }
        p-> data = n;
        p-> link = p; /* makes the pointer pointing to itself because
           it is a circular list*/
    }
    else
    {
        temp = p;
        /* traverses the existing list to get the pointer to the last node
           of it */
        while (temp->link != p)
            temp = temp->link;
        temp-> link = (struct node *)malloc(sizeof(struct node)); /**
           creates new node using
           data value passes as
           parameter and puts its
           address in the link field
           of last node of the
           existing list*/
        if(temp -> link == NULL)
        {
            printf("Error\n");
            exit(0);
        }
        temp = temp->link;
```

```

temp-> data = n;
temp-> link = p;
}
return (p);
}
void printlist( struct node *p )
{
struct node *temp;
temp = p;
printf("The data values in the list are\n");
if(p!=NULL)
{
do
{
printf("%d\t",temp->data);
temp=temp->link;
} while (temp!= p);
}
else
printf("The list is empty\n");
}
void main()
{
int n;
int x;
struct node *start = NULL ;
printf("Enter the nodes to be created \n");
scanf("%d",&n);
while ( n -- > 0 )
{
printf( "Enter the data values to be placed in a node\n");
scanf("%d",&x);
start = insert ( start, x );
}
printf("The created list is\n");
printlist( start );
}

```

Deleting the Specified Node in Singly Linked List

To delete a node, first we determine the node number to be deleted (this is based on the assumption that the nodes of the list are numbered serially from 1 to n). The list is then traversed to get a pointer to the node whose number is given, as well as a pointer to a node that appears before the

Unit 02: Arrays vs Linked Lists

node to be deleted. Then the link field of the node that appears before the node to be deleted is made to point to the node that appears after the node to be deleted, and the node to be deleted is freed.



Lab Exercise:

```
# include <stdio.h>
# include <stdlib.h>
struct node *delet( struct node *, int );
int length ( struct node * );
struct node
{
    int data;
    struct node *link;
};
struct node *insert(struct node *p, int n)
{
    struct node *temp;
    if(p==NULL)
    {
        p=(struct node *)malloc(sizeof(struct node));
        if(p==NULL)
        {
            printf("Error\n");
            exit(0);
        }
        p-> data = n;
        p-> link = NULL;
    }
    else
    {
        temp = p;
        while (temp->link != NULL)
            temp = temp->link;
        temp-> link = (struct node *)malloc(sizeof(struct node));
        if(temp -> link == NULL)
        {
            printf("Error\n");
            exit(0);
        }
        temp = temp->link;
        temp-> data = n;
        temp-> link = NULL;
    }
    return (p);
}
```

```

}

void printlist( struct node *p )
{
printf("The data values in the list are\n");
while (p!= NULL)
{
printf("%d\t",p-> data);
p = p->link;
}
}

void main()
{
int n;
int x;
struct node *start = NULL;
printf("Enter the nodes to be created \n");
scanf("%d",&n);
while ( n- > 0 )
{
printf( "Enter the data values to be placed in a node\n");
scanf("%d",&x);
start = insert ( start, x );
}
printf(" The list before deletion id\n");
printlist( start );
printf("% \n Enter the node no \n");
scanf( "%d",&n);
start = delet (start , n );
printf(" The list after deletion is\n");
printlist( start );
}

/* a function to delete the specified node*/
struct node *delet( struct node *p, int node_no )
{
struct node *prev, *curr ;
int i;
if (p == NULL )
{
printf("There is no node to be deleted \n");
}

```

[Subject]

```

else
{
if ( node_no> length (p))
{
printf("Error\n");
}
else
{
prev = NULL;
curr = p;
i = 1 ;
while ( i<node_no )
{
prev = curr;
curr = curr->link;
i = i+1;
}
if ( prev == NULL )
{
p = curr ->link;
free ( curr );
}
else
{
prev -> link = curr ->link ;
free ( curr );
}
}
}
return(p);
}

/* a function to compute the length of a linked list */
int length ( struct node *p )
{
int count = 0 ;
while ( p != NULL )
{
count++;
p = p->link;
}
}

```

```

return ( count );
}

```

Inserting a Node after the Specified Node in a Singly Linked List

To insert a new node after the specified node, first we get the number of the node in an existing list after which the new node is to be inserted. This is based on the assumption that the nodes of the list are numbered serially from 1 to n. The list is then traversed to get a pointer to the node, whose number is given. If this pointer is x, then the link field of the new node is made to point to the node pointed to by x, and the link field of the node pointed to by x is made to point to the new node. Figures 2.3 and 2.4 show the list before and after the insertion of the node, respectively.

Insertion in Linked List can happen at following places:

At the beginning of the linked list.

At the end of the linked list.

At a given position in the linked list.

Algorithm: Insertion at beginning

Step 1: IF PTR = NULL

Write OVERFLOW

 Go to Step 7

 [END OF IF]

Step 2: SET NEW_NODE = PTR

Step 3: SET PTR = PTR → NEXT

Step 4: SET NEW_NODE → DATA = VAL

Step 5: SET NEW_NODE → NEXT = HEAD

Step 6: SET HEAD = NEW_NODE

Step 7: EXIT

Deletion from a Linked List

Delete from beginning

Delete from end

Delete from middle/ given position

Find the previous node of the node to be deleted.

Change the next pointer of the previous node

Free the memory of the deleted node.

In case of first node deletion, we need to update the head of the linked list.

Algorithm: Deletion at beginning

Step 1: IF HEAD = NULL

Write UNDERFLOW

 Go to Step 5

 [END OF IF]

Step 2: SET PTR = HEAD

Step 3: SET HEAD = HEAD -> NEXT

Step 4: FREE PTR

Step 5: EXIT

Searching in linked list

Searching is performed to find the location of a particular element in the list. Traversing is performed in the list and make the comparison of every element of the list with the specified element. If the element is matched with any of the list element then the location of the element is returned from the function.

Algorithm: Searching in linked list

```

Step 1: SET PTR = HEAD
Step 2: Set I = 0
STEP 3: IF PTR = NULL
        WRITE "EMPTY LIST"
        GOTO STEP 8
    END OF IF
STEP 4: REPEAT STEP 5 TO 7 UNTIL PTR != NULL
STEP 5: if ptr → data = item
        write i+1
    End of IF
STEP 6: I = I + 1
STEP 7: PTR = PTR → NEXT
[END OF LOOP]
STEP 8: EXIT

```

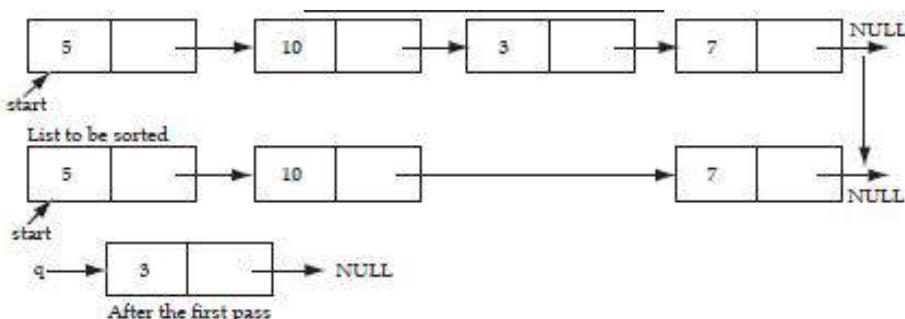
Linked List Common Errors

Here is summary of common errors of linked lists. Read these carefully, and read them againwhen you have problem that you need to solve.

1. Allocating a new node to step through the linked list; only a pointer variable is needed.
2. Confusing the and the \rightarrow operators.
3. Not setting the pointer from the last node to 0 (null).
4. Not considering special cases of inserting/removing at the beginning or the end of thelinked list.
5. Applying the delete operator to a node (calling the operator on a pointer to the node)before it is removed. Delete should be done after all pointer manipulations are completed.
6. Pointer manipulations that are out of order. These can ruin the structure of the linked list.

Sorting and Reversing a Linked List

To sort a linked list, first we traverse the list searching for the node with a minimum data value. Then we remove that node and append it to another list which is initially empty. We repeat thisprocess with the remaining list until the list becomes empty, and at the end, we return a pointerto the beginning of the list to which all the nodes are moved.



Sorting of linked list

To reverse a list, we maintain a pointer each to the previous and the next node, then we make the link field of the current node point to the previous, make the previous equal to the current, and the current equal to the next.

Arrays vs. Linked list

Array	Linked list
Data elements are stored in contiguous locations in memory.	New elements can be stored anywhere and a reference is created for the new element using pointers.
Insertion and Deletion operations are costlier since the memory locations are consecutive and fixed.	Insertion and Deletion operations are fast and easy in a linked list.
Memory is allocated during the compile time (<i>Static memory allocation</i>).	Memory is allocated during the run-time (<i>Dynamic memory allocation</i>).
Size of the array must be specified at the time of array declaration/initialization.	Size of a Linked list grows/shrinks as and when new elements are inserted/deleted.

Summary

- An array is a set of same data elements grouped together. Arrays can be one-dimensional or multidimensional.
- A linear or one-dimensional array is a structured collection of elements (often called as array elements) that are accessed individually by specifying the position of each element with a single index value.
- Multidimensional arrays are nothing but "arrays of arrays". Two subscripts are used to refer to the elements.
- The operations that are performed on an array are adding, sorting, searching, and traversing.
- Traversing an array refers to moving in inward and outward direction to access each element in an array.
- Linked list is a technique of dynamically implementing a list using pointers. A linked list contains two fields namely, data field and link field.
- A singly-linked list consists of only one pointer to point to another node and the last node always points to NULL to indicate the end of the list.
- A doubly-linked list consists of two pointers, one to point to the next node and the other to point to the previous node.
- In a circular singly-linked list, the last node always points to the first node to indicate the circular nature of the list.
- A circular doubly-linked list consists of two pointers for forward and backward traversal and the last node points to the first node.

- Searching operation involves searching for a specific element in the list using an associated key.
- Insertion operation involves inserting a node at the beginning or end of a list.
- Deletion operation involves deleting a node at the beginning or following a given node or at the end of a list.

Keywords

Non-linear Data Structure: Every data item is attached to several other data items in a way that is specific for reflecting relationships. The data items are not arranged in a sequential structure.

Searching: Finding the location of the record with a given key value, or finding the locations of all records, which satisfy one or more conditions.

Traversing: Accessing each record exactly once so that certain items in the record may be processed.

Circular Linked List: A linear linked list in which the last element points to the first element, thus forming a circle.

Doubly Linked List: A linear linked list in which each element is connected to the two nearest elements through pointers.

Self-Assessment

1. Which one is correct statement?
 - A. Search in array is delete an element from array
 - B. Search in array is find an element from array
 - C. Search in array is insert an element in array
 - D. None of above

2. Which is not correct syntax?
 - A. abcname[];
 - B. abcname[10];
 - C. abcname[3] = {20,25,35};
 - D. abcname[5] = {15;20;28};

3. Searching is a process in which we find element in array
 - A. True
 - B. False

4. Operations can be performed on array
 - A. Sorting
 - B. Merging
 - C. Traversing
 - D. All of above

5. To merge two arrays, how many variables are required (minimum)?

- A. 1
 - B. 2
 - C. 5
 - D. 3
6. Elements of arrays are stored in memory locations
- A. Random
 - B. Sequential
 - C. Both random and sequential
 - D. None of above
7. Which is correct statement?
- A. Insertion at the given index of an array
 - B. Insertion after the given index of an array
 - C. Insertion before the given index of an array
 - D. All of above
8. Traversal is process of visit each element of an array
- A. True
 - B. False
9. Which statement is incorrect?
- 1. int arr[MAX]={10,12,15,20},i,val;
 - 2. printf("the array element are \n");
 - 3. for(i=0;i<4;i++)
 - 4. none of above
- A. 1
 - B. 2
 - C. 3
 - D. 4
10. Array is_____
- A. A group of elements of same data type
 - B. Array elements are stored in memory in continuous or contiguous locations
 - C. An array contains more than one element
 - D. All of above
11. What are the advantages of arrays?
- A. Objects of mixed data types can be stored
 - B. Easier to store elements of same data type
 - C. Elements in an array cannot be sorted

D. Index of first element of an array is 1

12. The index of the first element in an array is _____

- A. 1
- B. 2
- C. -1
- D. 0

13. Linked list consist of...

- A. Data field
- B. Link field
- C. Both data and link field
- D. None of above

14. What are the shortcomings of array?

- A. Memory allocation
- B. Memory efficiency
- C. Execution time
- D. All of above

15. What are the types of linked list?

- A. Single
- B. Double
- C. Circular
- D. All of above

16. Operations performed on Linked list are...

- A. Insertion
- B. Deletion
- C. Search
- D. All of above

17. Insertion in Linked List can happen at following places

- A. At the beginning of the linked list.
- B. At the end of the linked list.
- C. At a given position in the linked list.
- D. All of above

18. Linked list is considered as an example of _____ type of memory allocation

- A. Static
- B. Heap
- C. Dynamic

D. Compile time

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. B | 2. D | 3. A | 4. D | 5. D |
| 6. B | 7. D | 8. A | 9. B | 10. D |
| 11. B | 12. D | 13. C | 14. D | 15. D |
| 16. D | 17. D | 18. C | | |

Review Questions

1. Define array and its types.
2. Give an example of multidimensional array.
3. Discuss any two types of array initialization methods with example.
4. Discuss different sorting methods.
5. Write a program to sort the elements of a linked list.
6. Differentiate between array and linked list with suitable example.
7. Discuss different operation performed with linked list.
8. Discuss advantages of linked list as compared to arrays.



Further Readings

Data Structures and Efficient Algorithms, Burkhard Monien, Thomas Ottmann, Springer.

Kruse Data Structure & Program Design, Prentice Hall of India, New Delhi

Mark Allen Weles: Data Structure & Algorithm Analysis in C Second Adition. Addison-Wesley publishing

Sorenson and Tremblay: An Introduction to Data Structure with Algorithms.

Thomas H. Cormen, Charles E. Leiserson & Ronald L. Rivest: Introduction to Algorithms. Prentice-Hall of India Pvt. Limited, New Delhi

Timothy A. Budd, Classic Data Structures in C++, Addison Wesley.



Web Links

www.en.wikipedia.org

www.web-source.net

www.webopedia.com

<https://www.geeksforgeeks.org/data-structures/>

<https://www.programiz.com/dsa/data-structure-types>

[Subject]

Unit 03: Stacks

CONTENTS

- Objectives
- Introduction
- 3.1 Stack Structure
- 3.2 Basic Operations of Stack
- 3.3 Implementation of Stacks
- 3.4 Applications of Stacks
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to:

- Learn fundamentals of stacks
- Explain the basic operations of stack
- Explain the implementation and applications of stacks

Introduction

Stacks are simple data structures and an important tool in programming language. Stacks are linear lists which have restrictions on the insertion and deletion operations. These are special cases of ordered list in which insertion and deletion is done only at the ends.

The basic operations performed on stack are push and pop. Stack implementation can be done in two ways - static implementation or dynamic implementation. Stack can be represented in the memory using a one-dimensional array or a singly linked list.

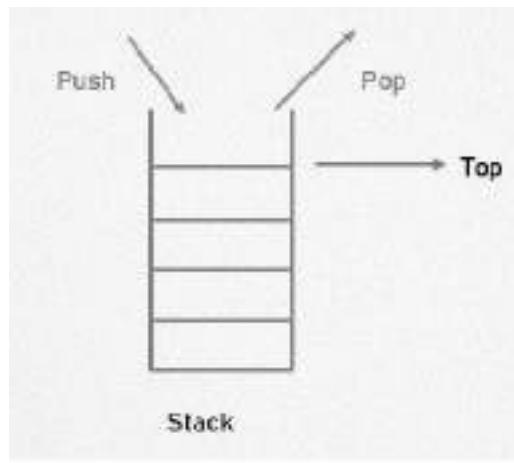
Stack is another linear data structure having a very interesting property. Unlike arrays and link lists, an element can be inserted and deleted not at any arbitrary position but only at one end. Thus, one end of a stack is sealed for insertion and deletion while the other end allows both the operations.

3.1 Stack Structure

The stack data structure is used to maintain records of a file in which the order among the records of file is not important. Figure 7.1 displays the structure of a stack where stack is like a hollow cylinder with a closed bottom end and an open top end. In the stack data structure, the records are added and deleted at the top end. Last-In-First-Out (LIFO) principle is followed to retrieve records from the stack. The records added last are accessed first.

A stack is a linear data structure in as much as its member elements are ordered as 1st, 2nd,... and last. However, an element can be inserted in and deleted from only one end. The other end remains sealed. This open end to which elements can be inserted and deleted from is called stack top or top of the stack. Consequently, the elements are removed from a stack in the reverse order of insertion.

A stack is said to possess LIFO (Last In First Out) property. A data structure has LIFO property if the element that can be retrieved first is the one that was inserted last.



3.2 Basic Operations of Stack

The basic operations of stack are to:

1. Insert an element in the stack (Push operation)
2. Delete an element from the stack (Pop operation)

Push Operation

The procedure to insert a new element to the stack is called push operation. The push operation adds an element on the top of the stack. 'Top' refers to the element on the top of stack. Push makes the 'Top' point to the recently added element on the stack. After every push operation, the 'Top' is incremented by one. When the array is full, the status of stack is FULL and the condition is called stack overflow. No element can be inserted when the stack is full

Algorithm to Implement Push Operation on Stack

```
PUSH(STACK, n, top, item) /* n = size of stack*/
if (top = n) then STACK_FULL; /* checks for stack overflow */
else
{ top = top+1; /* increases the top by 1 */
STACK [top] = item; /* inserts item in the new top position */
end PUSH
```

Pop Operation

The procedure to delete an element from the top of the stack is called pop operation. After every pop operation, the 'Top' is decremented by one. When there is no element in the stack, the status of the stack is called empty stack or stack underflow. The pop operation cannot be performed when it is in stack underflow condition.

Algorithm to Implement Pop Operation in a Stack

```
POP(STACK, top, item)
if (top = 0) then STACK_EMPTY; /* check for stack underflow */
else { item = STACK [top]; /* remove top element */
top = top - 1; /* decrement stack top */
}
end POP
```

3.3 Implementation of Stacks

There are two basic methods for the implementation of stacks – one where the memory is used statically and the other where the memory is used dynamically.

Array-based Implementation

A stack is a sequence of data elements. To implement a stack structure, an array can be used as it is a storage structure. Each element of the stack occupies one array element. Static implementation of stack can be achieved using arrays. The size of the array, once declared, cannot be changed during the program execution. Memory is allocated according to the array size. The memory requirement is determined before the compilation. The compiler provides the required memory. This is suitable when the exact number of elements is known. The static allocation is an inefficient memory allocation technique because if fewer elements are stored than declared, the memory is wasted and if more elements need to be stored than declared, the array cannot expand. In both the cases, there is inefficient use of memory.

The following pseudo-code shows the array-based implementation of a stack. In this, the elements of the stack are of type T.

```
struct stk
{ T array[max_size];
/* max_size is the maximum size */
int top = -1;
/* stack top initially given value -1 */
} stack;

void push(T e)
/*inserts an element e into the stack s*/
{
if (stack.top == max_size)
printf("Stack is full-insertion not possible");
else
{
stack.top = stack.top + 1;
stack.array[stack.top] = e;
}
}

T pop()
/*Returns the top element from the stack */
{
T x;
if(stack.top == -1)
printf("Stack is empty");
else
{
x = stack.array[stack.top];
stack.top = stack.top - 1;
return(x);
}
}
```

```

}

booleanempty()
/* checks if the stack is empty */
{
    boolean empty = false;
    if(stack.top == -1)
        empty = true else empty = false;
    return(empty);
}

void initialise()
/* This procedure initializes the stack s */
{
    stack.top = -1;
}

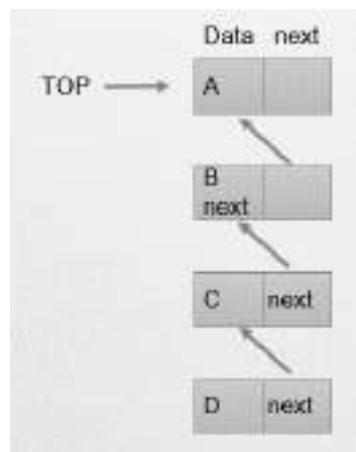
```

The above implementation strategy is easy and fast since it does not have run-time overheads. At the same time it is not flexible since it cannot handle a situation when the number of elements exceeds max_size. Also, let us say, if max_size is derided statically to 100 and a stack actually has only 10 elements, then memory space for the rest of the 90 elements would be wasted.

Linked List Representation of Stacks

The array representation of stacks is easy and convenient. However, it allows the representation of only fixed sized stacks. The size of the stack varies during program application for different applications. Representing stack using linked list can solve this problem. A singly linked list can be used to represent any stack. In a singly linked list, the data field represents the ITEM and the LINK field points to the next item.

In linked list implementation of the stack, we need to create nodes and the nodes are maintained non-contiguously in the memory. Each node contains a pointer to its immediate successor node in the stack. Stack is said to be overflowed if the space left in the memory heap is not enough to create a node.



Here the memory is used dynamically. For every push operation, the memory space for one element is allocated at run-time and the element is inserted into the stack. For every pop operation, the memory space for the deleted element is de-allocated and returned to the free space pool. Hence the shortcomings of the array-based implementation are overcome. But since, this allocates memory dynamically, the execution is slowed down.

The following pseudo-code is for the pointer-based implementation of a stack. Each element of the stack is of type T.

```
struct stk
{
T element;
struct stk *next;
};

struct stk *stack;

void push(struct stk *p, T e)
{
struct stk *x;
x = new(stk);
x.element = e;
x.next = NULL;
p = x;
}
```

Here the stack full condition is checked by the call to new which would give an error if no memory space could be allocated.

```
T pop(struct stk *p)
{
struct stk *x;
if (p == NULL)
printf("Stack is empty");
else
{x = p;
x = x.next;
return(p.element);
}
booleanempty(sstructstk *p)
{
if (p == NULL)
return(true);
else
return(false);
}
void initialize(struct stk *p)
{
p = NULL;
}
```

3.4 Applications of Stacks

There are numerous applications of the stack data structure in computer algorithms. It is used to store return information in the case of function/procedure/subroutine calls. Hence, one would find a stack in architecture of any Central Processing Unit (CPU). In this section, we would just illustrate a few of them.

Expression Evaluation and Conversion

Parenthesis Checking

Backtracking

Function Call

String Reversal

Memory Management

Syntax Parsing

Parenthesis checker

Parenthesis checker is used for balanced Brackets in an expression. The balanced parenthesis means that when the opening parenthesis is equal to the closing parenthesis, then it is a balanced parenthesis.

(a+b*(c/d))

[10+20*(6+7)]

(x+y)/(c-d)

Balanced parenthesis

A = (50+25)

In the above expression there is one opening and one closing parenthesis means that both opening and closing brackets are equal; therefore, the above expression is a balanced parenthesis.

Unbalanced parenthesis

A= [(15+25)

The above expression has two opening brackets and one closing bracket, which means that both opening and closing brackets are not equal; therefore, the above expression is unbalanced.

Algorithm

- Initialize a character stack.
- Now traverse the expression string exp.
- If the current character is a starting bracket ('(' or '{' or '[') then push it to stack.
- If the current character is a closing bracket (')' or '}' or ']') then pop from stack and if the popped character is the matching starting bracket then balanced else brackets are not balanced.
- After complete traversal, if there is some starting bracket left in stack then not balanced

Expression conversion and evaluation

Arithmetic expressions can be represented in 3 forms:

Infix notation

Postfix notation (Reverse Polish Notation)

Prefix notation (Polish Notation)

Infix Notation

Infix Notation can be represented as:

operand1 operator operand1



Example: 15 + 26

a + b

Postfix Notation

Postfix Notation can be represented as

operand1 operand2 operator



Example: 15 29 +

a b +

Prefix notation

Prefix notation can be represented as

operator operand1 operand2



Example: + 10 20

+ a b

Infix notation is used most frequently in our day to day tasks. Machines find infix notations tougher to process than prefix/postfix notations. Hence, compilers convert infix notations to prefix/postfix before the expression is evaluated.

The precedence of operators needs to be taken care as per hierarchy

(^) > (*) > (/) > (+) > (-)

Brackets have the highest priority.

To evaluate an infix expression, We need to perform 2 steps:

- Convert infix to postfix
- Evaluate postfix

Sorting

A Sorting process is used to rearrange a given array or elements based upon selected algorithm/ sort function.



Quick Sort is used for sorting a list of data elements. Quicksort is a sorting algorithm based on the divide and conquer approach. An array is divided into subarrays by selecting a pivot element. During array dividing, the pivot element should be positioned in such a way that elements less than pivot are kept on the left side and elements greater than pivot are on the right side of the pivot. The left and right subarrays are also divided using the same approach. This process continues until each subarray contains a single element

There are many different versions of quick Sort that pick pivot in different ways.

Always pick first element as pivot.

Always pick last element as pivot

Pick a random element as pivot.

Pick median as pivot.

Algorithm

```
quickSort(arr, beg, end)
if (beg < end)
    pivotIndex = partition(arr,beg, end)
    quickSort(arr, beg, pivotIndex)
    quickSort(arr, pivotIndex + 1, end)
partition(arr, beg, end)
set end as pivotIndex
pIndex = beg - 1
for i = beg to end-1
    if arr[i] < pivot
        swap arr[i] and arr[pIndex]
    pIndex++
    swap pivot and arr[pIndex+1]
return pIndex + 1
```

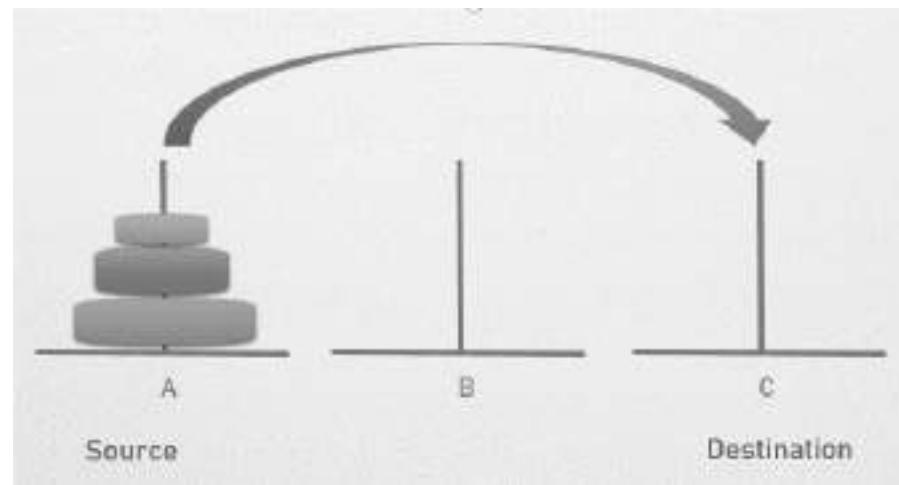
Tower of Hanoi

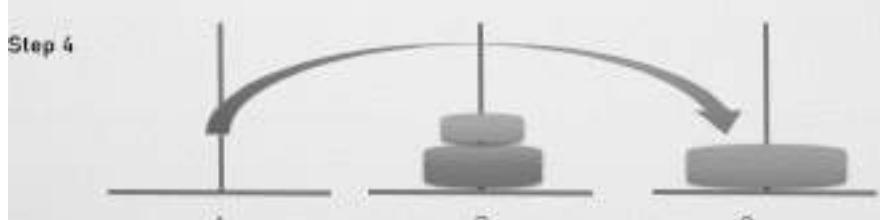
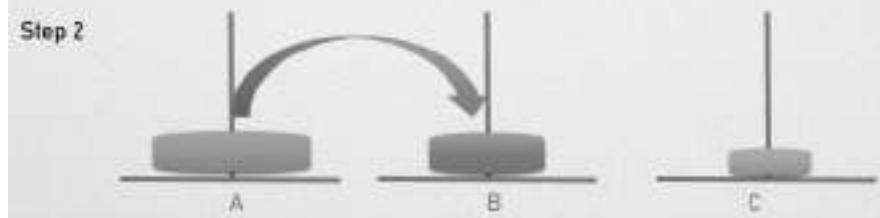
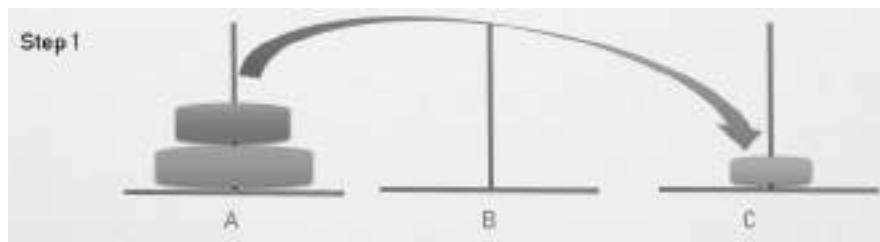
The Tower of Hanoi, is a mathematical problem which consists of three rods and multiple disks. Initially, all the disks are placed on one rod, one over the other in ascending order of size similar to a cone-shaped tower.

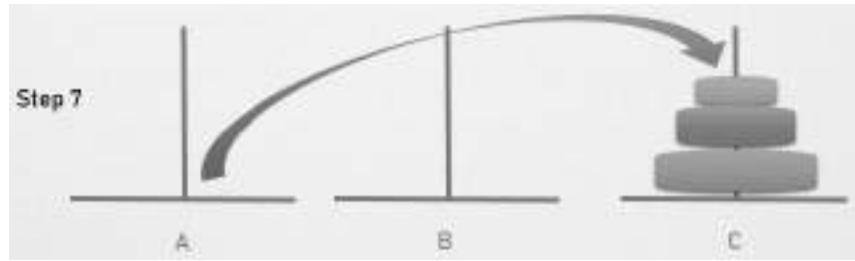


The objective of this problem is to move the stack of disks from the source to destination, following these rules:

1. Only one disk can be moved at a time.
2. Only the top disk can be removed.
3. No large disk can sit over a small disk.







Iterative Algorithm

1. At First Calculate the number of moves required i.e. "pow(2,n) - 1" where "n" is number of discs.
2. If the number of discs i.e n is even then swap Destination Rod and Auxiliary Rod.
3. for i = 1 upto number of moves:
 - Check if "i mod 3" == 1:
Perform Movement of top disc between Source Rod and Destination Rod.
 - Check if "i mod 3" == 2:
Perform Movement of top disc between Source Rod and Auxiliary Rod.
 - Check if "i mod 3" == 0:
Perform Movement of top disc between Auxiliary Rod and Destination Rod.

Simulating Recursive Function using Stack

A recursive solution to a problem is often more expensive than a non-recursive solution, both in terms of time and space. Frequently, this expense is a small price to pay for the logical simplicity and self-documentation of the recursive solution. However, in a production program (such as a compiler, for example) that may be run thousands of times, the recurrent expense is a heavy burden on the system's limited resources.

Thus, a program may be designed to incorporate a recursive solution in order to reduce the expense of design and certification, and then carefully converted to a non-recursive version to be put into actual day-to-day use. As we shall see, in performing such as conversion it is often possible to identify parts of the implementation of recursion that are superfluous in a particular application and thereby significantly reduce the amount of work that the program must perform.

Suppose that we have the statement

`rout (x);` where rout is defined as a function by the header

`rout(a);` x is referred to as an argument (of the calling function), and a is referred to as a parameter (of the called function).

What happens when a function is called? The action of calling a function may be divided into three parts:

1. Passing Arguments
2. Allocating and initializing local variables
3. Transferring control to the function.

1. **Passing arguments:** For a parameter in C, a copy of the argument is made locally within the function, and any changes to the parameter are made to that local copy. The effect to this scheme is that the original input argument cannot be altered. In this method, storage for the argument is allocated within the data area of the function.

2. **Allocating and initializing local variables:** After arguments have been passed, the local variables of the function are allocated. These local variables include all those declared directly in the function and any temporaries that must be created during the course of execution.

3. **Transferring control to the function:** At this point control may still not be passed to the function because provision has not yet been made for saving the return address. If a function is

given control, it must eventually restore control to the calling routine by means of a branch. However, it cannot execute that branch unless it knows the location to which it must return. Since this location is within the calling routine and not within the function, the only way that the function can know this address is to have it passed as an argument. This is exactly what happens. Aside from the explicit arguments specified by the programmer, there is also a set of implicit arguments that contain information necessary for the function to execute and return correctly. Chief among these implicit arguments is the return address. The function stores this address within its own data area. When it is ready to return control to the calling program, the function retrieves the return address and branches to that location. Once the arguments and the return address have been passed, control may be transferred to the function, since everything required has been done to ensure that the function can operate on the appropriate data and then return to the calling routine safely.

Summary

- A stack is a linear data structure in which allocation and deallocation are made in a last-in-first-out (LIFO) method.
- The basic operations of stack are inserting an element on the stack (push operation) and deleting an element from the stack (pop operation).
- Stacks are represented in main memory by using one-dimensional array or a singly linked list.
- To implement a stack structure, an array can be used as its storage structure. Each element of the stack occupies one array element. Static implementation of stack can be achieved using arrays.
- Stack is used to store return information in the case of function/procedure/subroutine calls. Hence, one would find a stack in architecture of any Central Processing Unit (CPU).
- In infix notation operators come in between the operands. An expression can be evaluated using stack data structure.

Keywords

LIFO: (Last In First Out) the property of a list such as stack in which the element which can be retrieved is the last element to enter it.

Pop: Stack operation retrieves a value from the stack.

Infix: Notation of an arithmetic expression in which operators come in between their operands.

Postfix: Notation of an arithmetic expression in which operators come after their operands.

Prefix: Notation of an arithmetic expression in which operators come before their operands.

Push: Stack operation which puts a value on the stack.

Stack: A linear data structure where insertion and deletion of elements can take place only at one end.

SelfAssessment

1. Which method is followed by Stack?

- A. FILO
- B. LIFO
- C. Both FILO and LIFO
- D. None of above

2. Which is not stack operation?
 - A. count()
 - B. peek()
 - C. getche()
 - D. display()

3. Which is not part of stack?
 - A. Overflow
 - B. Enqueue
 - C. Underflow
 - D. None of above

4. To returns the element at the given position which function is used?
 - A. isEmpty()
 - B. isFull()
 - C. peek()
 - D. display()

5. Stack implementation is done using.....
 - A. Array
 - B. Linked list
 - C. Both using array and linked list
 - D. None of above

6. Parenthesis checker is used for
 - A. Balanced Brackets
 - B. Unbalanced Brackets
 - C. Both balanced and unbalanced
 - D. None of above

7. Which is incorrect statement?
 - A. $(a+b*(c/d))$
 - B. $[10+20*(6+7)]$
 - C. $(x+y)/(c-d)$
 - D. None of above

8. Which is not Sorting type
 - A. Bubble
 - B. Merge
 - C. Insertion
 - D. Linear

9. Tower of Hanoi is associated with....

- A. Queue
- B. Stack
- C. Array
- D. None of above

10. Arithmetic expressions can be represented as

- A. Infix notation
- B. Postfix notation
- C. Prefix notation
- D. All of above

11. Prefix notation can be represented as

- A. operator operand1 operand2
- B. operand1 operand2 operator
- C. operand1 operator operand1
- D. None of above

12. Postfix notation can be represented as

- A. operator operand1 operand2
- B. operand1 operator operand1
- C. operand1 operand2 operator
- D. None of above

13. Stack implementation is done using_____

- A. Statically
- B. Dynamically
- C. Both Statically and Dynamically
- D. None of above

14. Which is not stack operation?

- A. peek()
- B. pop()
- C. display()
- D. printf()

15. Underflow condition occur when stack is_____

- A. Full
- B. Empty
- C. Half
- D. None of above

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. C | 2. C | 3. B | 4. C | 5. C |
| 6. C | 7. B | 8. D | 9. B | 10. D |
| 11. A | 12. C | 13. C | 14. D | 15. B |

Review Questions

- 1 Explain how function calls may be implemented using stacks for return values.
- 2 What are the advantages of implementing a stack using dynamic memory allocation method?
- 3 Explain concept of tower of Hanoi.
- 4 what are the different methods for implementing stacks?
- 5 Give an example of push and pop operation using stack.
- 6 Write an algorithm to reverse an input string of characters using a stack.

**Further Readings**

Data Structures and Efficient Algorithms, Burkhard Monien, Thomas Ottmann, Springer.

Kruse Data Structure & Program Design, Prentice Hall of India, New Delhi

Mark Allen Weles: Data Structure & Algorithm Analysis in C Second Adition. Addison-Wesley publishing

RG Dromey, How to Solve it by Computer, Cambridge University Press.

Shi-kuo Chang, Data Structures and Algorithms, World Scientific

Sorenson and Tremblay: An Introduction to Data Structure with Algorithms.

Thomas H. Cormen, Charles E, Leiserson& Ronald L. Rivest: Introduction to Algorithms. Prentice-Hall of India Pvt. Limited, New Delhi

Timothy A. Budd, Classic Data Structures in C++, Addison Wesley.

**Web Links**

www.en.wikipedia.org

www.webopedia.com

<https://www.programiz.com/>

<https://www.javatpoint.com/data-structure-stack>

https://www.tutorialspoint.com/data_structures_algorithms/stack_algorithm.htm

Unit 04: Queues

CONTENTS

- Objectives
- Introduction
- 4.1 Fundamentals of Queues
- 4.2 Basic Operations of Queue
- 4.3 Types of Queue
- 4.4 Implementation of Queues
- 4.5 Applications of Queues
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to:

- Learn implementation of queues
- Explain priority queue
- Discuss applications of queues

Introduction

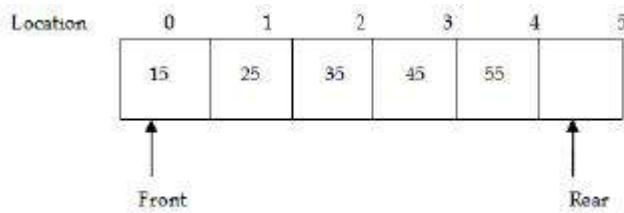
A queue is a linear list of elements that consists of two ends known as front and rear. We can delete elements from the front end and insert elements at the rear end of a queue. A queue in an application is used to maintain a list of items that are ordered not by their values but by their sequential value.

The queue abstract data type is also a widely used one with applications very common in real life. An example comes from the operating system software where the scheduler picks up the next process to be executed on the system from a queue data structure. In this unit, we would study the various properties of queues, their operations and implementation strategies.

4.1 Fundamentals of Queues

A queue is an ordered collection of items in which deletion takes place at one end, which is called the front of the queue, and insertion at the other end, which is called the rear of the queue. The queue is a 'First In First Out' system (FIFO). In a time-sharing system, there can be many tasks waiting in the queue, for access to disk storage or for using the CPU. The queues in a bank, or railway station counter are examples of queue. The first person in the queue is the first to be attended.

The two main operations in the queue are insertion and deletion of items. The queue has two pointers, the front pointer points to the first element of the queue and the rear pointer points to the last element of the queue.



4.2 Basic Operations of Queue

The basic operations of queue are insertion and deletion of items which are referred as enqueue and dequeue respectively. In enqueue operation, an item is added to the rear end of the queue. In dequeue operation, the item is deleted from the front end of the queue.

Insert at Rear End

To insert an item into the queue, first it should be verified whether the queue is full or not. If the queue is full, a new item cannot be inserted into the queue. The condition `FRONT=NULL` indicates that the queue is empty. If the queue is not full, items are inserted at the rear end. When an item is added to the queue, the value of `rear` is incremented by 1.

In queue we need to maintain two data pointers, front and rear. Operations on queue are comparatively difficult to implement than that of stacks

Step 1 – Check if the queue is full.

Step 2 – If the queue is full, produce overflow error and exit.

Step 3 – If the queue is not full, increment rear pointer to point the next empty space.

Step 4 – Add data element to the queue location, where the rear is pointing.

Step 5 – return success.

Algorithm: Enqueue operation

```

procedure enqueue(data)
    if queue is full
        return overflow
    endif
    rear ← rear + 1
    queue[rear] ← data
    return true
end procedure

```

Delete from the Front End

To delete an item from the stack, first it should be verified that the queue is not empty. If the queue is not empty, the items are deleted at the front end of the queue. When an item is deleted from the queue, Dequeue operation include two tasks: access the data where front is pointing and remove the data after access.

Step 1 – Check if the queue is empty.

Step 2 – If the queue is empty, produce underflow error and exit.

Step 3 – If the queue is not empty, access the data where front is pointing.

Step 4 – Increment front pointer to point to the next available data element.

Step 5 – Return success. the value of the front is incremented by 1.

Algorithm: Dequeue operation

procedure dequeue

if queue is empty

```

    return underflow
end if
data = queue[front]
front ← front + 1
return true
end procedure

```



Example:

```

/*Program of queue using array*/
/*insertion and deletion in a queue*/
/*insertion and deletion in a queue*/
#include <stdio.h>
#define MAX 50
int queue_arr[MAX];
int rear = -1;
int front = -1;
void ins_delete();
void insert();
void display();
void main()
{
int choice;
while(1)
{
printf("1.Insert\n");
printf("2.Delete\n");
printf("3.Display\n");
printf("4.Quit\n");
printf("Enter your choice : \n");
scanf("%d",&choice);
switch(choice)
{
case 1 : insert();
break;
case 2 :ins_delete();
break;
case 3: ins_display();
break;
case 4: exit(1);
default:
}
}

```

```
printf("Wrong choice\n");
}/*End of switch*/
}/*End of while*/
}/*End of main()*/
void insert()
{
int added_item;
if (rear==MAX-1)
printf("Queue overflow\n");
else
{
if (front==-1) /*If queue is initially empty */
front=0;
printf("Enter an element to add in the queue : ");
scanf("%d", &added_item);
rear=rear+1;
queue_arr[rear] = added_item ;
}
} /*End of insert()*/
void ins_delete()
{
if (front == -1 || front > rear)
{
printf("Queue underflow\n");
return ;
}
else
{
printf("Element deleted from queue is : %d\n", queue_arr[front]);
front=front+1;
}
} /*End of delete() */
void display()
{
int i;
if (front == -1)
printf("Queue is empty\n");
else
{
printf("Elements in the queue:\n");
for(i=front;i<= rear;i++)
}
```

```

printf("%d ",queue_arr[i]);
printf("\n");
}
} /*End of display() */

```

Output:

1. Insert
2. Delete
3. Display
4. Quit

Enter your choice: 1

Enter an element to add in the queue: 25

Enter your choice: 1

Enter an element to add in the queue: 36

Enter your choice: 3

Elements in the queue: 25, 36

Enter your choice: 2

Element deleted from the queue is: 25

In this example:

1. The preprocessor directives #include are given. MAXSIZE is defined as 50 using the #define statement.
2. The queue is declared as an array using the declaration int queue_arr[MAX].
3. In the while loop, the different options are displayed on the screen and the value entered in the variable choice is accepted.
4. The switch case compares the value entered and calls the method corresponding to it. If the value entered is invalid, it displays the message "Wrong choice".
5. Insert method: The insert method inserts item in the queue. The if condition checks whether the queue is full or not. If the queue is full, the "Queue overflow" message is displayed. If the queue is not full, the item is inserted in the queue and the rear is incremented by 1.
6. Delete method: The delete method deletes item from the queue. The if condition checks whether the queue is empty or not. If the queue is empty, the "Queue underflow" message is displayed. If the queue is not empty, the item is deleted and front is incremented by 1.
7. Display method: The display method displays the contents of the queue. The if condition checks whether the queue is empty or not. If the queue is not empty, it displays all the items in the queue.

4.3 Types of Queue

Simple Queue

In a simple queue, insertion takes place at the rear and removal occurs at the front. It follows the FIFO (First in First out) rule.

Circular Queue

In a circular queue, the last element points to the first element making a circular link. In a circular queue, the rear end is connected to the front end forming a circular loop. An advantage of circular queue is that, the insertion and deletion operations are independent of one another. This prevents an interrupt handler from performing an insertion operation at the same time when the main function is performing a deletion operation.

Double ended queue

Double ended queue is also known as deque. It is a type of queue where the insertions and deletions happen at the front or the rear end of the queue. The various operations that can be performed on the double ended queue are:

1. Insert an element at the front end
2. Insert an element at the rear end
3. Delete an element at the front end
4. Delete an element at the rear end



Example:

Program for Implementation of Circular Queue.

```
#include<stdio.h>
#include<conio.h>
#define SIZE 5
int Q_F(int COUNT)
{
    return (COUNT==SIZE)? 1:0;
}
int Q_E(int COUNT)
{
    return (COUNT==0)? 1:0;
}
void rear_insert(int item, int Q[], int *R, int *COUNT)
{
    if(Q_F(*COUNT))
    {
        printf("Queue overflow");
        return;
    }
    *R=(*R+1) % SIZE;
    Q[*R]=item;
    *COUNT+=1;
}
void front_delete(int Q[], int *F, int *COUNT)
{
    if(Q_E(*COUNT))
    {
        printf("Queue underflow");
        return;
    }
    printf("The deleted element is %d\n", Q[*F]);
    *F=(*F+1) % SIZE;
}
```

```

*COUNT-=1;
}

void display(int Q[], int F, int COUNT)
{
int i,j;
if(Q_E(COUNT))
{
printf("Queue is empty\n");
return;
}
printf("The contents of the queue are:\n");
i=F;
for(j=1;j<=COUNT; j++)
{
printf("%d\n", Q[i]);
i=(i+1) % SIZE;
}
printf("\n");
}
void main()
{
int choice, num, COUNT, F, R, Q[20];
clrscr();
F=0;
R=-1;
COUNT=0;
for(;;)
{
printf("1. Insert at front\n");
printf("2. Delete at rear end\n");
printf("3. Display\n");
printf("4. Exit\n");
scanf("%d", &choice);
switch(choice)
{
case 1: printf("Enter the number to be inserted\n");
scanf("%d", &num);
rear_insert(num, Q, &R, &COUNT);
break;
case 2: front_delete(Q, &F, &COUNT);
break;
}
}
}

```

Advanced Data Structures

```

case 3: display(Q, F, COUNT);
break;
default: exit(0);
}
}
}

```

Output:

1. Insert at rear end
2. Delete at front end
3. Display
4. Exit

1

Enter the number to be inserted

50

1. Insert at rear end
2. Delete at front end
3. Display
4. Exit

1

Enter the number to be inserted

60

1. Insert at rear end
2. Delete at front end
3. Display
4. Exit

3

The contents of the queue are 50 60

1. Insert at rear end
2. Delete at front end
3. Display
4. Exit

2

The element deleted is 50



In this example:

1. The header files are included and a constant value 5 is defined for variable SIZE using #define statement. The SIZE defines the size of the queue.
2. A queue is created using an array named Q with an element capacity of 20. A variable named COUNT is declared to store the count of number elements present in the queue.
3. Four functions are created namely, Q_F(), Q_E(), rear_insert(), front_delete(), and display(). The user has to select an appropriate function to be performed.

Unit 04: Queues

4. The switch statement is used to call the rear_insert(), front_delete(), and display() functions.
5. When the user enters 1, rear_insert() function is called. In the rear_insert() function, the if loop checks if the count is full. If the condition is true, then the program prints a message "Queue is empty". Else, it checks for the value of R and assigns the element (num) entered by the user to R. Initially, when there are no elements in the queue, the value of R will be 0. After every insertion, the variable COUNT is incremented.
6. When the user enters 2, the front_delete() function is called. In this function, the if loop checks if the variable COUNT is empty. If the condition is true, then the program prints a message "Queue underflow". Else, the element in the 0th
7. When the user enters 3, the display() function is called. In this function, the if loop checks if the value of COUNT is 0. If the condition is true, the program prints a message "Queue is empty". Else, the value of F is assigned to the variable i. The for loop then displays the elements present in the queue. position will be deleted. The size of F is computed and the COUNT is set to 1.
8. When the user enters 4, the program terminates.

Priority Queue

In priority queue, the elements are inserted and deleted based on their priority. Each element is assigned a priority and the element with the highest priority is given importance and processed first. If all the elements present in the queue have the same priority, then the first element is given importance.

A priority queue is an abstract data type in which each element is associated with a priority value. Elements are served on the basis of their priority. An element with high priority is dequeued before an element with low priority. If two elements have the same priority, they are served according to their order in the queue.

The priority queue moves the highest priority elements at the beginning of the priority queue and the lowest priority elements at the back of the priority queue. It supports only those elements that are comparable. Priority queue in the data structure arranges the elements in either ascending or descending order.

Types of Priority Queue**Ascending Order Priority Queue**

An ascending order priority queue gives the highest priority to the lower number in that queue



Example:

List: 5 6 20 22 10

Arrange these numbers in ascending order.

List 5 6 10 20 22

Descending Order Priority Queue

A descending order priority queue gives the highest priority to the highest number in that queue.



Example:

List: 5 6 35 22 10

Arrange these numbers

List: 35 22 10 6 5

Priority Queue Operations

Inserting an Element into the Priority Queue

Deleting an Element from the Priority Queue

Peeking from the Priority Queue (Find max/min)

Extract-Max/Min from the Priority Queue

Priority queue can be implemented using
Array
Linked list
Heap data structure
Binary search tree

Priority Queue Applications

Dijkstra's algorithm: To find shortest path in graph.

Prim's Algorithm: Prim's algorithm uses the priority queue to the values or keys of nodes and draws out the minimum of these values at every step.

Data Compression: Huffman codes use the priority queue to compress the data.

Operating Systems: load balancing and interrupt handling in an operating system

4.4 Implementation of Queues

There are two possible implementation strategies – one where the memory is used statically and the other where memory is used dynamically.

Queue can be implemented using:

Array

Linked List

Queue implementation Using Array

To represent a queue we require a one-dimensional array of some maximum size say n to hold the data items and two other variables front and rear to point to the beginning and the end of the queue.

Queue implemented using array stores only fixed number of data values. Two variables front and rear, that are implemented in queue. Front and rear variables point to the position from where insertions and deletions are performed in a queue.

Initially both front and rear are set to -1. For insert a new value into the queue, increment rear value by one and then insert at that position. For delete a value from the queue, then delete the element which is at front position and increment front value by one.

Enqueue operation

Enqueue() function is used to insert a new element into the queue. In a queue, the new element is always inserted at rear position. The enqueue() function takes one integer value as a parameter and inserts that value into the queue.

Algorithm: Enqueue operation

Step 1: IF REAR = MAX - 1

 Write OVERFLOW

 Go to step

 [END OF IF]

Step 2: IF FRONT = -1 and REAR = -1

 SET FRONT = REAR = 0

 ELSE

 SET REAR = REAR + 1

 [END OF IF]

Step 3: Set QUEUE[REAR] = NUM

Step 4: EXIT

Dequeue operation

Dequeue() is a function used to delete an element from the queue. In a queue, the element is always deleted from front position. The Dequeue() function does not take any value as parameter.

Algorithm: Dequeue operation

Step 1: IF FRONT = -1 or FRONT > REAR

Write UNDERFLOW

ELSE

SET VAL = QUEUE[FRONT]

SET FRONT = FRONT + 1

[END OF IF]

Step 2: EXIT

Queue implementation Using Linked list

Due to the drawbacks of array. The array implementation cannot be used for the large scale applications where the queues are implemented. The alternative of array implementation is linked list implementation of queue. In a linked queue, each node of the queue consists of two parts i.e. data part and the link part. Each element of the queue points to its immediate next element in the memory.

In the linked queue, there are two pointers maintained in the memory i.e. front pointer and rear pointer. The front pointer contains the address of the starting element of the queue while the rear pointer contains the address of the last element of the queue.

Insert operation

There can be the two scenario of inserting this new node ptr into the linked queue. In the first scenario, we insert element into an empty queue. In this case, the condition front = NULL becomes true. In the second case, the queue contains more than one element. The condition front = NULL becomes false.

Algorithm

Step 1: Allocate the space for the new node PTR

Step 2: SET PTR -> DATA = VAL

Step 3: IF FRONT = NULL

SET FRONT = REAR = PTR

SET FRONT -> NEXT = REAR -> NEXT = NULL

ELSE

SET REAR -> NEXT = PTR

SET REAR = PTR

SET REAR -> NEXT = NULL

[END OF IF]

Step 4: END

Delete operation

Deletion operation removes the element that is first inserted among all the queue elements. The condition front == NULL becomes true if the list is empty. Otherwise, we will delete the element that is pointed by the pointer front.

Algorithm

Step 1: IF FRONT = NULL

Write " Underflow "

Go to Step 5

[END OF IF]

Step 2: SET PTR = FRONT

Step 3: SET FRONT = FRONT -> NEXT

Step 4: FREE PTR

Step 5: END

4.5 Applications of Queues

One major application of the queue data structure is in the computer simulation of a real-world situation. Queues are also used in many ways by the operating system, the program that schedules and allocates the resources of a computer system. One of these resources is the CPU (Central Processing Unit) itself. If you are working on a multi-user system and you tell the computer to run a particular program, the operating system adds your request to its "job queue". When your request gets to the front of the queue, the program you requested is executed. Similarly, the various users for the system must share the I/O devices (printers, disks etc.). Each device has its own queue of requests to print, read or write to these devices. The following subsection discusses one application of the queues – the priority queue. It is used in time-sharing multi-user systems where programs of high priority are processed first and programs with the same priority form a standard queue.

In Operating systems:

- a) Semaphores
- b) FCFS (first come first serve) scheduling,
- c) Spooling in printers
- d) Buffer for devices like keyboard

In Networks:

- a) Queues in routers/ switches
- b) Mail Queues

Queues are used in operating systems for handling interrupts.

Queues are used as buffers in most of the applications like MP3 media player, CD player, etc

When a resource is shared among multiple consumers.

- CPU scheduling,
- Disk Scheduling.

Summary

- A queue is an ordered collection of items in which deletion takes place at the front and insertion at the rear of the queue.
- In a memory, a queue can be represented in two ways; by representing the way in which the elements are stored in the memory, and by naming the address to which the front and rear pointers point to.
- The different types of queues are double ended queue, circular queue, and priority queue.
- The basic operations performed on a queue include inserting an element at the rear end and deleting an element at the front end.
- A priority queue is a collection of elements such that each element has been assigned a priority. An element of higher priority is processed before any element of lower priority.
- Two elements with the same priority are processed according to the order in which they were inserted into the queue.

Keywords

FIFO: (First In First Out) The property of a linear data structure which ensures that the element retrieved from it is the first element that went into it.

Front: The end of a queue from where elements are retrieved.

Queue: A linear data structure in which the element is inserted at one end while retrieved from another end.

Rear: The end of a queue where new elements are inserted.

Dequeue: Process of deleting elements from the queue.

Enqueue: Process of inserting elements into queue.

SelfAssessment

1. Which technique is followed by queue?

- A. FIFO
- B. LIFO
- C. Both LIFO and FIFO
- D. None of above

2. Enqueue () operation is used to perform

- A. Deletion
- B. Insertion
- C. Display
- D. All of above

3. Which operation is part of queue

- A. Peek
- B. isFull
- C. isEmpty
- D. All of above

4. Queue implementation is done using

- A. Array
- B. Stack
- C. Linked List
- D. All of above

5. Which is not types of Queue

- A. Circular
- B. Simple
- C. Complex
- D. Priority

6. Queue is a _____ data structure.
 - A. Static
 - B. Linear
 - C. Dynamic
 - D. None of above

7. Front pointer and rear pointer are used in...
 - A. Implementation of Queue using Linked List
 - B. Implementation of Queue using array
 - C. Implementation of Queue using stack
 - D. All of above

8. Underflow condition represent.
 - A. It checks if the queue is full before enqueueing any element.
 - B. It checks if there exists any item before popping from the queue.
 - C. It checks whether all variables are declared
 - D. All of above

9. Overflow condition represent.
 - A. It checks if there exists any item before popping from the queue.
 - B. It checks whether all variables are initialized.
 - C. It checks if the queue is full before enqueueing any element.
 - D. All of above

10. Front pointer contains
 - A. Address of the starting element of the queue
 - B. Address of the last element of the queue.
 - C. Link for next element
 - D. All of above

11. Priority queue is a_____ data structure.
 - A. Static
 - B. Linear
 - C. Abstract
 - D. All of above

12. Priority queue types are
 - A. Ascending order
 - B. Descending order
 - C. Both ascending and descending order
 - D. None of above

13. Priority Queue Operations are

- A. Deleting an Element from the Priority Queue
- B. Peeking from the Priority Queue (Find max/min)
- C. Extract-Max/Min from the Priority Queue
- D. All of above

14. Priority Queue Implementation are performed using.

- A. Linked list
- B. Heap data structure
- C. Binary search tree
- D. All of above

15. Binary Heap can be divided into

- A. Max heap
- B. Min-heap.
- C. Both max and min heap
- D. None of above

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. A | 2. B | 3. D | 4. D | 5. C |
| 6. B | 7. A | 8. B | 9. C | 10. A |
| 11. B | 12. C | 13. D | 14. D | 15. C |

Review Questions

- 1 "Using double ended queues is more advantageous than using circular queues. " Discuss
- 2 "Stacks are different from queues." Justify.
- 3 "Using priority queues is advantageous in job scheduling algorithms. " Analyze
- 4 Can a basic queue be implemented to function as a dynamic queue? Discuss
- 5 Describe the application of queue.
- 6 How will you insert and delete an element in queue?
- 7 Explain dynamic memory allocation advantages.



Further Readings

Data Structures and Algorithms; Shi-Kuo Chang; World Scientific.

Data Structures and Efficient Algorithms, Burkhard Monien, Thomas Ottmann, Springer.

Kruse Data Structure & Program Design, Prentice Hall of India, New Delhi

Mark Allen Weis: Data Structure & Algorithm Analysis in C Second Edition.

Addison-Wesley publishing

RG Dromey, How to Solve it by Computer, Cambridge University Press.

Shi-kuo Chang, Data Structures and Algorithms, World Scientific

Sorenson and Tremblay: An Introduction to Data Structure with Algorithms.

Thomas H. Cormen, Charles E. Leiserson & Ronald L. Rivest: Introduction to Algorithms. Prentice-Hall of India Pvt. Limited, New Delhi

Timothy A. Budd, Classic Data Structures in C++, Addison Wesley.



Web Links

www.en.wikipedia.org

www.web-source.net

www.webopedia.com

<https://www.geeksforgeeks.org/>

<https://www.javatpoint.com/data-structure-queue>

https://www.tutorialspoint.com/data_structures_algorithms/dsa_queue.htm

Unit 05: Search Trees

CONTENTS

- Objectives
- Introduction
- 5.1 Concept of Tree
- 5.2 Binary Tree
- 5.3 Binary Search Tree
- 5.4 Binary Search Tree Operations
- Summary
- Keywords
- Self Assessment
- Review Questions
- Answers for self Assessment
- Further Readings

Objectives

After studying this unit, you will be able to:

- Discuss the basics of tree
- Learn concept of binary tree
- Know binary tree traversal
- Explain the representation of tree in memory

Introduction

We know that data structure is a set of data elements grouped together under one name. A data structure can be considered as a set of rules that hold the data together. Almost all computer programs use data structures. Data structures are an essential part of algorithms. We can use it to manage huge amount of data in large databases. Some modern programming languages emphasize more on data structures than algorithms.

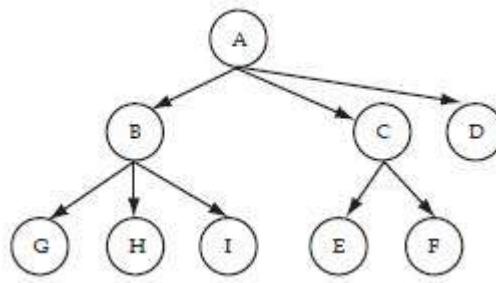
There are many data structures that help us to manipulate the data stored in the memory, which we have discussed in the previous units. These include array, stack, queue, and linked-list.

Choosing the best data structure for a program is a challenging task. Similar tasks may require different data structures. We derive new data structures for complex tasks using the already existing ones. We need to compare the characteristics of the data structures before choosing the right data structure. A tree is a hierarchical data structure suitable for representing hierarchical information. The tree data structure has the characteristics of quick search, quick inserts, and quick deletes.

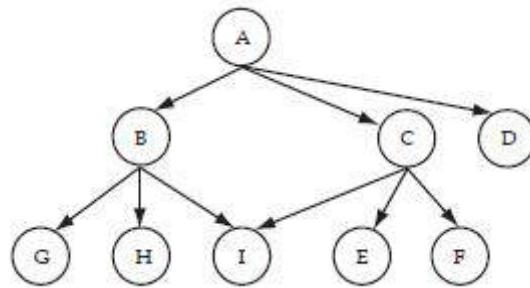
5.1 Concept of Tree

A tree is a set of one or more nodes T such that:

1. There is a specially designated node called root, and
2. Remaining nodes are partitioned into $n \geq 0$ disjoint set of nodes T_1, T_2, \dots, T_n each of which is a tree.



This is a tree because it is a set of nodes {A, B, C, D, E, F, G, H, I}, with node A as a root node, and the remaining nodes are partitioned into three disjoint sets: {B, G, H, I}, {C, E, F} AND {D} respectively. Each of these sets is a tree individually because each of these sets satisfies the above properties.



This is not a tree because it is a set of nodes {A, B, C, D, E, F, G, H, I}, with node A as a root node, but the remaining nodes cannot be partitioned into disjoint sets, because the node I is shared.

Given below are some of the important definitions, which are used in connection with trees.

Degree of Node of a Tree: The degree of a node of a tree is the number of sub-trees having this node as a root, or it is a number of decedents of a node. If degree is zero then it is called terminal node or leaf node of a tree.

Degree of a Tree: It is defined as the maximum of degree of the nodes of the tree, i.e. degree of tree = $\max(\text{degree}(n_i) \text{ for } i = 1 \text{ to } n)$.

Level of a Node: We define the level of the node by taking the level of the root node to be 1, and incrementing it by 1 as we move from the root towards the sub-trees i.e. the level of all the descendants of the root nodes will be 2. The level of their descendants will be 3 and so on. We then define depth of the tree to be the maximum value of level for node of a tree.

Root Node: The root of a tree is called a root node. A root node occurs only once in the whole tree.

Parent Node: The parent of a node is the immediate predecessor of that node.

Child Node: Child nodes are the immediate successors of a node.

Leaf Node: A node which does not have any child nodes is known as a leaf node.

Link: The pointer to a node in the tree is known as a link. A node can have more than one link.

Path: Every node in the tree is reachable from the root node through a unique sequence of links. This sequence of links is known as a path. The number of links in a path is considered to be the length of the path.

Levels: The level of a node in the tree is considered to be its hierarchical rank in the tree.

Height: The height of a non-empty tree is the maximum level of a node in the tree. The height of an empty tree (no node in a tree) is 0. The height of a tree containing a single node is 1. The longest path in the tree has to be considered to measure the height of the tree.

Height of a tree (h) = $I_{\max} + 1$, where I_{\max} is the maximum level of a tree.

Siblings: The nodes which have the same parent node are known as siblings.

Graphs consist of a set of nodes and edges, just like trees. But for graphs, there are no rules for the connections between nodes. In graphs, there is no concept of a root node, nor a concept of parents and children. Rather, a graph is just a collection of interconnected nodes. All trees are graphs. A tree is a special case of a graph, in which the nodes are all reachable from some starting node.

Representation of Tree in Graphs

A graph G consists of a set of objects $V = \{v_1, v_2, v_3 \dots\}$ called vertices (points or nodes) and a set of objects $E = \{e_1, e_2, e_3 \dots\}$ called edges (lines or arcs).

The set $V(G)$ is called the vertex set of G and $E(G)$ is the edge set.

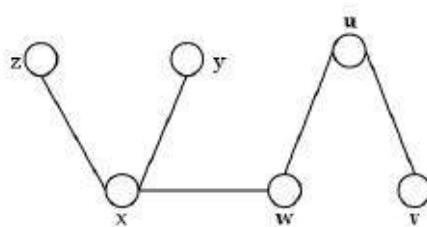
The graph is denoted as $G = (V, E)$

Let G be a graph and $\{u, v\}$ an edge of G . Since $\{u, v\}$ is 2-element set, we write $\{v, u\}$ instead of $\{u, v\}$.

This edge can be represented as uv or vu .

If $e = uv$ is an edge of a graph G , then u and v are adjacent in G and e joins u and v .

Consider given graph



This graph G is defined by the sets:

$$V(G) = \{u, v, w, x, y, z\} \text{ and } E(G) = \{uv, uw, wx, xy, xz\}$$

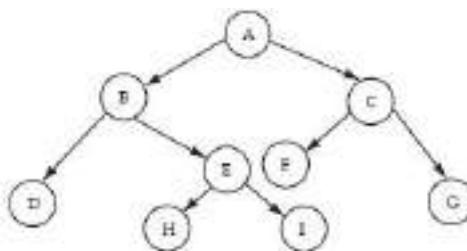
Every graph has a diagram associated with it. The vertex u and an edge e are incident with each other as are v and e . If two distinct edges e and f are incident with a common vertex, then they are adjacent edges.

5.2 Binary Tree

A binary tree is a special tree where each non-leaf node can have atmost two child nodes. Most important types of trees which are used to model yes/no, on/off, higher/lower, i.e., binary decisions are binary trees.

A tree data structure in which every node has a maximum of two child nodes is known as a binary tree. It is the most commonly used non-linear data structure. A binary tree could either have only a root node or two disjoint binary trees called the left sub-tree or the right sub-tree. An empty tree could also be a binary tree.

Recursive Definition: "A binary tree is either empty or a node that has left and right sub-trees that are binary trees. Empty trees are represented as boxes (but we will almost always omit the boxes)".



Binary Tree Structure

In a formal way, we can define a binary tree as a finite set of nodes which is either empty or partitioned in to sets of T_0 , T_l , T_r , where T_0 is the root and T_l and T_r are left and right binary trees, respectively.

So, for a binary tree we find that:

1. The maximum number of nodes at level i will be 2^{i-1}
2. If k is the depth of the tree then the maximum number of nodes that the tree can have is $2^k - 1 = 2^{k-1} + 2^{k-2} + \dots + 2^0$

Types of Binary Tree

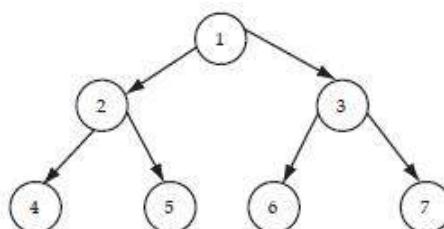
There are two main binary tree and these are:

1. Full binary tree
2. Complete binary tree

Full Binary Tree

A full binary tree is a binary of depth k having $2^k - 1$ nodes. If it has $< 2^k - 1$, it is not a full binary tree.

For $k = 3$, the number of nodes = $2^k - 1 = 2^3 - 1 = 8 - 1 = 7$. A full binary tree with depth $k = 3$ is shown in figure.



A Full Binary Tree

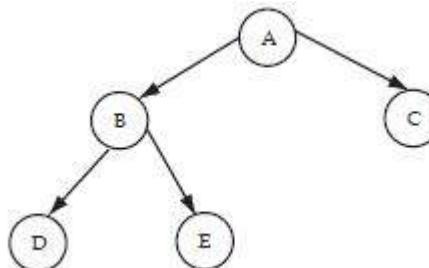
We use numbers from 1 to $2^k - 1$ as labels of the nodes of the tree.

If a binary tree is full, then we can number its nodes sequentially from 1 to $2^k - 1$, starting from the root node, and at every level numbering the nodes from left to right.

Complete Binary Tree

A complete binary tree of depth k is a tree with n nodes in which these n nodes can be numbered sequentially from 1 to n , as if it would have been the first n nodes in a full binary tree of depth k .

A complete binary tree with depth $k = 3$ is shown in Figure



A Complete Binary Tree

Properties of a Binary Tree

Main properties of binary tree are:

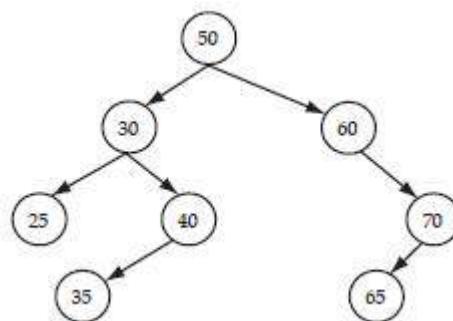
1. If a binary tree contains n nodes, then it contains exactly $n - 1$ edges;
2. A Binary tree of height h has $2^h - 1$ nodes or less.

3. If we have a binary tree containing n nodes, then the height of the tree is at most n and at least $\lceil \log_2(n+1) \rceil$.
4. If a binary tree has n nodes at a level l then, it has at most 2^n nodes at a level $l+1$
5. The total number of nodes in a binary tree with depth k (root has depth zero) is $N = 2^0 + 2^1 + 2^2 + \dots + 2^k = 2^{k+1} - 1$.

5.3 Binary Search Tree

A binary search tree is a binary tree which may be empty, and every node contains an identifier and

1. Identifier of any node in the left sub-tree is less than the identifier of the root
2. Identifier of any node in the right sub-tree is greater than the identifier of the root and the left sub-tree as well as right sub-tree both are binary search trees.



Binary Search Tree

A binary search tree is basically a binary tree, and therefore it can be traversed in inorder, preorder, and postorder. If we traverse a binary search tree in inorder and print the identifiers contained in the nodes of the tree, we get a sorted list of identifiers in the ascending order.

A binary search tree is an important search structure. For example, consider the problem of searching a list. If a list is an ordered then searching becomes faster, if we use a contiguous list and binary search, but if we need to make changes in the list like inserting new entries and deleting old entries. Then it is much slower to use a contiguous list because insertion and deletion in a contiguous list requires moving many of the entries every time. So we may think of using a linked list because it permits insertions and deletions to be carried out by adjusting only few pointers, but in a linked list there is no way to move through the list other than one node at a time hence permitting only sequential access. Binary trees provide an excellent solution to this problem. By making the entries of an ordered list into the nodes of a binary search tree, we find that we can search for a key in $O(n \log n)$ steps.

Creating a Binary Search Tree

We assume that every node a binary search tree is capable of holding an integer data item and the links which can be made pointing to the root of the left and the right sub-tree respectively.

Therefore the structure of the node can be defined using the following declaration:

```

struct tnode
{
    int data;
    tnode *lchild;
    tnode *rchild;
}
  
```

To create a binary search tree we use a procedure named `insert` which creates a new node with the data value supplied as a parameter to it, and inserts into an already existing tree whose root

pointer is also passed as a parameter. The procedure accomplishes this by checking whether the tree whose root pointer is passed as a parameter is empty. If it is empty then the newly created node is inserted as a root node. If it is not empty then it copies the root pointer into a variable temp1, it then stores value of temp1 in another variable temp2, compares the data value of the node pointed to by temp1 with the data value supplied as a parameter, if the data value supplied as a parameter is smaller than the data value of the node pointed to by temp1 then it copies the left link of the node pointed by temp1 into temp1 (goes to the left), otherwise it copies the right link of the node pointed by temp1 into temp1(going to the right). It repeats this process till temp1 becomes nil.

When temp1 becomes nil, the new node is inserted as a left child of the node pointed to by temp2 if data value of the node pointed to by temp2 is greater than data value supplied as parameter. Otherwise the new node is inserted as a right child of node pointed to by temp2. Therefore the insert procedure is

```
void insert(tnode *p, int val)
{
tnode *temp1, *temp2;
if (p == NULL)
{
p = new(tnode);
p->data = val;
p->lchild = NULL;
p->rchild = NULL;
}
else
{
temp1 = p;
while(temp1 != NULL)
{
temp2 = temp1;
if(temp1->data > val)
temp1 = temp1->left;
else
temp1 = temp1->right;
}
if(temp2->data > val)
{
temp2->left = new(tnode);
temp2 = temp2->left;
temp2->data = val;
temp2->left = NULL;
temp2->right= NULL;
}
else
{
temp2->right = new(tnode);
```

```

temp2 = temp2->right;
temp2->data = val;
temp2->left = NULL;
temp2->right = NULL;
}
}
}

```

5.4 Binary Search Tree Operations

The four main operations that we perform on binary trees are:

1. Searching
2. Insertion
3. Deletion
4. Traversal

Searching in Binary Search Trees

In searching, the node being searched is called as key node. We first match the key node with the root node. If the value of the key node is greater than the current node, then we search for it in the right subtree, else we search in the left sub-tree. We continue this process until we find the node or until no nodes are left. The pseudo code for searching a binary search tree is as follows:

Pseudocode for a Binary Search Tree

```

find(X, node){
if(node = NULL)
return NULL
if(X = node:data)
return node
else if(X<node:data)
return find(Y,node:leftChild)
else if(X>node:data)
return find(X,node:rightChild)
}

```

Inserting in Binary Search Trees

To insert a new element in an existing binary search tree, first we compare the value of the new node with the current node value. If the value of the new node is lesser than the current node value, we insert it as a left sub-node. If the value of the new node is greater than the current node value, then we insert it as a right sub-node. If the root node of the tree does not have any value, we can insert the new node as the root node.

Algorithm for Inserting a Value in a Binary Search Tree

1. Read the value for the node that needs to be created and store it in a node called NEW.
2. At first, if (root! =NULL) then root = NEW.
3. If (NEW->value < root->value) then attach NEW node as a left child node of root, else attach NEW node as a right child node of root.
4. Repeat steps 3 and 4 for creating the desired binary search tree completely.

Advanced data structures

When inserting any node in a binary search tree, it is necessary to look for its proper position in the binary search tree. The new node is compared with every node of the tree. If the value of the node which is to be inserted is more than the value of the current node, then the right sub-tree is considered, else the left sub-tree is considered. Once the proper position is identified, the new node is attached as the left or right child node. Let us now discuss the pseudo code for inserting a new element in a binary search tree.

Pseudocode for Inserting a Value in a Binary Search Tree

```
//Purpose: Insert data object X into the Tree
//Inputs: Data object X (to be inserted), binary-search-tree node
//Effect: Do nothing if tree already contains X;
// otherwise, update binary search tree by adding a new node containing data object X
insert(X, node){
    if(node = NULL){
        node = new binaryNode(X,NULL,NULL)
        return
    }
    if(X = node:data)
        return
    else if(X<node:data)
        insert(X, node:leftChild)
    else // X>node:data
        insert(X, node:rightChild)
}
```

Deleting in Binary Search Trees

If the node to be deleted has no children, we can just delete it. If the node to be deleted has one child,

then the node is deleted and the child is connected directly to the parent node.

There are mainly three cases possible for deletion of any node from a binary search tree. They are:

1. Deletion of the leaf node
2. Deletion of a node that has one child
3. Deletion of a node that has two children

We can delete an existing element from a binary search tree using the following pseudocode:

Pseudocode for Deleting a Value from a Binary Search Tree

```
//Purpose: Delete data object X from the Tree
//Inputs: Data object X (to be deleted), binary-search-tree node
//Effect: Do nothing if tree does not contain X;
// otherwise, update binary search tree by deleting the node containing data object X
delete(X, node){
    if(node = NULL) //nothing to do
    return
    if(X<node:data)
        delete(X, node:leftChild)
```

```

else if(X>node:data)
delete(X, node:rightChild)
else { // found the node to be deleted. Take action based on number of node children
if(node:leftChild = NULL and node:rightChild = NULL){
delete node
node = NULL
return
}
else if(node:leftChild = NULL){
tempNode = node
node = node:rightChild
delete tempNode}
else if(node:rightChild = NULL){
tempNode = node
node = node:leftChild
delete tempNode
}
else { //replace node:data with minimum data from right sub-tree
tempNode = findMin(node:rightChild)
node:data = tempNode:data
delete(node:data,node:rightChild)
}
}
}

```

Pseudocode for Finding Minimum Value from a Binary Search Tree

```

//Purpose: return least data object X in the Tree
//Inputs: binary-search-tree node node
// Output: bst-node n containing least data object X, if it exists; NULL otherwise
findMin(node)
{
if(node = NULL) //empty tree
return NULL
if(node:leftChild = NULL)
return node
return findMin(node:leftChild)
}

```

Deleting a node with one child

Step 1 - Find the node to be deleted using search operation

Step 2 - If it has only one child then create a link between its parent node and child node.

Step 3 - Delete the node using free function and terminate the function.

Deleting a node with two children

Step 1 - Find the node to be deleted using search operation

Advanced data structures

Step 2 - If it has two children, then find the largest node in its left subtree (OR) the smallest node in its right subtree.

Step 3 - Swap both deleting node and node which is found in the above step.

Step 4 - Then check whether deleting node came to case 1 or case 2 or else goto step 2

Step 5 - If it comes to case 1, then delete using case 1 logic.

Step 6- If it comes to case 2, then delete using case 2 logic.

Step 7 - Repeat the same process until the node is deleted from the tree.

Binary Search Tree time complexities

Search Operation - $O(n)$

Insertion Operation - $O(1)$

Deletion Operation - $O(n)$

Application of a Binary Search Tree

1. A prominent data structure used in many systems programming applications for representing and managing dynamic sets.

2. Average case complexity of Search, Insert, and Delete Operations is $O(\log n)$, where n is the number of nodes in the tree.

One of the applications of a binary search tree is the implementation of a dynamic dictionary.

A dictionary is an ordered list which is required to be searched frequently, and is also required to be updated (insertions and deletions) frequently. Hence can be very well implemented using a binary search tree, by making the entries of dictionary into the nodes of binary search tree. A more efficient implementation of a dynamic dictionary involves considering a key to be a sequence of characters, and instead of searching by comparison of entire keys, we use these characters to determine a multi-way branch at each step, this will allow us to make a 26-way branching according the first letter, followed by another branch according to the second letter and so on.

Summary

- Search trees are data structures that support many dynamic-set operations such as searching, finding the minimum or maximum value, inserting, or deleting a value.
- In a binary search tree, for a given node n , each node to the left has a value lesser than n and each node to the right has a value greater than n .
- The time taken to perform operations on a binary search tree is directly proportional to the height of the tree.
- Binary trees provide an excellent solution to this problem. By making the entries of an ordered list into the nodes of a binary tree, we shall find that we can search for a target key in $O(\log n)$ steps, just as with binary search, and we shall obtain algorithms for inserting and deleting entries also in time $O(\log n)$.

Keywords

- **Binary Search Tree:** A binary search tree is a binary tree which may be empty, and every node contains an identifier.
- **Searching:** Searching for the key in the given binary search tree, start with the root node and compare the key with the data value of the root node.
- **Degree of a tree:** The highest degree of a node appearing in the tree.

- **Inorder:** A tree traversing method in which the tree is traversed in the order of left-tree, node and then right-tree.
- **Level of a node:** The number of nodes that must be traversed to reach the node from the root.
- **Root node:** The node in a tree which does not have a parent node.
- **Tree:** A two-dimensional data structure comprising of nodes where one node is the root and rest of the nodes form two disjoint sets each of which is a tree.

Self Assessment

1. Tree is a _____ hierarchical data structure.
 - A. Linear
 - B. Nonlinear
 - C. Abstract
 - D. All of above
2. Data access is quick and easier in
 - A. Nonlinear data structure
 - B. Linear data structure
 - C. Both Linear and Nonlinear
 - D. None of above
3. Types of trees are
 - A. Binary Tree
 - B. Binary Search Tree
 - C. AVL Tree
 - D. All of above
4. Binary search tree is also called.
 - A. Ordered
 - B. Sorted
 - C. Both ordered and sorted binary tree
 - D. None of above
5. value of the nodes in the left sub-tree is less than the value of the root is
 - A. General Tree
 - B. Binary Tree
 - C. Binary Search Tree
 - D. None of above
6. Types of Binary Trees are
 - A. Full binary tree
 - B. Complete binary tree

- C. Perfect binary tree
 - D. All of above
7. Which is not Binary Tree operation?
- A. Search
 - B. Peek
 - C. Insertion
 - D. Deletion
8. Binary Search Tree applications are
- A. In multilevel indexing in the database.
 - B. For dynamic sorting.
 - C. It is used to implement various searching algorithms.
 - D. All of above
9. Binary Search Tree time complexity for search operation is
- A. $O(n)$
 - B. $O(1)$
 - C. $O(2)$
 - D. None of above
10. Binary Search Tree time complexity for insert operation is
- A. $O(0)$
 - B. $O(1)$
 - C. $O(2)$
 - D. None of above
11. What is the worst case time complexity for Delete operation?
- A. $O(0)$
 - B. $O(1)$
 - C. $O(n)$
 - D. None of above
12. Which is incorrect statement about Binary search tree.
- A. The left and right sub-trees should also be binary search trees
 - B. In order sequence gives decreasing order of elements
 - C. The left child is always lesser than its parent
 - D. The right child is always greater than its parent
13. To arrange binary tree in ascending order...
- A. Pre order traversal only
 - B. Post order traversal only

- C. In order traversal only
 D. None of above
14. Which of the following is not component of binary tree
 A. Data element
 B. Pointer to right subtree
 C. Super tree
 D. Pointer to left subtree
15. Which of the following is not a type of tree data structure
 A. General Tree
 B. Primary Tree
 C. Binary Tree
 D. Binary Search Tree
 E.

Answers for self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. B | 2. A | 3. D | 4. C | 5. C |
| 6. D | 7. B | 8. D | 9. A | 10. B |
| 11. C | 12. B | 13. C | 14. C | 15. B |

Review Questions

1. define tree with suitable example.
2. Draw a binary tree with six child node.
4. Discuss degree of tree.
5. Explain representation of tree.
6. what are the application of tree.
7. Discuss time complexity of Binary search tree.



Further Readings

Data Structures and Efficient Algorithms, Burkhard Monien, Thomas Ottmann, Springer.

Kruse Data Structure & Program Design, Prentice Hall of India, New Delhi

Mark Allen Weis: Data Structure & Algorithm Analysis in C Second Edition. Addison-Wesley publishing

RG Dromey, How to Solve it by Computer, Cambridge University Press.

Shi-kuo Chang, Data Structures and Algorithms, World Scientific

Sorenson and Tremblay: An Introduction to Data Structure with Algorithms.

Thomas H. Cormen, Charles E. Leiserson & Ronald L. Rivest: Introduction to Algorithms. Prentice-Hall of India Pvt. Limited, New Delhi

Timothy A. Budd, Classic Data Structures in C++, Addison Wesley.



Web Links

www.en.wikipedia.org

www.webopedia.com

<https://www.programiz.com/>

<https://www.javatpoint.com/data-structure-stack>

https://www.tutorialspoint.com/data_structures_algorithms/stack_algorithm.htm

Unit 06: Tree Data Structure 1

CONTENTS

- Objectives
- Introduction
- 6.1 AVL Tree
- 6.2 AVL Operation
- 6.3 Applications of AVL Trees
- 6.4 B-tree
- 6.5 Operations on B-trees
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to:

- State about AVL tree
- Describe balancing operations
- Discuss B-tree
- Learn B-tree properties and operations

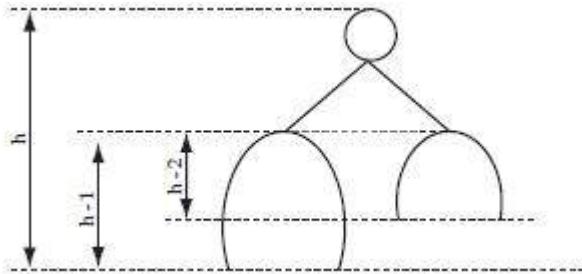
Introduction

One of the most essential data structures is the tree, which is used to conduct operations like insertion, deletion, and searching of items efficiently. Construction of a well-balanced tree for sorting all data is not practicable when working with a huge number of data, though. As a result, only valuable data is saved as a tree, and the actual volume of data used changes over time as new data is inserted and old data is deleted. It is possible to conduct traversals, insertions, and deletions without utilizing either stack or recursion in some circumstances where the NULL link to a binary tree to special links is referred to as threads.

6.1 AVL Tree

An AVL tree is another balanced binary search tree. AVL Tree is invented in 1962. It takes its name from the initials of its inventors – Adelson, Velskii and Landis. An AVL tree has the following properties:

1. The sub-trees of every node differ in height by at most one level.
2. Every sub-tree is an AVL tree.

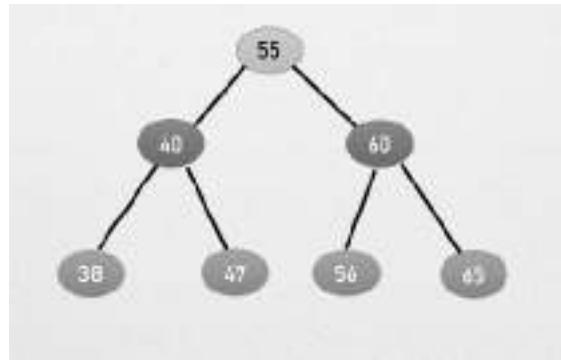


Here, the height of the tree is h . Height of one subtree is $h-1$ while that of another subtree of the same node is $h-2$, differing from each other by just 1. Therefore, it is an AVL tree.

AVL Tree is defined as height balanced binary search tree. In AVL tree each node is associated with a balance factor which is calculated by subtracting the height of its right sub-tree from that of its left sub-tree.

Why AVL Tree

AVL tree controls the height of the binary search tree. The time taken for all operations in a binary search tree of height h is $O(h)$. For skewed BST it can be extended to $O(n)$ (worst case). By limiting this height to $\log n$, AVL tree imposes an upper bound on each operation to be $O(\log n)$ where n is the number of nodes.



Balance Factor

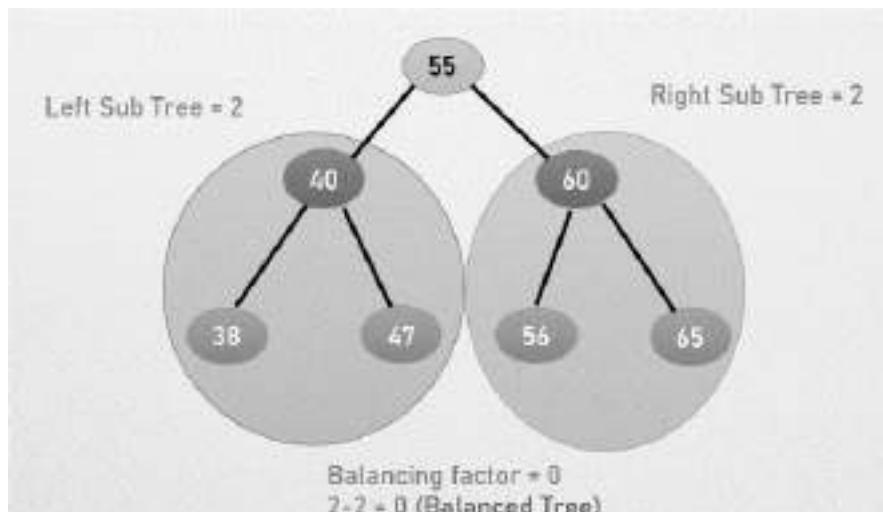
Balance factor of a node in an AVL tree is the difference between the height of the left sub tree and that of the right sub tree of that node.

Balance Factor = (Height of Left Sub tree - Height of Right Sub tree) or (Height of Right Sub tree - Height of Left Sub tree). Balance factor value are: -1, 0 or 1.

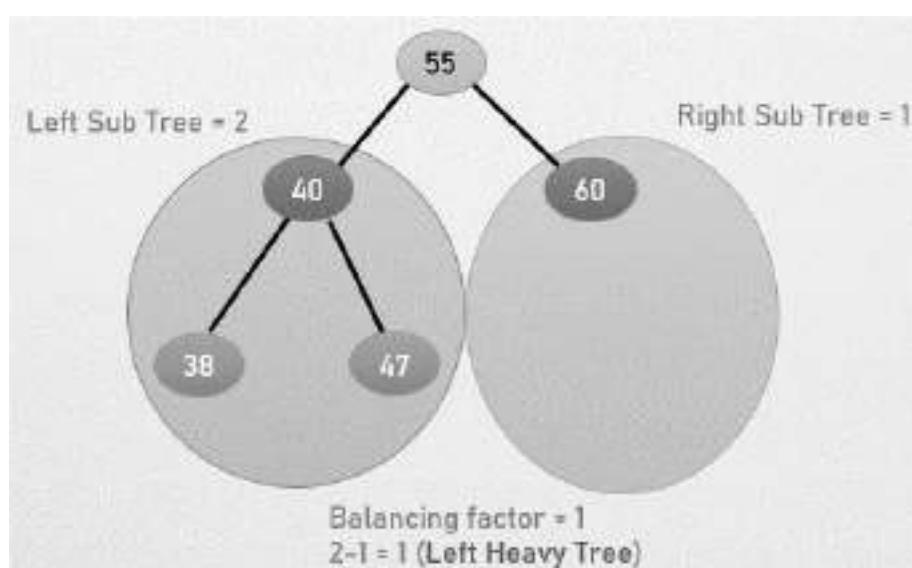
If balance factor of any node is 1, it means that the left sub-tree is one level higher than the right sub-tree.

If balance factor of any node is 0, it means that the left sub-tree and right sub-tree contain equal height.

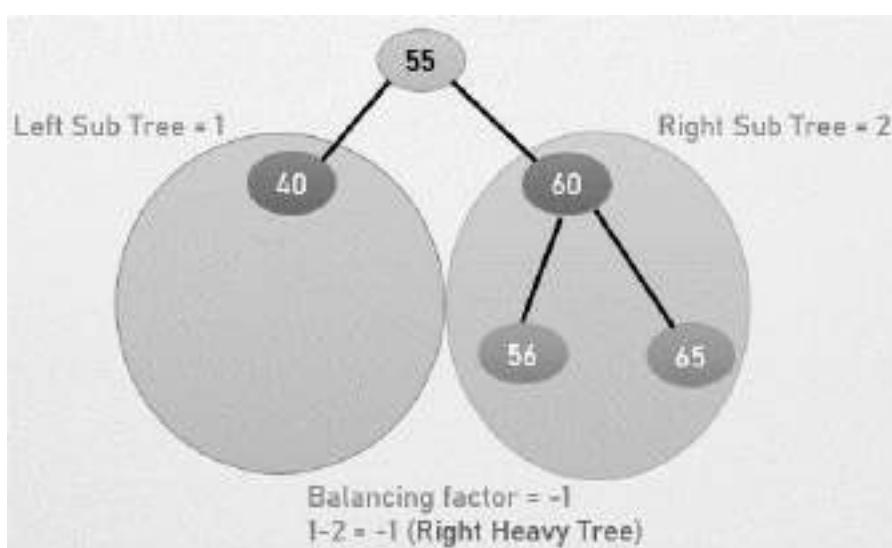
If balance factor of any node is -1, it means that the left sub-tree is one level lower than the right sub-tree.



Balanced Tree



AVL tree (Left Heavy Tree)



AVL Tree (Right Heavy Tree)

AVL Tree Rotations for balancing

Rotations are performed in AVL tree only in case if Balance Factor is other than -1, 0, and 1.

Left rotation

Right rotation

Left-Right rotation

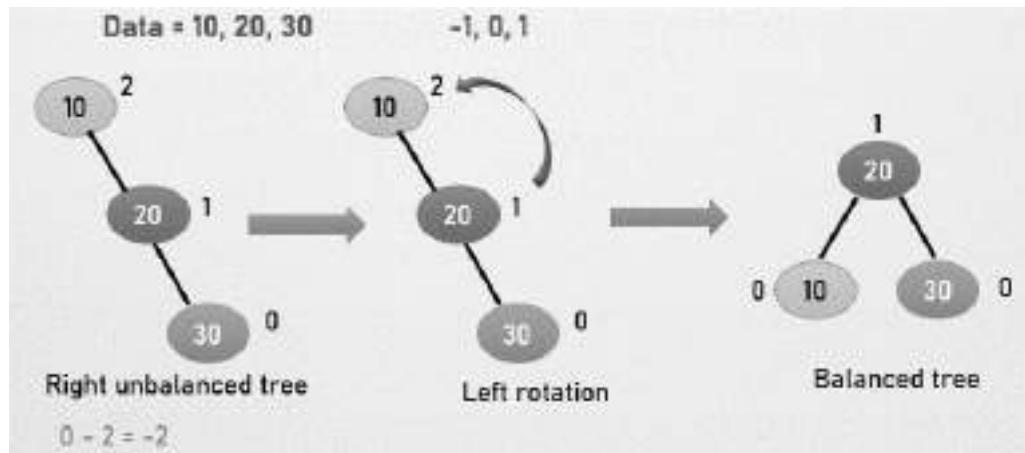
Right-Left rotation

The Left rotation and Right rotation are **single rotations**.

Left-Right rotation and Right-Left rotation are **double rotations**.

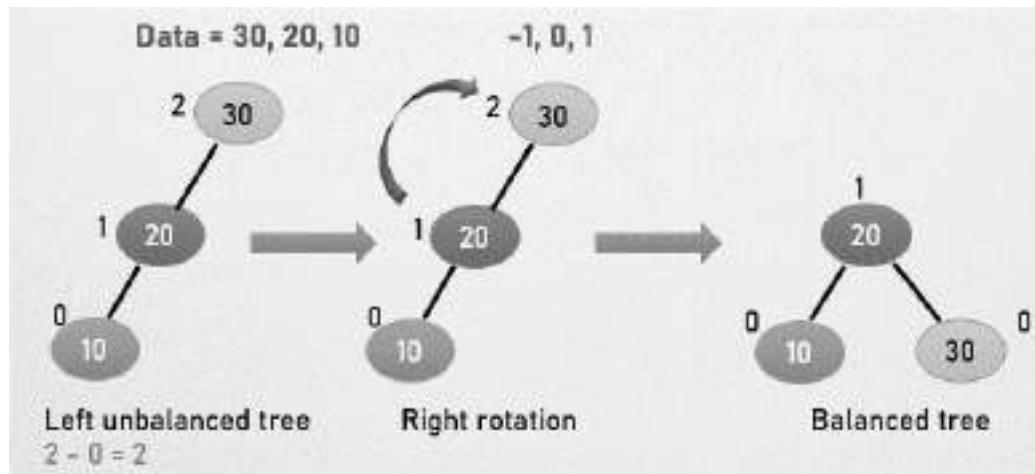
Left rotation

If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotation.



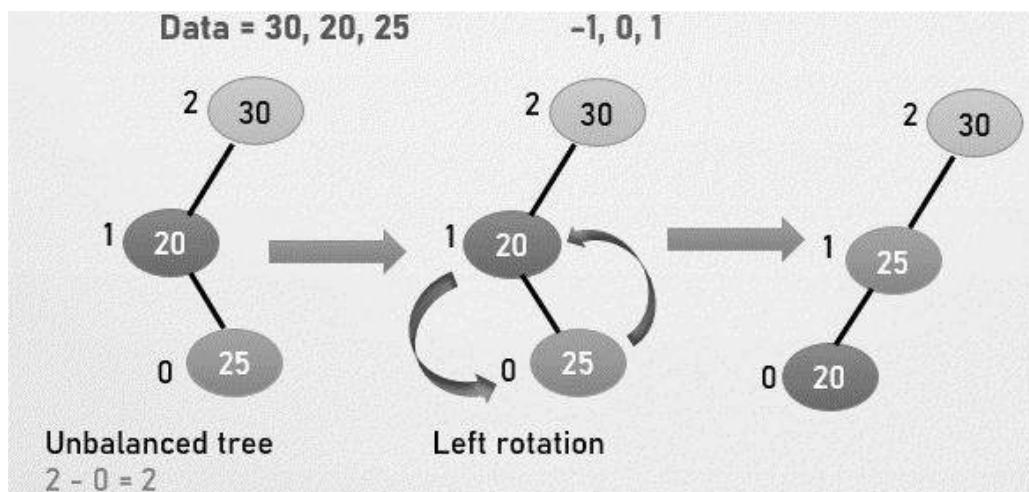
Right rotation

AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.

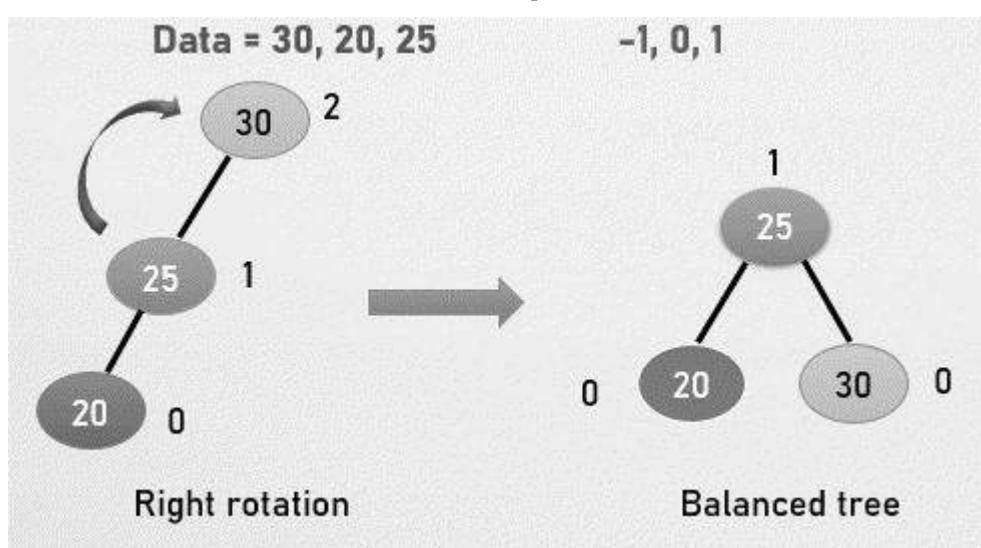


Left-Right rotation

Double rotations are slightly complex rotations. To understand them better, we should take note of each action performed while rotation. A left-right rotation is a combination of left rotation followed by right rotation.



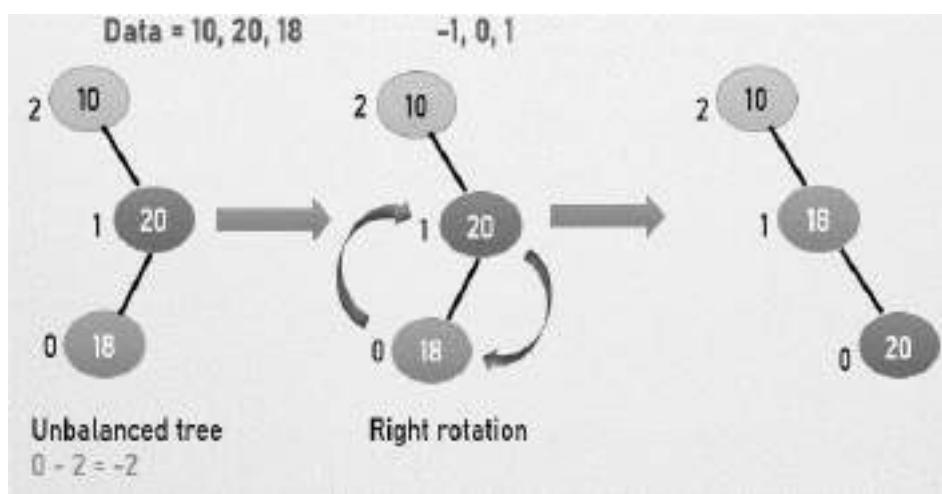
Step - 1



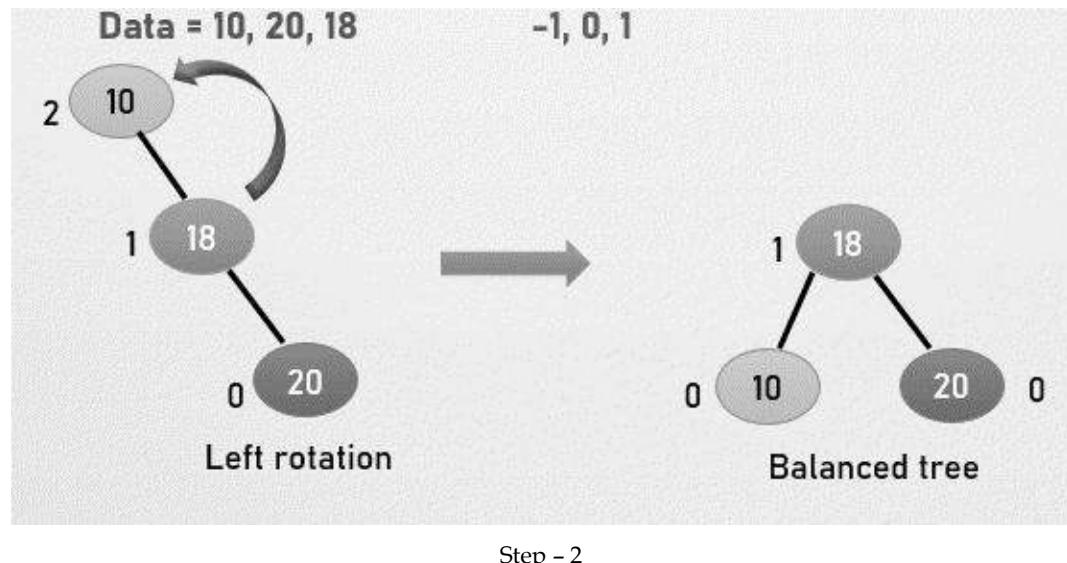
Step - 2

Right-Left rotation

It is a combination of right rotation followed by left rotation.



Step - 1



6.2 AVL Operation

Insertion into AVL Trees

To implement AVL trees, you need to maintain the height of each node. You insert into an AVL tree by performing a standard binary tree insertion. When you're done, you check each node on the path from the new node to the root. If that node's height hasn't changed because of the insertion, then you are done. If the node's height has changed, but it does not violate the balance property, then you continue checking the next node in the path. If the node's height has changed and it now violates the balance property, then you need to perform one or two rotations to fix the problem, and then you are done.

Insertion in AVL is same as binary search tree. Insertion may lead to violation in the tree property and therefore the tree may need balancing. The tree can be balanced by applying rotations.

Deletion

When you delete a node, there are three things that can happen to the parent:

1. Its height is decremented by one.
2. Its height doesn't change and it stays balanced.
3. Its height doesn't change, but it becomes imbalanced.

You handle these three cases in different ways:

1. The parent's height is decremented by one. When this happens, you check the parent's parent: you keep doing this until you return or you reach the root of the tree.
2. The parent's height doesn't change and it stays balanced. When this happens you may return - deletion is over.
3. The parent's height doesn't change, but it becomes imbalanced. When this happens, you have to rebalance the subtree rooted at the parent. After rebalancing, the subtree's height may be one smaller than it was originally. If so, you must continue checking the parent's parent.

To rebalance, you need to identify whether you are in a zig-zig situation or a zig-zag situation and rebalance accordingly.

Complexities of Different Operations on an AVL Tree

The rotation operations (left and right rotate) take constant time as only a few pointers are being changed there. Updating the height and getting the balance factor also takes constant time.

Insertion

$O(\log n)$

Deletion

$O(\log n)$

Search

$O(\log n)$

6.3 Applications of AVL Trees

AVL trees are applied in the following situations:

1. There are few insertion and deletion operations
2. Short search time is needed
3. Input data is sorted or nearly sorted

AVL tree structures can be used in situations which require fast searching. But, the large cost of rebalancing may limit the usefulness.

Consider the following:

1. A classic problem in computer science is how to store information dynamically so as to allow for quick look up. This searching problem arises often in dictionaries, telephone directory, symbol tables for compilers and while storing business records etc. The records are stored in a balanced binary tree, based on the keys (alphabetical or numerical) order.

The balanced nature of the tree limits its height to $O(\log n)$, where n is the number of inserted records.

2. AVL trees are very fast on searches and replacements. But, have a moderately high cost for addition and deletion. If application does a lot more searches and replacements than it does addition and deletions, the balanced (AVL) binary tree is a good choice for a data structure.

3. AVL tree also has applications in file systems.

Advantages of AVL tree

The height of the AVL tree is always balanced. The height never grows beyond $\log N$, where N is the total number of nodes in the tree.

Search time complexity is better as compared to Binary Search trees.

AVL trees have self-balancing capabilities.

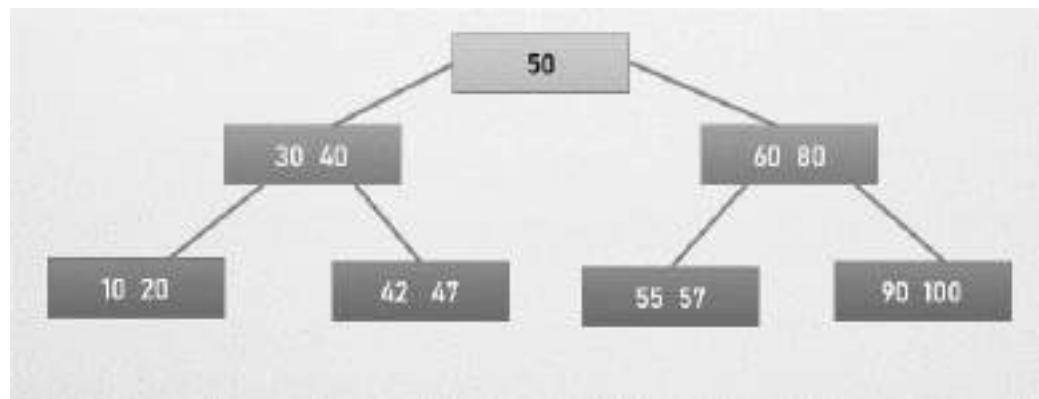
6.4 B-tree

A B-tree is a tree data structure that keeps data sorted and allows insertions and deletions that is logarithmically proportional to file size. It is commonly used in databases and file systems.

In B-trees, internal nodes can have a variable number of child nodes within some pre-defined range. When data is inserted or removed from a node, its number of child nodes changes. In order to maintain the pre-defined range, internal nodes may be joined or split. Because a range of child nodes is permitted, B-trees do not need re-balancing as frequently as other self balancing search trees, but may waste some space, since nodes are not entirely full. The lower and upper bounds on the number of child nodes are typically fixed for a particular implementation.

A B-tree is kept balanced by requiring that all leaf nodes are at the same depth. This depth will increase slowly as elements are added to the tree, but an increase in the overall depth is infrequent, and results in all leaf nodes being one more hop further removed from the root.

B-trees are balanced trees that are optimized for situations when part or the entire tree must be maintained in secondary storage such as a magnetic disk. Since disk accesses are expensive (time consuming) operations, a b-tree tries to minimize the number of disk accesses.



Structure of B-trees

Unlike a binary-tree, each node of a b-tree may have a variable number of keys and children. The keys are stored in non-decreasing order. Each key has an associated child that is the root of a subtree containing all nodes with keys less than or equal to the key but greater than the preceding key. A node also has an additional rightmost child that is the root for a subtree containing all keys greater than any keys in the node.

A b-tree has a minimum number of allowable children for each node known as the minimization factor. If t is this minimization factor, every node must have at least $t - 1$ keys. Under certain circumstances, the root node is allowed to violate this property by having fewer than $t - 1$ keys.

Every node may have at most $2t - 1$ keys or, equivalently, $2t$ children. Since each node tends to have a large branching factor (a large number of children), it is typically necessary to traverse relatively few nodes before locating the desired key. If access to each node requires a disk access, then a b-tree will minimize the number of disk accesses required.

The minimization factor is usually chosen so that the total size of each node corresponds to a multiple of the block size of the underlying storage device. This choice simplifies and optimizes disk access. Consequently, a b-tree is an ideal data structure for situations where all data cannot reside in primary storage and accesses to secondary storage are comparatively expensive (or time consuming).

Why B Tree

B-Trees is used to reduce the number of disk accesses. Most of the tree operations (search, insert, delete, max, min) require $O(h)$ disk accesses where h is the height of the tree. B-tree is a fat tree. The height of B-Trees is kept low by putting maximum possible keys in a B-Tree node.

Data structures like binary search tree, avl tree, red-black tree, etc. can store only one key in one node. If you have to store a large number of keys, then the height of such trees becomes very large and the access time increases.

The height of the B-tree is low so total disk accesses for most of the operations are reduced significantly compared to balanced Binary Search Trees like AVL Tree, Red-Black Tree,etc.

B Tree properties

B-Tree of Order m has the following properties:

- 1 - All leaf nodes must be at same level.
- 2 - All nodes except root must have at least $[m/2]-1$ keys and maximum of $m-1$ keys.
- 3 - All non leaf nodes except root (i.e. all internal nodes) must have at least $m/2$ children.
- 4 - If the root node is a non leaf node, then it must have at least 2 children.
- 5 - A non leaf node with $n-1$ keys must have n number of children.
- 6 - All the key values in a node must be in Ascending Order.

6.5 Operations on B-trees

The algorithms for the search, create, and insert operations are shown below. Note that these algorithms are single pass; in other words, they do not traverse back up the tree. Since b-trees strive to minimize disk accesses and the nodes are usually stored on disk, this single-pass approach will reduce the number of node visits and thus the number of disk accesses. Simpler double-pass approaches that move back up the tree to fix violations are possible.

Since all nodes are assumed to be stored in secondary storage (disk) rather than primary storage (memory), all references to a given node be preceded by a read operation denoted by Disk-Read. Similarly, once a node is modified and it is no longer needed, it must be written out to secondary storage with a write operation denoted by Disk-Write. The algorithms below assume that all nodes referenced in parameters have already had a corresponding Disk-Read operation. New nodes are created and assigned storage with the Allocate-Node call. The implementation details of the Disk-Read, Disk-Write, and Allocate-Node functions are operating system and implementation dependent.

Search Operation

The search operation on a b-tree is analogous to a search on a binary tree. Instead of choosing between a left and a right child as in a binary tree, a b-tree search must make an n-way choice. The correct child is chosen by performing a linear search of the values in the node. After finding the value greater than or equal to the desired value, the child pointer to the immediate left of that value is followed. If all values are less than the desired value, the rightmost child pointer is followed. Of course, the search can be terminated as soon as the desired node is found. Since the running time of the search operation depends upon the height of the tree, B-Tree-Search is $O(\log n)$.

```
B-Tree-Search(x, k)
```

```
i<- 1
while i<= n[x] and k >keyi[x]
do i<- i + 1
if i<= n[x] and k = keyi[x]
then return (x, i)
if leaf[x]
then return NIL
else Disk-Read(ci[x])
return B-Tree-Search(ci[x], k)
```

Search algorithm

Let the key (the value) to be searched by "X".

Start searching from the root and recursively traverse down.

If X is lesser than the root value, search left sub tree, if X is greater than the root value, search the right sub tree.

If the node has the found X, simply return the node.

If the X is not found in the node, traverse down to the child with a greater Key.

If X is not found in the tree, we return NULL.

Insertion Operation

Insertions in B-Tree performed only at the leaf node level. Inserting operation performed with two steps: searching the appropriate node to insert the element and splitting the node if required.

Insertion algorithm

Check whether tree is Empty.

If tree is Empty, then create a new node with new key value and insert it into the tree as a root node

If tree is not empty, then, find the appropriate leaf node at which the node can be inserted.

If the leaf node contain less than $m-1$ keys then insert the element in the increasing order.

Else, if the leaf node contains $m-1$ keys, then follow the following steps.

- Insert the new element in the increasing order of elements.

- Split the node into the two nodes at the median.

- Push the median element upto its parent node.

- If the parent node also contain $m-1$ number of keys, then split it too by following the same steps.

Deletion operation

In case of deletion from B-Tree user need to follow more rule as compared to search and insertion.

Three case for deletion from B-Tree

- If the key is in the leaf node

- If the key is in an internal node

- If the key is in a root node

Key is in the leaf node, case-1

Target is in the leaf node, more than min keys.

Deleting this will not violate the property of B Tree

Target is in leaf node, it has min key nodes

Deleting this will violate the property of B Tree

Target node can borrow key from immediate left node, or immediate right node (sibling)

The sibling will say yes if it has more than minimum number of keys

The key will be borrowed from the parent node, the max value will be transferred to a parent, the max value of the parent node will be transferred to the target node, and remove the target value

Target is in the leaf node, but no siblings have more than min number of keys Search for key

Merge with siblings and the minimum of parent nodes

Total keys will be now more than min

The target key will be replaced with the minimum of a parent node

Key is in an internal node, case-2

Either choose, in-order predecessor or in-order successor

In case of in-order predecessor, the maximum key from its left sub tree will be selected

In case of in-order successor, the minimum key from its right sub tree will be selected

If the target key's in-order predecessor has more than the min keys, only then it can replace the target key with the max of the in-order predecessor

If the target key's in-order predecessor does not have more than min keys, look for in-order successor's minimum key.

If the target key's in-order predecessor and successor both have less than min keys, then merge the predecessor and successor.

Key is in a root node, Case- 3

Replace with the maximum element of the in-order predecessor sub tree

If, after deletion, the target has less than min keys, then the target node will borrow max value from its sibling via sibling's parent.

The max value of the parent will be taken by a target, but with the nodes of the max value of the sibling.

Summary

- AVL tree controls the height of the binary search tree. The time taken for all operations in a binary search tree of height h is $O(h)$.
- In AVL tree each node is associated with a balance factor which is calculated by subtracting the height of its right sub-tree from that of its left sub-tree.
- B-trees are balanced trees that are optimized for situations when part or the entire tree must be maintained in secondary storage such as a magnetic disk.
- A B-tree is a specialized multiway tree designed especially for use on disk. In a B-tree each node may contain a large number of keys. The number of subtrees of each node, then, may also be large.
- A B-tree is designed to branch out in this large number of directions and to contain a lot of keys in each node so that the height of the tree is relatively small.
- This means that only a small number of nodes must be read from disk to retrieve an item.
- The goal is to get fast access to the data, and with disk drives this means reading a very small number of records. Note that a large node size (with lots of keys in the node) also fits with the fact that with a disk drive one can usually read a fair amount of data at once.

Keywords

B-Tree Algorithms: A B-tree is a data structure that maintains an ordered set of data and allows efficient operations to find, delete, insert, and browse the data.

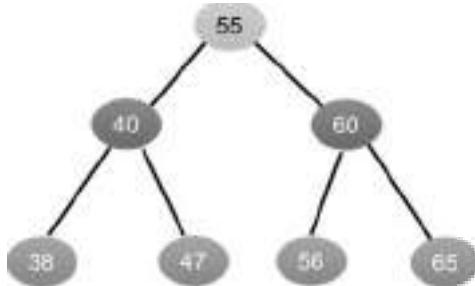
B-trees: B-trees are balanced trees that are optimized for situations when part or the entire tree must be maintained in secondary storage such as a magnetic disk.

SelfAssessment

1. AVL Tree is invented in
 - A. 1955
 - B. 1966
 - C. 1962
 - D. None of above
2. Which statement is true about AVL tree?
 - A. AVL tree controls the height of the binary search tree.
 - B. The time taken for all operations in a binary search tree of height h is $O(h)$.
 - C. For skewed BST it can be extended to $O(n)$ (worst case).
 - D. All of above
3. Balance Factor values in AVL tree is
 - A. -1
 - B. 0

- C. 1
- D. All of above

4. The balance factor in diagram is



- A. 1
- B. 0
- C. 2
- D. None of above

5. AVL Tree Rotations are

- A. Right rotation
- B. Left-Right rotation
- C. Right-Left rotation
- D. All of above

6. Left rotation and Right rotation are

- A. Double rotations
- B. Single rotation
- C. Triple rotation
- D. None of above

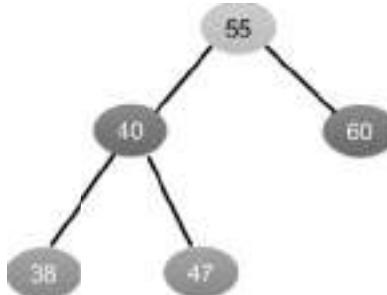
7. Which statement is true about AVL tree?

- A. AVL tree is a self-balancing Binary Search Tree.
- B. AVL Tree is defined as height balanced binary search tree.
- C. In AVL tree each node is associated with a balance factor.
- D. All of above

8. Left-Right rotation and Right-Left rotation are

- A. Single rotation
- B. Triple rotation
- C. Double rotations
- D. None of above

9. The balance factor in diagram is



- A. 1
- B. 0
- C. 2
- D. None of above

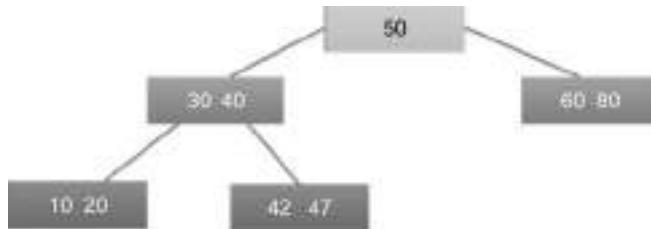
10. B Tree properties are

- A. All leaf nodes must be at same level
- B. All non-leaf nodes except root
- C. A non-leaf node with $n-1$ keys must have n number of children
- D. All of above

11. Which statement is true about B-tree?

- A. Each node can contain more than one key
- B. Each node can have more than two children.
- C. A B Tree of order m can have at most $m-1$ keys and m children.
- D. All of above

12. Diagram represents correct B-tree



- A. True
- B. False

13. Which is not B-tree operation

- A. Search
- B. Insert
- C. Data manipulation
- D. Delete

14. Insertion Operation can performed in B-tree at
- Root node
 - Leaf node
 - Both root and leaf node
 - None of above
15. What are the different cases for deletion from B-Tree?
- If the key is in the leaf node
 - If the key is in an internal node
 - If the key is in a root node
 - All of above

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. C | 2. D | 3. D | 4. B | 5. D |
| 6. B | 7. D | 8. C | 9. A | 10. B |
| 11. D | 12. B | 13. C | 14. B | 15. D |

Review Questions

- define AVL tree and its advantages.
- How AVL tree is different from B-tree.
- Describe the deletion of an item from b-trees.
- Describe of structure of B-tree. Also explain the operation of B-tree.
- Explain insertion of an item in b-trees.
- Differentiate between Left Heavy Tree and right Heavy Tree with example.
- Discuss different AVL tree rotations with suitable diagram.



Further Readings

Data Structures and Efficient Algorithms, Burkhard Monien, Thomas Ottmann, Springer.

Kruse Data Structure & Program Design, Prentice Hall of India, New Delhi

Mark Allen Weles: Data Structure & Algorithm Analysis in C Second Adition. Addison-Wesley publishing

RG Dromey, How to Solve it by Computer, Cambridge University Press.

Shi-kuo Chang, Data Structures and Algorithms, World Scientific

Sorenson and Tremblay: An Introduction to Data Structure with Algorithms.

Thomas H. Cormen, Charles E, Leiserson& Ronald L. Rivest: Introduction to Algorithms. Prentice-Hall of India Pvt. Limited, New Delhi

Timothy A. Budd, Classic Data Structures in C++, Addison Wesley.



Web Links

www.en.wikipedia.org

www.webopedia.com

<https://www.programiz.com/>

<https://www.javatpoint.com/data-structure-stack>

https://www.tutorialspoint.com/data_structures_algorithms/stack_algorithm.htm

Unit 07: Tree Data Structure 2

CONTENTS

- Objectives
- Introduction
- 7.1 Red-Black Tree
- 7.2 Red-Black Tree Properties
- 7.3 Red-Black Tree Operations
- 7.4 Splay Trees
- 7.5 Operations
- 7.6 Rotations in Splay Tree
- 7.7 2-3 Trees
- 7.8 Properties of 2-3 Trees
- 7.9 2-3 Tree Operations
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to:

- State about red-black trees
- Learn splay tree and its operations
- Discuss 2-3 trees and its properties
- Learn operations on 2-3 trees

Introduction

Recall that, for binary search trees, although the average-case times for the lookup, insert, and delete methods are all $O(\log N)$, where N is the number of nodes in the tree, the worst-case time is $O(N)$. We can guarantee $O(\log N)$ time for all three methods by using a balanced tree -- a tree that always has height $O(\log N)$ -- instead of a binary search tree.

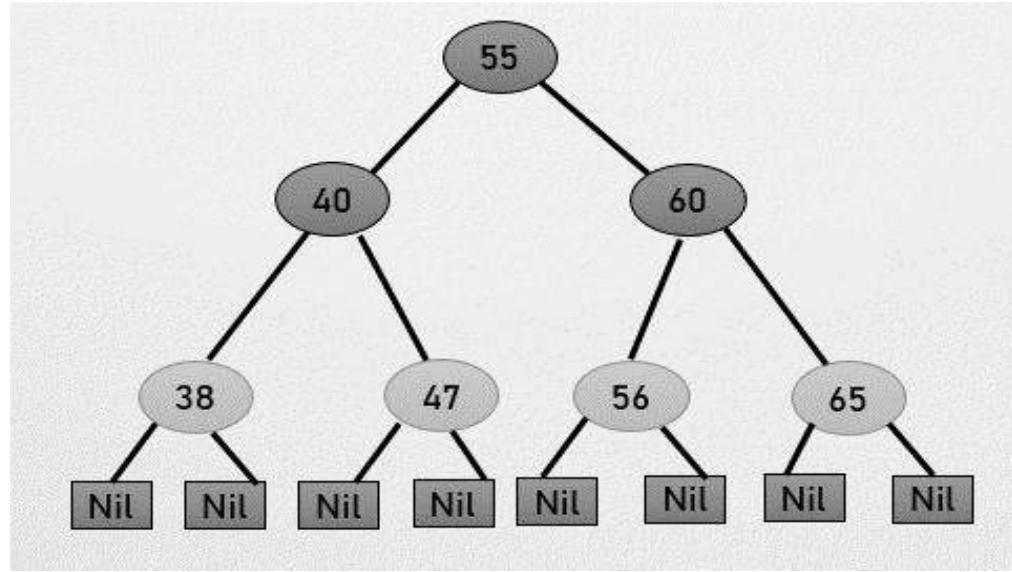
A number of different balanced trees have been defined, including AVL trees, 2-4 trees, and B trees. Here we will look at yet another kind of balanced tree called a red-black tree. The important idea behind all of these trees is that the insert and delete operations may restructure the tree to keep it balanced. So lookup, insert, and delete will always be logarithmic in the number of nodes but insert and delete may be more complicated than for binary search trees.

7.1 Red-Black Tree

A Red Black Tree is a self-balancing binary search tree with one extra attribute for each node: the colour, which is either red or black. It was invented in 1972 by Rudolf Bayer. Colours in tree are used to ensure that the tree remains balanced during insertion and deletion.

A Red Black Tree is a self-balancing binary search tree in which each node has a red or black colour. The red black tree satisfies all of the features of the binary search tree, but it also has several additional properties. A Red-Black tree's height is $O(\log n)$, where (n is the number of nodes in the tree). Red-black trees are one of many search-tree schemes that are "balance" in order to guarantee that basic dynamic-set operations take $O(\lg n)$ time in the worst case.

BST operations take $O(h)$ time where h is the height of the BST. Cost of these operations may become $O(n)$ for a skewed Binary tree. If we make sure that the height of the tree remains $O(\log n)$ after every insertion and deletion, then an upper bound of $O(\log n)$ for all these operations. The height of a Red-Black tree is always $O(\log n)$ where n is the number of nodes in the tree.



7.2 Red-Black Tree Properties

- Red - Black Tree must be a Binary Search Tree.
- The root of the tree is always black.
- The children of Red colored node must be colored BLACK. (There should not be two consecutive RED nodes)
- Every new node must be inserted with RED color.
- Every leaf (e.i. NULL node) must be colored BLACK.
- In all the paths of the tree, there should be same number of BLACK colored nodes.

Each node has the following attributes:

Color

Key

Left Child

Right Child

Parent (except root node)

Three Invariants

A red/black tree is a binary search tree in which each node is colored either red or black. At the interface, we maintain three invariants:

Ordering Invariant This is the same as for binary search trees: all the keys to left of a node are smaller, and all the keys to the right of a node are larger than the key at the node itself.

Height Invariant The number of black nodes on every path from the root to each leaf is the same. We call this the black height of the tree.

Color Invariant No two consecutive nodes are red. The balance and color invariants together imply that the longest path from the root to a leaf is at most twice as long as the shortest path. Since insert and search in a binary search tree have time proportional to the length of the path from the root to the leaf, this guarantees $O(\log(n))$ times for these operations, even if the tree is not perfectly balanced. We therefore refer to the height and color invariants collectively as the balance invariant.

7.3 Red-Black Tree Operations

Insertion

Deletion

Search

Recolor and Rotation performed on insertion and deletion operation as per Red-Black Tree properties.

Insertion operation

The insertion operation in Red Black Tree is similar to the Binary Search Tree. Every new node must be inserted with the color RED.

After every insertion operation, we need to check all the properties of Red-Black Tree. If all the properties are satisfied then we go to next operation otherwise we perform the following operation to make it Red Black Tree.

1. Recolor
2. Rotation
3. Rotation followed by Recolor

Steps for Red-black tree insertion operations

Check whether tree is Empty.

If tree is Empty then insert the newNode as Root node with color Black and exit from the operation.

If tree is not empty then insert the newNode as leaf node with color Red.

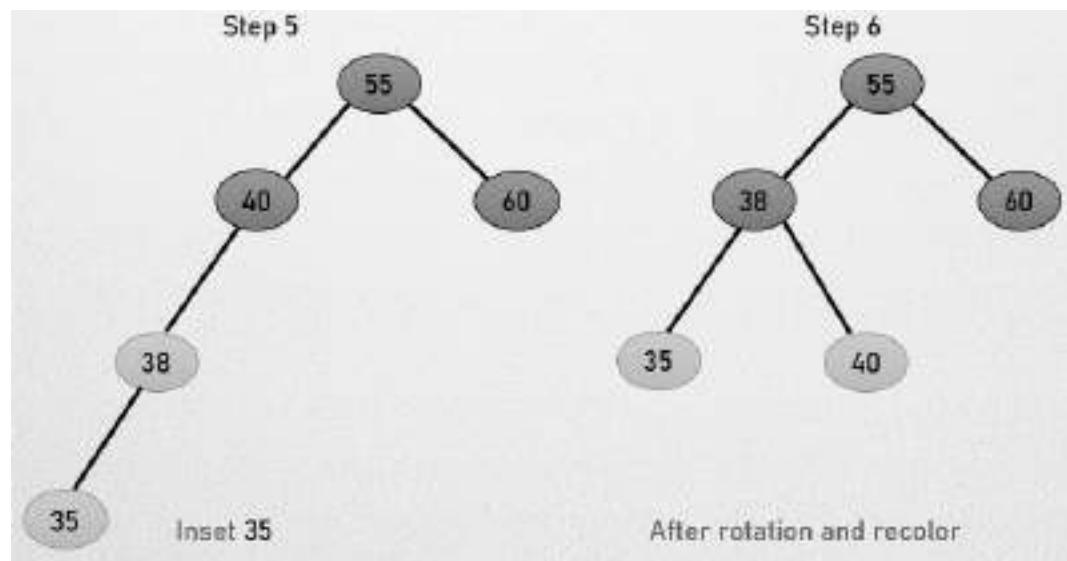
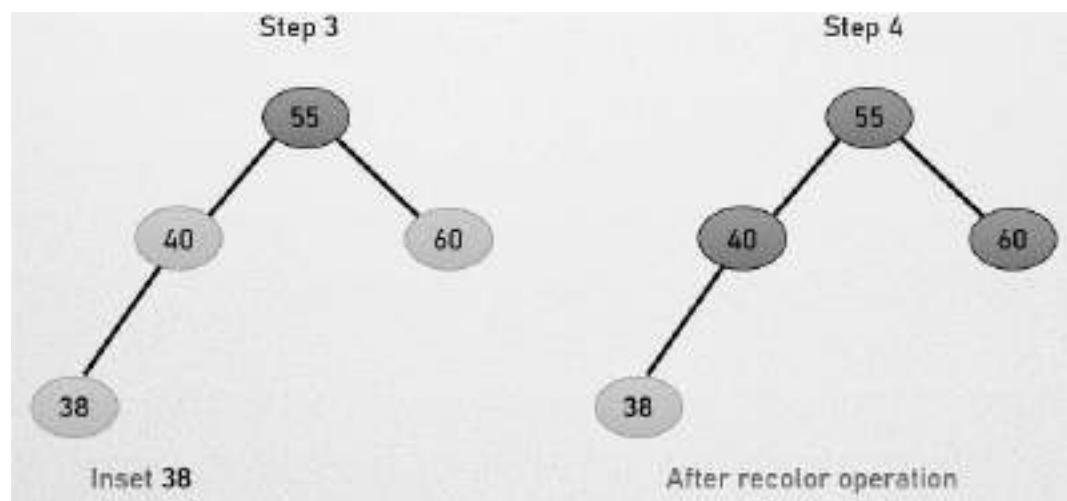
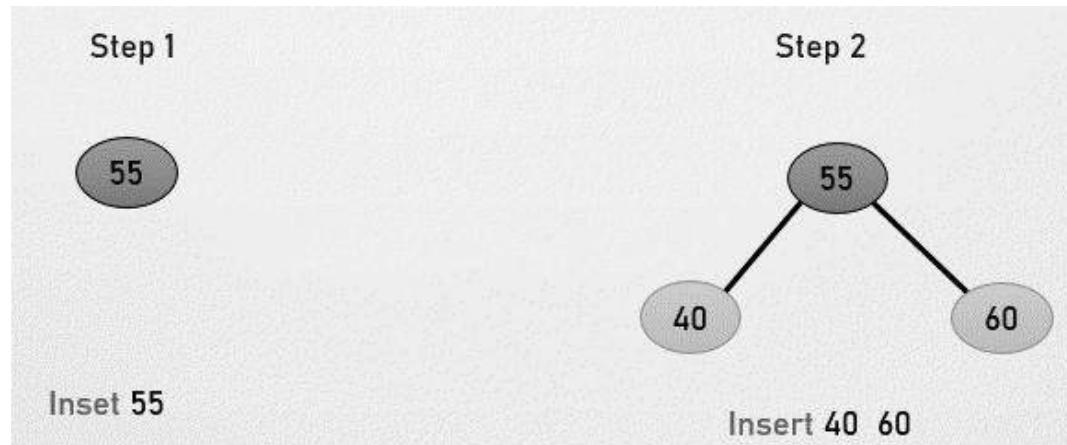
If the parent of newNode is Black then exit from the operation.

If the parent of newNode is Red then check the color of parentnode's sibling of newNode.

If it is colored Black or NULL then make suitable Rotation and Recolor it.

If it is colored Red then perform Recolor. Repeat the same until tree becomes Red Black Tree.

Red-black tree insertion operation



Algorithm to insert a node

Following steps are followed for inserting a new element into a red-black tree:

Let y be the leaf (ie. NIL) and x be the root of the tree.

Check if the tree is empty (ie. whether x is NIL). If yes, insert newNode as a root node and color it black.

Else, repeat steps following steps until leaf (NIL) is reached.

Compare newKey with rootKey.

If newKey is greater than rootKey, traverse through the right subtree.

Else traverse through the left subtree.

Assign the parent of the leaf as a parent of newNode.

If leafKey is greater than newKey, make newNode as rightChild.

Else, make newNode as leftChild.

Assign NULL to the left and rightChild of newNode.

Assign RED color to newNode.

Deletion operation

The deletion operation in Red-Black Tree is similar to the BST. In deletion operation, we need to check with the Red-Black Tree properties. If any of the properties are violated then make suitable operations like Recolor, Rotation and Rotation followed by Recolor to make it Red-Black Tree.

Algorithm to delete a node

Save the color of nodeToBeDeleted in originalColor.

If the left child of nodeToBeDeleted is NULL

Assign the right child of nodeToBeDeleted to x.

Transplant nodeToBeDeleted with x.

Else if the right child of nodeToBeDeleted is NULL

Assign the left child of nodeToBeDeleted into x.

Transplant nodeToBeDeleted with x.

Else

Assign the minimum of right subtree of noteToBeDeleted into y.

Save the color of y in originalColor.

Assign the rightChild of y into x.

If y is a child of nodeToBeDeleted, then set the parent of x as y.

Else, transplant y with rightChild of y.

Transplant nodeToBeDeleted with y.

Set the color of y with originalColor.

If the originalColor is BLACK, call DeleteFix(x).

Red-black tree applications

- To implement Java packages.
- To implement Standard Template Libraries (STL) in C++.
- It is used in K-mean clustering algorithm for reducing time complexity.
- To implement finite maps
- It is used to implement CPU Scheduling in Linux.
- In MySQL Red-Black tree is used for indexes on tables.

Time complexity in big O notation

Sr. No.	Algorithm	Time Complexity
1.	Search	$O(\log n)$
2.	Insert	$O(\log n)$
3.	Delete	$O(\log n)$

**Lab Exercise:**

Implementation of red-black tree

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node *parent;
    Node *left;
    Node *right;
    int color;
};
typedef Node *NodePtr;
class RedBlackTree {
private:
    NodePtr root;
    NodePtr TNULL;
    void initializeNULLNode(NodePtr node, NodePtr parent) {
        node->data = 0;
        node->parent = parent;
        node->left = nullptr;
        node->right = nullptr;
        node->color = 0;
    }
    // Preorder
    void preOrderHelper(NodePtr node) {
        if (node != TNULL) {
            cout << node->data << " ";
            preOrderHelper(node->left);
            preOrderHelper(node->right);
        }
    }
    // Inorder
    void inOrderHelper(NodePtr node) {
        if (node != TNULL) {
            inOrderHelper(node->left);
```

```

cout << node->data << " ";
inOrderHelper(node->right);
}
}

// Post order
void postOrderHelper(NodePtr node) {
if (node != TNULL) {
    postOrderHelper(node->left);
    postOrderHelper(node->right);
    cout << node->data << " ";
}
}

NodePtr searchTreeHelper(NodePtr node, int key) {
if (node == TNULL || key == node->data) {
    return node;
}
if (key < node->data) {
    return searchTreeHelper(node->left, key);
}
return searchTreeHelper(node->right, key);
}

// For balancing the tree after deletion
void deleteFix(NodePtr x) {
NodePtr s;
while (x != root && x->color == 0) {
    if (x == x->parent->left) {
        s = x->parent->right;
        if (s->color == 1) {
            s->color = 0;
            x->parent->color = 1;
            leftRotate(x->parent);
            s = x->parent->right;
        }
        if (s->left->color == 0 && s->right->color == 0) {
            s->color = 1;
            x = x->parent;
        } else {
            if (s->right->color == 0) {
                s->left->color = 0;
                s->color = 1;
            }
        }
    }
}

```

```

        rightRotate(s);
        s = x->parent->right;
    }
    s->color = x->parent->color;
    x->parent->color = 0;
    s->right->color = 0;
    leftRotate(x->parent);
    x = root;
}
} else {
    s = x->parent->left;
    if (s->color == 1) {
        s->color = 0;
        x->parent->color = 1;
        rightRotate(x->parent);
        s = x->parent->left;
    }
    if (s->right->color == 0 && s->right->color == 0) {
        s->color = 1;
        x = x->parent;
    } else {
        if (s->left->color == 0) {
            s->right->color = 0;
            s->color = 1;
            leftRotate(s);
            s = x->parent->left;
        }
        s->color = x->parent->color;
        x->parent->color = 0;
        s->left->color = 0;
        rightRotate(x->parent);
        x = root;
    }
}
}
x->color = 0;
}

void rbTransplant(NodePtr u, NodePtr v) {
    if (u->parent == nullptr) {
        root = v;
    }
}

```

```

} else if (u == u->parent->left) {
    u->parent->left = v;
} else {
    u->parent->right = v;
}
v->parent = u->parent;
}

void deleteNodeHelper(NodePtr node, int key) {
    NodePtr z = TNULL;
    NodePtr x, y;
    while (node != TNULL) {
        if (node->data == key) {
            z = node;
        }
        if (node->data <= key) {
            node = node->right;
        } else {
            node = node->left;
        }
    }
    if (z == TNULL) {
        cout << "Key not found in the tree" << endl;
        return;
    }
    y = z;
    int y_original_color = y->color;
    if (z->left == TNULL) {
        x = z->right;
        rbTransplant(z, z->right);
    } else if (z->right == TNULL) {
        x = z->left;
        rbTransplant(z, z->left);
    } else {
        y = minimum(z->right);
        y_original_color = y->color;
        x = y->right;
        if (y->parent == z) {
            x->parent = y;
        } else {
            rbTransplant(y, y->right);
        }
    }
}

```

```

y->right = z->right;
y->right->parent = y;
}
rbTransplant(z, y);
y->left = z->left;
y->left->parent = y;
y->color = z->color;
}
delete z;
if (y_original_color == 0) {
    deleteFix(x);
}
}

// For balancing the tree after insertion
void insertFix(NodePtr k) {
    NodePtr u;
    while (k->parent->color == 1) {
        if (k->parent == k->parent->parent->right) {
            u = k->parent->parent->left;
            if (u->color == 1) {
                u->color = 0;
                k->parent->color = 0;
                k->parent->parent->color = 1;
                k = k->parent->parent;
            } else {
                if (k == k->parent->left) {
                    k = k->parent;
                    rightRotate(k);
                }
                k->parent->color = 0;
                k->parent->parent->color = 1;
                leftRotate(k->parent->parent);
            }
        } else {
            u = k->parent->parent->right;
            if (u->color == 1) {
                u->color = 0;
                k->parent->color = 0;
                k->parent->parent->color = 1;
            }
        }
    }
}

```

```

k = k->parent->parent;
} else {
    if (k == k->parent->right) {
        k = k->parent;
        leftRotate(k);
    }
    k->parent->color = 0;
    k->parent->parent->color = 1;
    rightRotate(k->parent->parent);
}
}

if (k == root) {
    break;
}
}

root->color = 0;
}

void printHelper(NodePtr root, string indent, bool last) {
if (root != TNULL) {
    cout << indent;
    if (last) {
        cout << "R----";
        indent += "  ";
    } else {
        cout << "L----";
        indent += "| ";
    }
    string sColor = root->color ? "RED" : "BLACK";
    cout << root->data << "(" << sColor << ")" << endl;
    printHelper(root->left, indent, false);
    printHelper(root->right, indent, true);
}
}

public:
RedBlackTree() {
    TNULL = new Node;
    TNULL->color = 0;
    TNULL->left = nullptr;
    TNULL->right = nullptr;
    root = TNULL;
}

```

```
}

void preorder() {
    preOrderHelper(this->root);
}

void inorder() {
    inOrderHelper(this->root);
}

void postorder() {
    postOrderHelper(this->root);
}

NodePtr searchTree(int k) {
    return searchTreeHelper(this->root, k);
}

NodePtr minimum(NodePtr node) {
    while (node->left != TNULL) {
        node = node->left;
    }
    return node;
}

NodePtr maximum(NodePtr node) {
    while (node->right != TNULL) {
        node = node->right;
    }
    return node;
}

NodePtr successor(NodePtr x) {
    if (x->right != TNULL) {
        return minimum(x->right);
    }
    NodePtr y = x->parent;
    while (y != TNULL && x == y->right) {
        x = y;
        y = y->parent;
    }
    return y;
}

NodePtr predecessor(NodePtr x) {
    if (x->left != TNULL) {
        return maximum(x->left);
    }
}
```

```
NodePtr y = x->parent;
while (y != TNULL && x == y->left) {
    x = y;
    y = y->parent;
}
return y;
}

void leftRotate(NodePtr x) {
    NodePtr y = x->right;
    x->right = y->left;
    if (y->left != TNULL) {
        y->left->parent = x;
    }
    y->parent = x->parent;
    if (x->parent == nullptr) {
        this->root = y;
    } else if (x == x->parent->left) {
        x->parent->left = y;
    } else {
        x->parent->right = y;
    }
    y->left = x;
    x->parent = y;
}

void rightRotate(NodePtr x) {
    NodePtr y = x->left;
    x->left = y->right;
    if (y->right != TNULL) {
        y->right->parent = x;
    }
    y->parent = x->parent;
    if (x->parent == nullptr) {
        this->root = y;
    } else if (x == x->parent->right) {
        x->parent->right = y;
    } else {
        x->parent->left = y;
    }
    y->right = x;
    x->parent = y;
```

```
}

// Inserting a node

void insert(int key) {

    NodePtr node = new Node;

    node->parent = nullptr;
    node->data = key;
    node->left = TNULL;
    node->right = TNULL;
    node->color = 1;

    NodePtr y = nullptr;
    NodePtr x = this->root;
    while (x != TNULL) {

        y = x;
        if (node->data < x->data) {
            x = x->left;
        } else {
            x = x->right;
        }
    }

    node->parent = y;
    if (y == nullptr) {
        root = node;
    } else if (node->data < y->data) {
        y->left = node;
    } else {
        y->right = node;
    }

    if (node->parent == nullptr) {
        node->color = 0;
        return;
    }

    if (node->parent->parent == nullptr) {
        return;
    }

    insertFix(node);
}

NodePtr getRoot() {
    return this->root;
}
```

```

void deleteNode(int data) {
    deleteNodeHelper(this->root, data);
}

void printTree() {
    if (root) {
        printHelper(this->root, "", true);
    }
}

int main() {
    RedBlackTree bst;
    bst.insert(55);
    bst.insert(40);
    bst.insert(65);
    bst.insert(60);
    bst.insert(75);
    bst.insert(57);
    bst.printTree();
    cout << endl
    << "After deleting" << endl;
    bst.deleteNode(40);
    bst.printTree();
}

```

7.4 Splay Trees

Splay trees are binary search trees that achieve our goals by being self-adjusting in a quite remarkable way: Every time we access a node of the tree, whether for insertion or retrieval, we perform radical surgery on the tree, lifting the newly accessed node all the way up, so that it becomes the root of the modified tree. Other nodes are pushed out of the way as necessary to make room for this new root. Nodes that are frequently accessed will frequently be lifted up to become the root, and they will never drift too far from the top position. Inactive nodes, on the other hand, will slowly be pushed farther and farther from the root.

It is possible that splay trees can become highly unbalanced, so that a single access to a node of the tree can be quite expensive. Later in this section, however, we shall prove that, over a long sequence of accesses, splay trees are not at all expensive and are guaranteed to require not many more operations even than AVL trees. The analytical tool used is called amortized algorithm analysis, since, like insurance calculations, the few expensive cases are averaged in with many less expensive cases to obtain excellent performance over a long sequence of operations.

Splay Trees were invented by Sleator and Tarjan. This data structure is essentially a binary tree with special update and access rules. It has the property to adapt optimally to a sequence of tree operations. More precisely, a sequence of m operations on a tree with initially n nodes takes time $O(n \ln(n) + m \ln(n))$.

Splaying

Splaying is a process in which a node is transferred to the root by performing suitable rotations. In a splay tree, whenever we access any node (searching, inserting or deleting a node), it is splayed to the root.

7.5 Operations

Splay trees support the following operations. We write S for sets, x for elements and k for key values.

splay(S, k) returns an access to an element x with key k in the set S. In case no such element exists, we return an access to the next smaller or larger element.

split(S, k) returns (S_1,S_2), where for each x in S_1 holds: $\text{key}[x] \leq k$, and for each y in S_2 holds: $k < \text{key}[y]$.

join(S_1,S_2) returns the union $S = S_1 + S_2$. Condition: for each x in S_1 and each y in S_2: $x \leq y$.

insert(S,x) augments S by x.

delete(S,x) removes x from S.

Each split, join, delete and insert operation can be reduced to splay operations and modifications of the tree at the root which take only constant time. Thus, the run time for each operation is essentially the same as for a splay operation.

The most important tree operation is splay(x), which moves an element x to the root of the tree. In case x is not present in the tree, the last element on the search path for x is moved instead. The run time for a splay(x) operation is proportional to the length of the search path for x. While searching for x we traverse the search path top-down. Let y be the last node on that path. In a second step, we move y along that path by applying rotations as described later.

The time complexity of maintaining a splay tree is analyzed using an Amortized Analysis. Consider a sequence of operations op_1, op_2, ..., op_m. Assume that our data structure has a potential. One can think of the potential as a bank account. Each tree operation op_i has actual costs proportional to its running time. We're paying for the costs c_i of op_i with its amortized costs a_i. The difference between concrete and amortized costs is charged against the potential of the data structure. This means that we're investing in the potential if the amortized costs are higher than the actual costs, otherwise we're decreasing the potential.

Thus, we're paying for the sequence op_1, op_2, ..., op_m no more than the initial potential plus the sum of the amortized costs $a_1 + a_2 + \dots + a_m$.

The trick of the analysis is to define a potential function and to show that each splay operation has amortized costs $O(\ln(n))$. It follows that the sequence has costs $O(m \ln(n) + n \ln(n))$

7.6 Rotations in Splay Tree

Zig rotation / Right rotation

Zag rotation / Left rotation

Zig zag / Zig followed by zag

Zag zig / Zag followed by zig

Zig zig / two right rotations

Zag zag / two left rotations

Factors for selecting a type of rotation

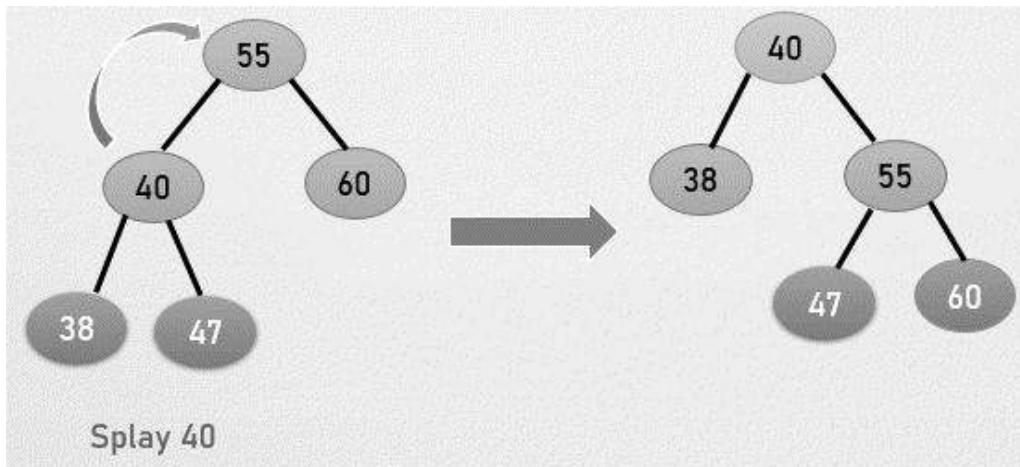
Does the node which we are trying to rotate have a grandparent?

Is the node left or right child of the parent?

Is the node left or right child of the grandparent?

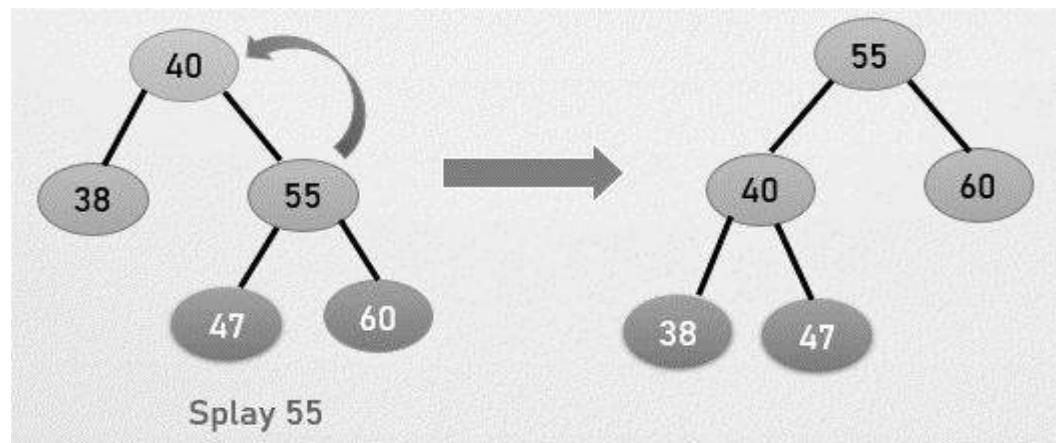
Zig Rotation

The Zig Rotation in splay tree is like single right rotation in AVL Tree rotations. In zig rotation, every node moves one position to the right from its current position.



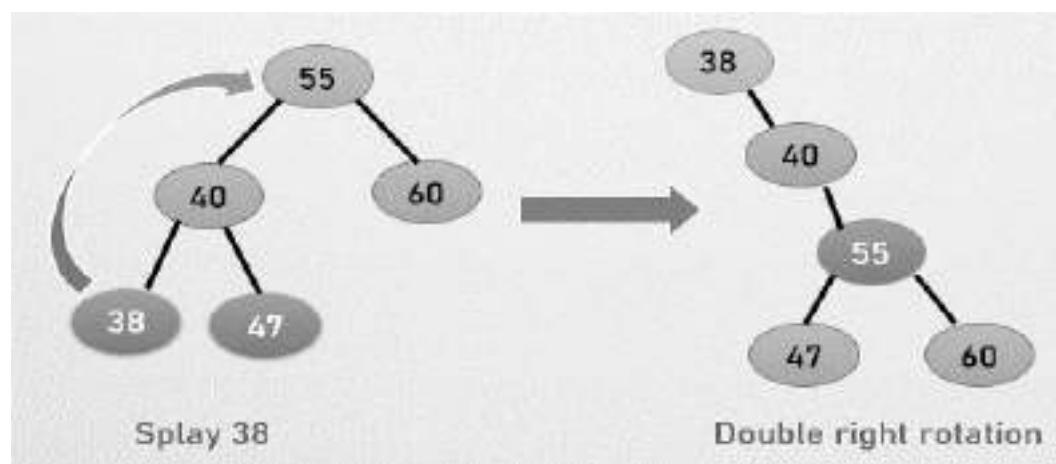
Zag Rotation

In zag rotation, every node moves one position to the left from its current position.

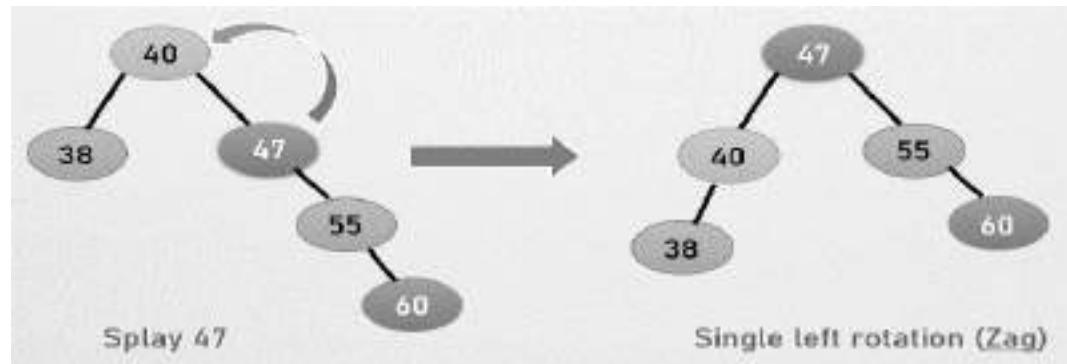


Zig-Zig Rotation

The Zig-Zig Rotation in splay tree is a double zig rotation. In zig-zig rotation, every node moves two positions to the right from its current position.



Zig-Zag Rotation



Advantages of Splay Trees

Splaying ensures that frequently accessed elements stay near the root of the tree so that they are easily accessible.

The average case performance of splay trees is comparable to other fully-balanced trees: $O(\log n)$. Splay trees do not need bookkeeping data; therefore, they have a small memory footprint.

Disadvantages

A splay tree can arrange itself linearly. Therefore, the worst-case performance of a splay tree is $O(n)$.

Multithreaded operations can be complicated since, even in a read-only configuration, splay trees can reorganize themselves.

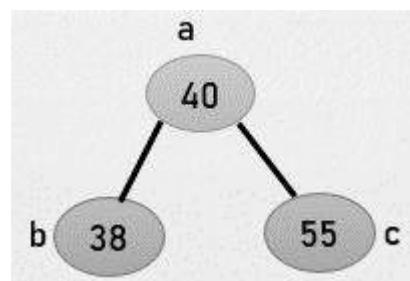
7.7 2-3 Trees

A 2-3 tree is another type of tree which has 2 types of nodes, 2-node and 3-node. A 2-3 tree data structure is a specific form of a B tree where every node with children has either two children and one data element or three children and two data elements. It is a self-balancing tree. It is balanced with every leaf node at equal distance from the root node.

2-Node

2-Node: A node with a single data element that has two child nodes.

1. Every value appearing in the child (b) 38 must be \leq (a) 40.
2. Every value appearing in the child (c) 55 must be \geq (a) 40.
3. The length of the path from the root of a 2-node to every leaf in its child must be the same.

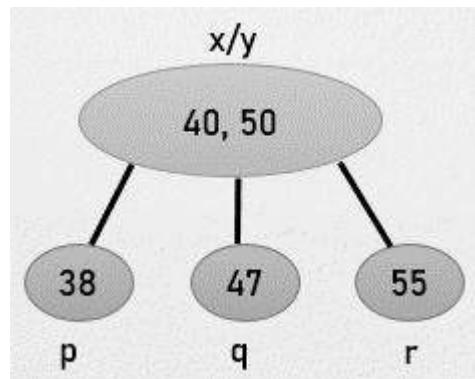


3-Node

3-Node: A node with two data elements that has three child nodes.

1. Every value appearing in child P must be \leq X.
2. Every value appearing in child Q must be in between X and Y.
3. Every value appearing in child R must be \geq Y.

4. The length of the path from the root of a 3-node to every leaf in its child must be the same.



7.8 Properties of 2-3 Trees

- Every internal node in the tree is a 2-node or a 3-node i.e it has either one value or two values.
- A node with one value is either a leaf node or has exactly two children. Values in left sub tree < value in node < values in right sub tree.
- Data stored in sorted manner.
- Insertion operation performed in leaf node.
- A node with two values is either a leaf node or has exactly 3 children. It cannot have 2 children.
- Values in left sub tree < first value in node < values in middle sub tree < second value in node < value in right sub tree.
- All leaf nodes are at the same level.

7.9 2-3 Tree Operations

Insertion

Deletion

Search

Insertion operation

If the tree is empty, create a node and put value into the node

Otherwise find the leaf node where the value belongs.

If the leaf node has only one value, put the new value into the node

If the leaf node has more than two values, split the node and promote the median of the three values to parent.

If the parent then has three values, continue to split and promote, forming a new root node if necessary

Search operation

If Tree is empty, return False (data item cannot be found in the tree).

If current node contains data value which is equal to data, return True.

If we reach the leaf-node and it doesn't contain the required key value, return False.

Recursive Calls

If data < currentNode.leftVal, we explore the left sub tree of the current node.

Else if `currentNode.leftVal < data < currentNode.rightVal`, we explore the middle sub tree of the current node.

Else if `data > currentNode.rightVal`, we explore the right sub tree of the current node.

Deletion process

There are three cases in deletion process

1. When the record is to be removed from a leaf node containing two records.

In this case, the record is simply removed, and no other nodes are affected.

2. When the only record in a leaf node is to be removed.

3. When a record is to be removed from an internal node.

In both the second and the third cases, the deleted record is replaced with another that can take its place while maintaining the correct order of 2-3 tree.

Summary

- A Red Black Tree is a self-balancing binary search tree in which each node has a red or black colour.
- The red black tree satisfies all of the features of the binary search tree, but it also has several additional properties.
- Splay trees are self-adjusting binary search trees in which every access for insertion or retrieval of a node, lifts that node all the way up to become the root, pushing the other nodes out of the way to make room for this new root of the modified tree. Hence, the frequently accessed nodes will frequently be lifted up and remain around the root position; while the most infrequently accessed nodes would move farther and farther away from the root.
- A 2-3 tree data structure is a specific form of a B tree where every node with children has either two children and one data element or three children and two data elements.

Keywords

AVL tree

2-3 tree

Zag rotation / Left rotation

Zig zag / Zig followed by zag

Zag zig / Zag followed by zig

Zig zig / two right rotations

Zag zag / two left rotations

2-Node, 3-Node

Self Assessment

1. Red Black Tree invented in

- A. 1960
- B. 1972
- C. 1976
- D. None of above

2. The height of a Red-Black tree is

- A. O(1)
- B. O(log n)
- C. O(0)
- D. None of above

3. The color of the root in red black tree is
 - A. Red
 - B. Black
 - C. Green
 - D. All of above

4. Red-Black Tree must be
 - A. AVL tree
 - B. BST
 - C. Binary tree
 - D. All of above

5. What is color of newly inserted node in red-black tree
 - A. Red
 - B. Black
 - C. Blue
 - D. None of above

6. Recolor and Rotation performed in
 - A. Insertion
 - B. Deletion
 - C. Both insertion and deletion
 - D. Search

7. 90-10 rule is part of
 - A. BST
 - B. Binary tree
 - C. Splay tree
 - D. All of above

8. Left rotation is also called
 - A. Zig rotation
 - B. Zag rotation
 - C. Zag-zag rotation
 - D. None of above

9. Two right rotations is equal to
 - A. Zag zag
 - B. Zag zig
 - C. Zig zig
 - D. All of above

10. What are factors for selecting a type of rotation

- A. Does the node which we are trying to rotate have a grandparent?
- B. Is the node left or right child of the parent?
- C. Is the node left or right child of the grandparent?
- D. All of above

11. What are the operations performed on Splay Tree

- A. Insertion
- B. Deletion
- C. Search
- D. All of above

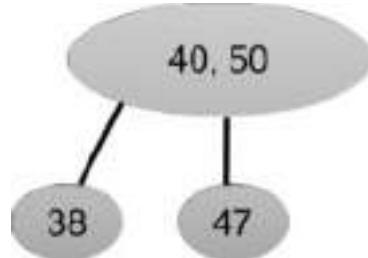
12. 2-node is

- A. A node with a double data element that has two child nodes.
- B. A node with a single data element that has two child nodes.
- C. A node with a single data element that has one child nodes.
- D. All of above

13. 3-node is

- A. A node with two data elements that has three child nodes
- B. A node with three data elements that has three child nodes
- C. A node with two data elements that has two child nodes
- D. None of above

14. Is diagram represent correct 3-node tree?



- A. True
- B. False

15. Properties of 2-3 Trees are

- A. Data stored in sorted manner
- B. Every internal node in the tree is a 2-node or a 3-node
- C. Insertion operation performed in leaf node
- D. All of above

Answers for Self Assessment

1. B 2. B 3. B 4. C 5. A

- | | | | | |
|-------|-------|-------|-------|-------|
| 6. C | 7. C | 8. B | 9. C | 10. D |
| 11. D | 12. B | 13. A | 14. B | 15. D |

Review Questions

1. Discuss red black tree properties.
2. Define recolor and rotation process.
3. Differentiate between zig-zag rotation.
4. Discuss concept of 2-node and 3-node with suitable diagram.
5. Explain splay operation in splay trees.
6. "The time complexity of maintaining a splay tree is analyzed using an Amortized Analysis." Explain
7. "A splay tree does not keep track of heights and does not use any balance factors like anAVL tree". Explain



Further Readings

- Burkhard Monien, Data Structures and Efficient Algorithms, Thomas Ottmann, Springer.
- Kruse, Data Structure & Program Design, Prentice Hall of India, New Delhi.
- Mark Allen Weis, Data Structure & Algorithm Analysis in C, Second Ed., Addison-Wesley Publishing.
- RG Dromey, How to Solve it by Computer, Cambridge University Press.
- Lipschutz. S. (2011). Data Structures with C. Delhi: Tata McGraw hill
- Reddy. P. (1999). Data Structures Using C. Bangalore: Sri Nandi Publications
- Samantha. D (2009). Classic Data Structures. New Delhi: PHI Learning Private Limited



Web Links

www.en.wikipedia.org

www.web-source.net

www.webopedia.com

https://www.cs.auckland.ac.nz/software/AlgAnim/red_black.html

<https://www.javatpoint.com/daa-red-black-tree>

<http://www.btechsmhttp://www.cs.cornell.edu/courses/cs3110/2011sp/Recitations/rec25-splay/splay.htm>

Unit 08: Heaps

CONTENTS

- Objectives
- Introduction
- 8.1 Heap
- 8.2 Heapify Method (Min Heap)
- 8.3 Deletion Operation on Heap Tree
- 8.4 Applications of Heaps
- 8.5 Priority Queue Operations
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to:

- Understand basics of heap
- Learn max and min heap
- Discuss operations on heap
- Learn priority queue

Introduction

The heap data structure is a complete binary tree where each node of the tree has an orderly relationship with its successors. Binary search trees are totally ordered, but the heap data structure is only partially ordered. It is suitable for inserting and deleting minimum value operations.

Heap is an array object that is considered as a complete binary tree. Each node of the tree corresponds to an element of the array that stores the value in the node. The tree is completely filled at all levels except possibly the lowest, which is filled from the left upwards to a point. Heap data structures are suitable for implementing priority queues. The heap serves as a foundation of a theoretically important sorting algorithm called heap sort, which we will discuss after defining the heap.

8.1 Heap

A heap is a specialized tree-based data structure that satisfies the heap property: if B is a child node of A, then $\text{key}(A) \geq \text{key}(B)$. This implies that an element with the greatest key is always in the root node, and so such a heap is sometimes called a max-heap. (Alternatively, if the comparison is reversed, the smallest element is always in the root node, which results in a min-heap.) The heap is one maximally-efficient implementation of an abstract data type called a priority queue. Heaps are crucial in several efficient graph algorithms.

A heap is a storage pool in which regions of memory are dynamically allocated. For example, in C++ the space for a variable is allocated essentially in one of three possible places: Global variables are allocated in the space of initialized static variables; the local variables of a procedure are allocated in the procedure's activation record, which is typically found in the processor stack; and dynamically allocated variables are allocated in the heap. In this unit, the term heap is taken to mean the storage pool for dynamically allocated variables.

I consider heaps and heap-ordered trees in the context of priority queue implementations. While it may be possible to use a heap to manage a dynamic storage pool, typical implementations do not. In this context, the technical meaning of the term heap is closer to its dictionary definition—"a pile of many things."

A binary tree has the heap property iff

1. It is empty or
2. The key in the root is larger than that in either child and both subtrees have the heap property.

A heap can be used as a priority queue: the highest priority item is at the root and is trivially extracted. But if the root is deleted, you are left with two sub-trees and you must efficiently re-create a single tree with the heap property.

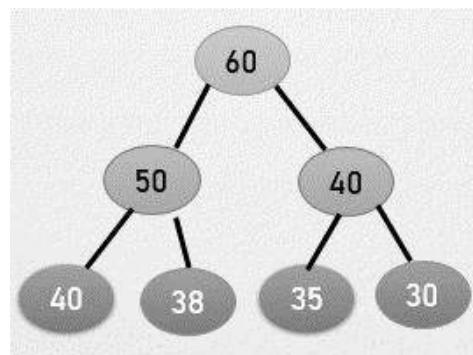
The value of the heap structure is that you can both extract the highest priority item and insert a new one in $O(\log n)$ time.

A heap can be defined as binary trees with keys assigned to its nodes (one key per node). The two types of heaps are:

1. Max heaps
2. Min heaps

Max heaps

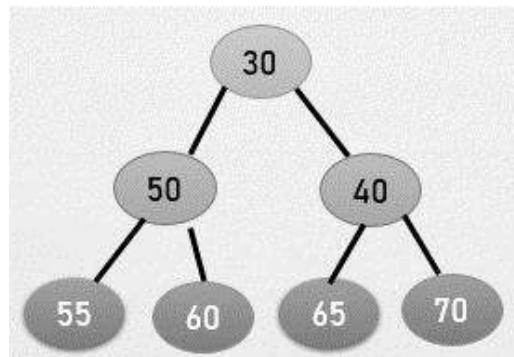
The key present at the root node must be greatest or equal to the keys present at all of its children. The same property must be true for all sub-trees in that Binary Tree.



Max heap

Min heaps

The key present at the root node must be minimum or equal to the keys present at all of its children. The same property must be true for all sub-trees in that Binary Tree.



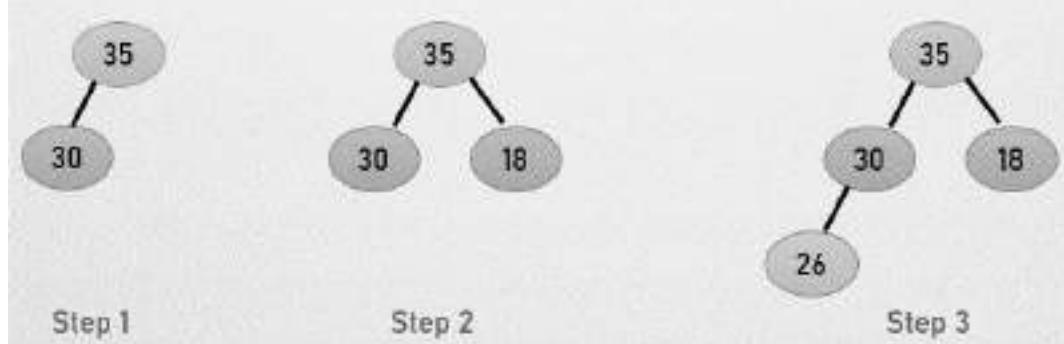
Heap Tree construction

1. Create a new node at the end of heap.
2. Assign new value to the node.
3. Compare the value of this child node with its parent.
4. If value of parent is less than child, then swap them.
5. Repeat step 3 & 4 until Heap property holds.

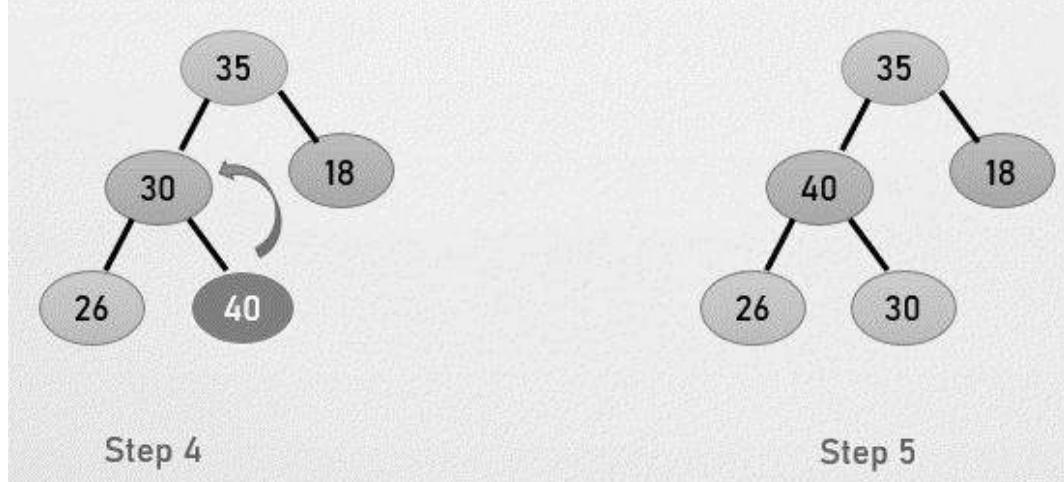


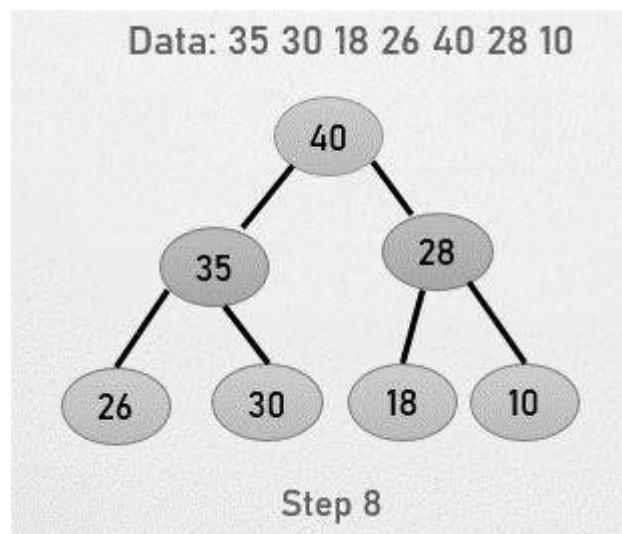
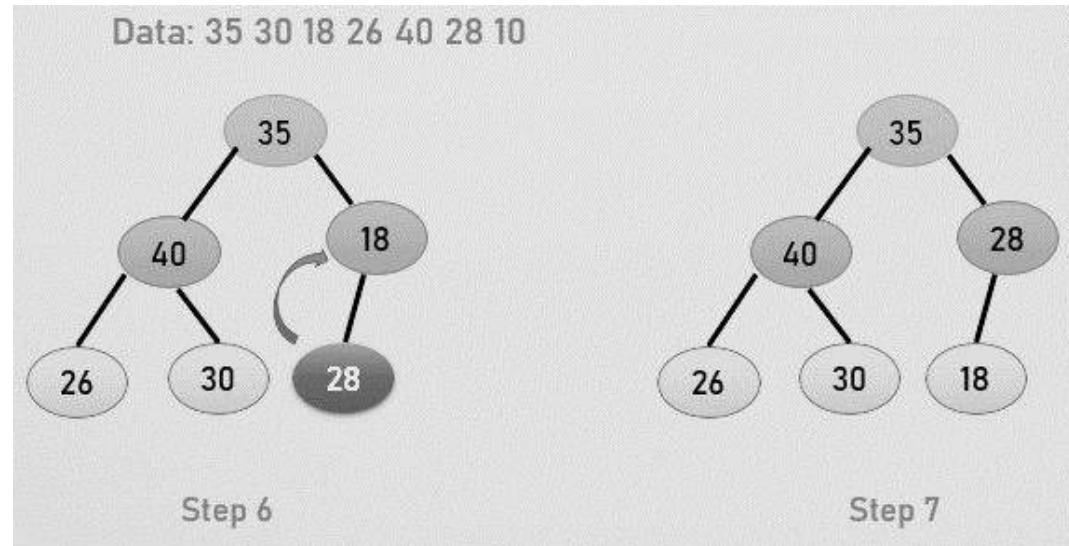
Example:Max heap

Data: 35 30 18 26 40 28 10



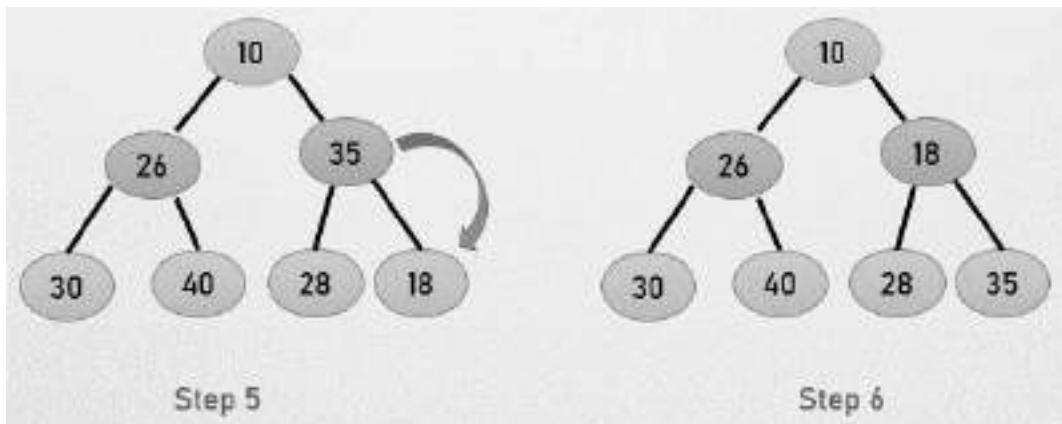
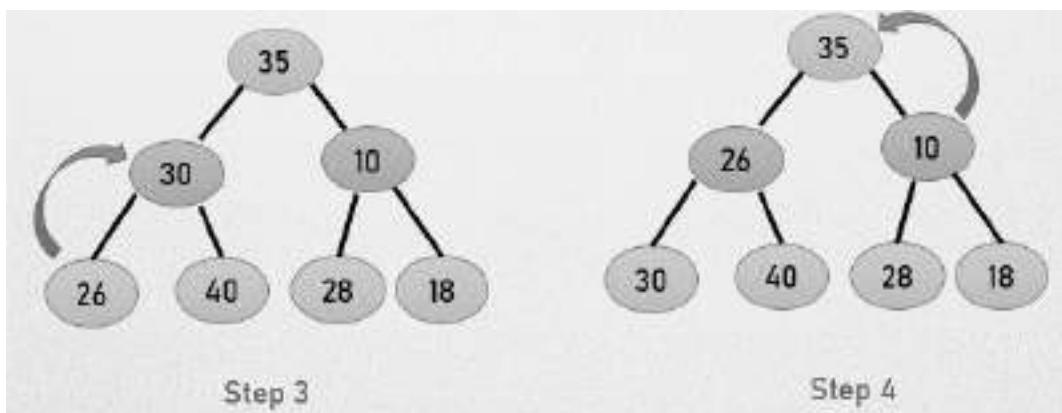
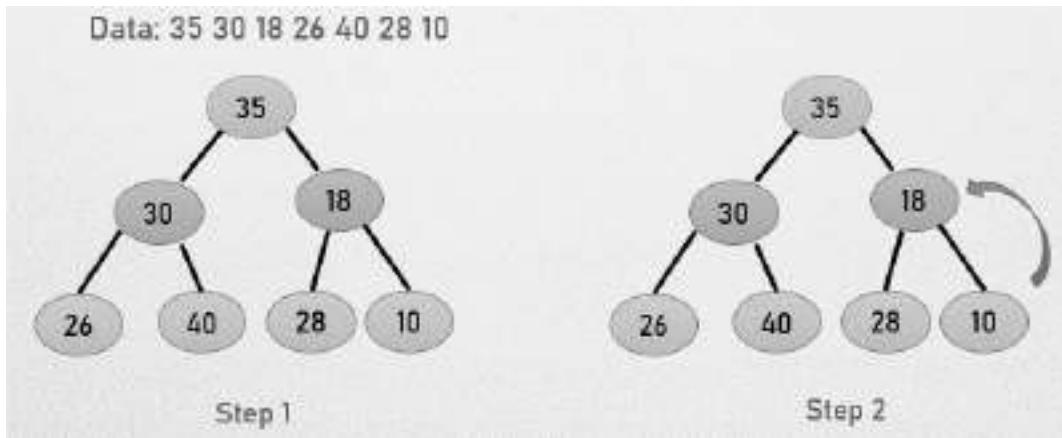
Data: 35 30 18 26 40 28 10





8.2 Heapify Method (Min Heap)

Complexity: $O(n)$



8.3 Deletion Operation on Heap Tree

There are two methods for deletion in heap tree.

Method 1

1. Remove root node.
2. Move the last element of last level to root.
3. Compare the value of this child node with its parent.

4. If value of parent is less than child, then swap them.
5. Repeat step 3 & 4 until Heap property holds.

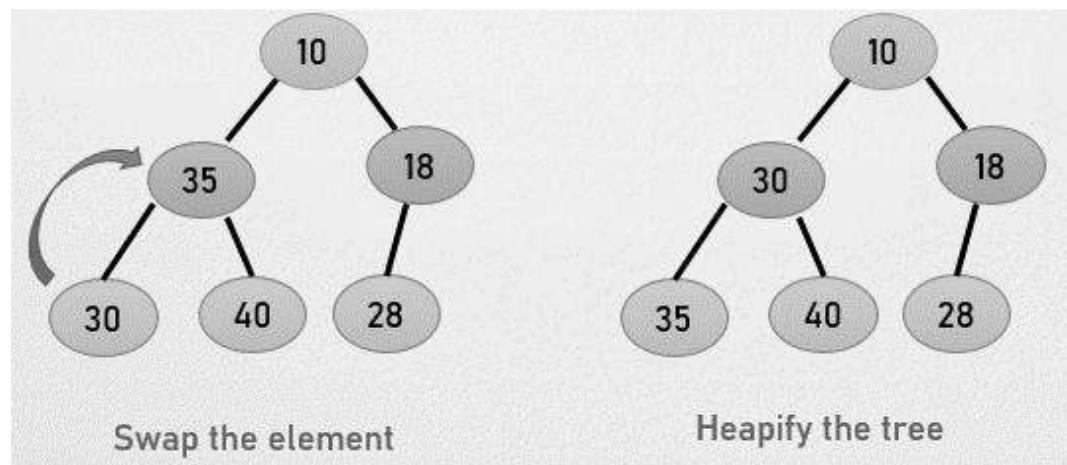
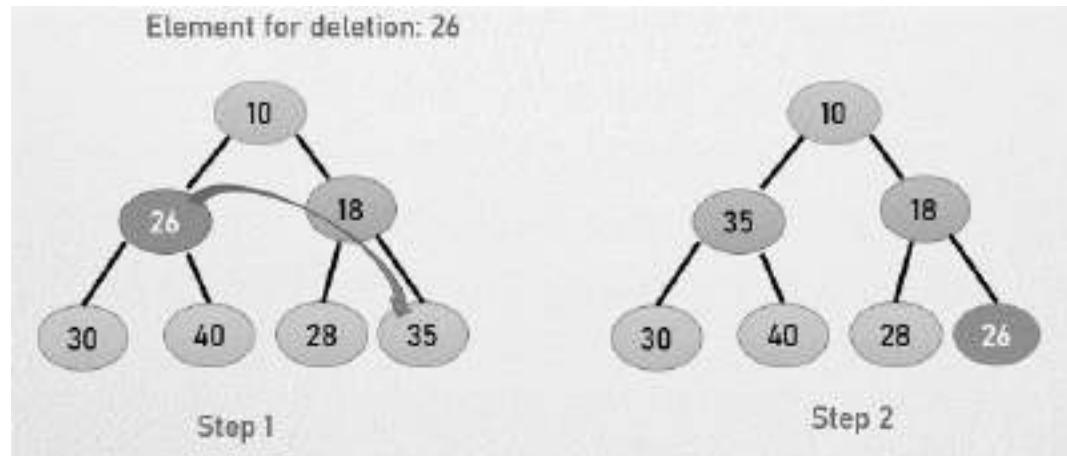
Method 2

1. Select the element to be deleted.
2. Swap it with the last element.
3. Remove the last element.
4. Heapify the tree.



Example:

Deletion operation on heap



8.4 Applications of Heaps

Priority queue implementation

Heap sort

Order statistics

Priority queue

In priority queue key is associated with every element. The element with highest priority will be moved to the front of the queue and one with lowest priority will move to the back of the queue. Queue returns the element according to priority. However, if elements with the same priority occur, they are served according to their order in the queue.

One of the most important applications of priority queues is in discrete event simulation. Simulation is a tool which is used to study the behavior of complex systems. The first step in simulation is modeling. You construct a mathematical model of the system I wish to study. Then you write a computer program to evaluate the model.

The systems studied using discrete event simulation have the following characteristics: The system has a state which evolves or changes with time. Changes in state occur at distinct points in simulation time. A state change moves the system from one state to another instantaneously. State changes are called events.

A priority queue is a queue with items having an orderable characteristic called priority. The objects having the highest priority are always removed first from the priority queues. A priority queue can be obtained by creating a heap. First call a function that creates an ascending heap.

After creating the heap, delete the root node and call a function to recreate the heap for the remaining elements. This method helps in implementing an ascending priority queue. In the same way, we can implement a descending priority queue.

A **max-priority** queue returns the element with maximum key first. A max-heap is used for a max-priority queue.

A **min-priority** queue returns the element with the smallest key first. A min-heap is used for a min-priority queue.

Ascending order priority queue: In ascending order priority queue, a lower priority number is given as a higher priority in a priority.

Descending order priority queue: In descending order priority queue, a higher priority number is given as a higher priority in a priority.

8.5 Priority Queue Operations

Insert

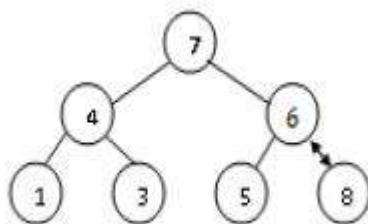
Delete

Peeking from the Priority Queue (Find max/min)

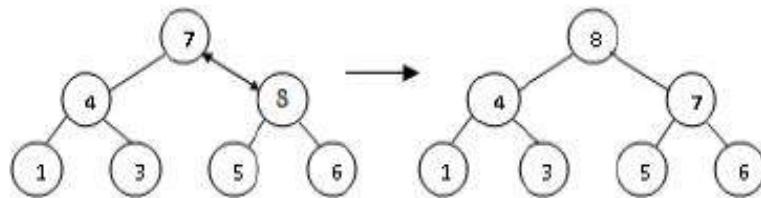
Extract-Max/Min from the Priority Queue

Insertion operation

To insert a new key into a heap, add a new node with key K after the last leaf of the existing heap. Then, shift K up to its suitable place in the new heap. Consider inserting value 8 into the heap shown in the figure



Compare 8 with its parent key. Stop if the parent key is greater than or equal to 8. Else, swap these two keys and compare 8 with its new parent (Refer to figure 14.8). This swapping continues until 8 is not greater than its last parent or it reaches the root. In this algorithm too, we can shift up an empty node until it reaches its proper position, where it acquires the value 8.



This insertion operation does not require more key comparisons than the heap's height. Since the height of a heap with n nodes is about $\log_2 n$, the time efficiency of insertion is in $O(\log n)$.

Insertion operation: algorithm

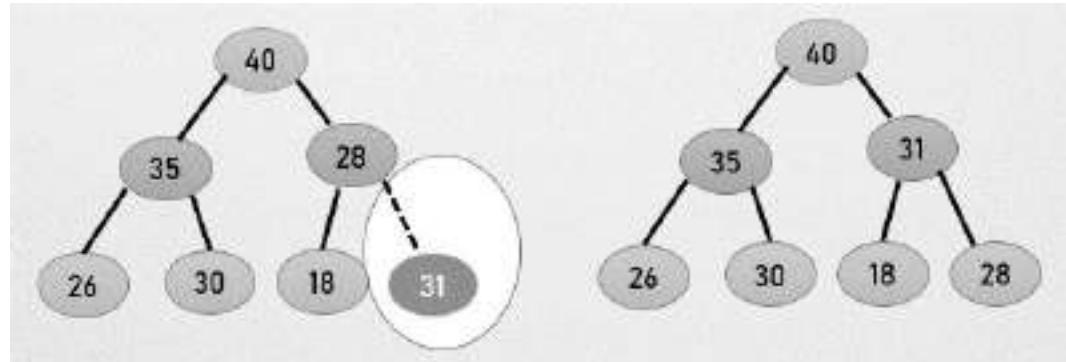
If there is no node,

create a newNode.

else (a node is already present)

insert the newNode at the end (last node from left to right.)

Heapify the array



Delete operation

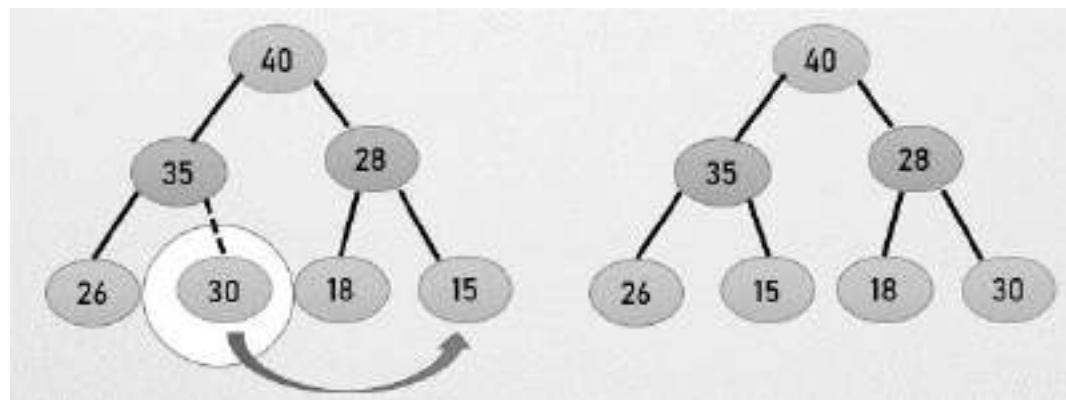
If nodeToBeDeleted is the leafNode

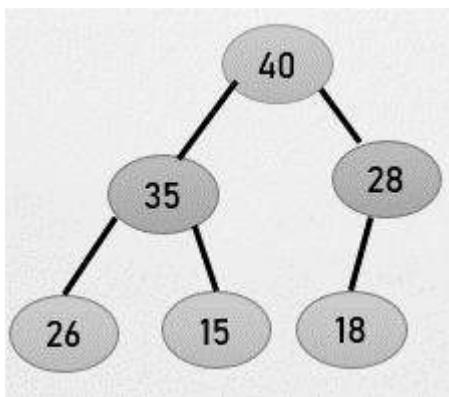
remove the node

Else swap nodeToBeDeleted with the lastLeafNode

remove noteToBeDeleted

heapify the array



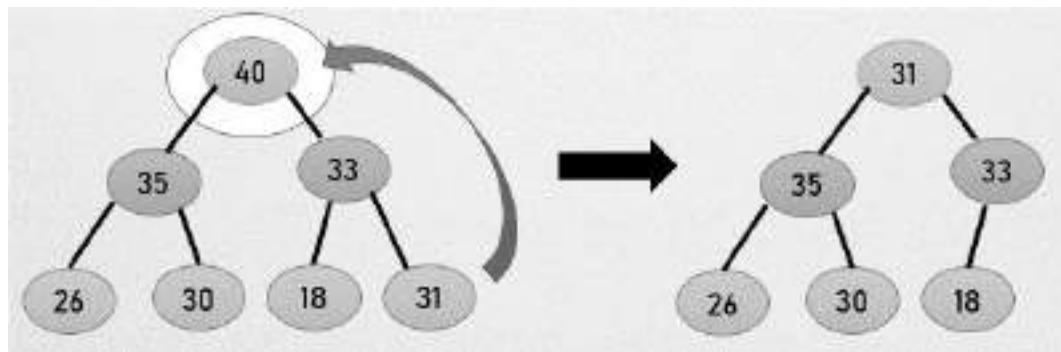


Delete operation

Deleting the Root of a Heap/ Extract Maximum

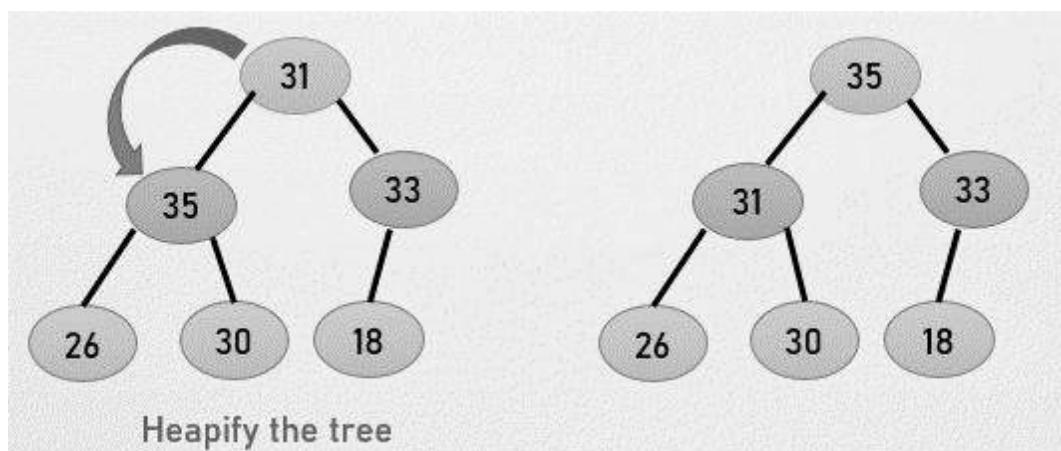
The following steps show the method to delete the root key from a heap in the figure

Step 1: Exchange the root's key with the last key K of the heaps as shown in the figure



Step 2: Decrease the heap's size by 1

Step 3: "Heapify" the smaller tree by shifting K down the tree as we did in the bottom-up heap construction algorithm. That is, verify the parental dominance for K and if it holds, we complete the process. If not, swap K with the largest of its children and repeat this operation until the parental dominance condition holds for K in its new position.



We can determine the efficiency of deletion by the number of key comparisons required to "heapify" the tree after the swap is done, and the size of the tree is decreased by 1. Since it does not need more key comparisons than twice the heap's height, the time efficiency of deletion is in $O(\log n)$.

Applications of Priority Queue

Dijkstra's algorithm for shortest path.

Load balancing and interrupt handling in an operating system.

Data compression in Huffman code.

Summary

- A heap is a partially sorted binary tree. Although a heap is not completely in order, it conforms to a sorting principle: every node has a value less (for the sake of simplicity, I will assume that all orderings are from least to greatest) than either of its children.
- The heap data structure is a complete binary tree where each node of the tree relates to an element of the array that stores the value in the node.
- The two principal ways to construct a heap are by using the bottom-up heap construction algorithm and the top-down heap construction algorithm
- A heap is used to implement heapsort. Heapsort is a comparison-based sorting algorithm which has a worst-case of $O(n \log n)$ runtime.
- A priority queue is a queue with items having an orderable characteristic called priority. The objects having the highest priority are always removed first from the priority queues.
- Priority queue can be attained by creating a heap.

Keywords

- ***Ascending Heap:*** It is a complete binary tree in which the value of each node is greater than or equal to the value of its parent.
- ***Heapify:*** Heapify is a procedure for manipulating heap data structures.
- ***N-ary Tree:*** An n-ary tree is either an empty tree, or a non-empty set of nodes which consists of a root and exactly N sub-trees. The degree of each node of an N-ary tree is either zero or N.
- ***Heap:*** A heap is a specialized tree-based data structure that satisfies the heap property: if B is a child node of A, then $\text{key}(A) \geq \text{key}(B)$.
- ***Binary Heap:*** A binary heap is a heap-ordered binary tree which has a very special shape called a complete tree.
- ***Discrete Event Simulation:*** One of the most important applications of priority queues is in discrete event simulation.

Self Assessment

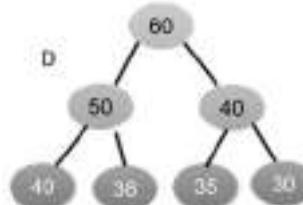
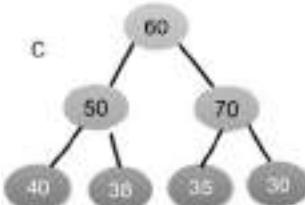
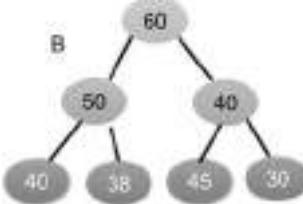
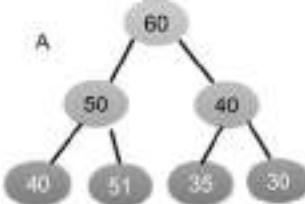
1. Heap satisfy following properties
 - A. Structural property
 - B. Ordering property
 - C. Both Structural and Ordering property
 - D. None of above

2. What are the types of Heap
 - A. Max-Heap
 - B. Min-Heap

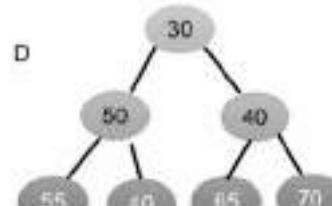
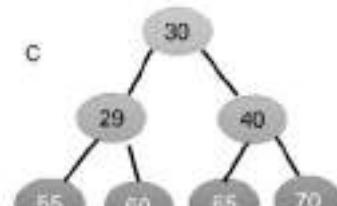
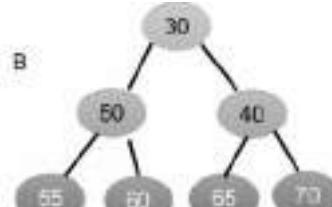
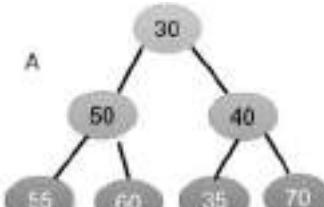
C. Both Max-Heap and Min-Heap

D. None of above

3. Which is correct option for Max-Heap?



4. Which is correct option for Min-Heap?



5. In which heap the root node must be greatest among the keys present at all of its children?

A. Max-heap

B. Min-heap

C. Both A and B

D. None of the above

6. Heap can be used to perform____

A. A decreasing order array

B. Normal Array

C. Priority queue

D. Stack

7. What is the complexity of adding an element to the heap?
 - A. $O(\log n)$
 - B. $O(\log h)$
 - C. Both $O(\log n)$ and $O(\log h)$
 - D. None of above

8. In the worst case, the time complexity of inserting a node in a heap would be
 - A. $O(\log N)$
 - B. $O(1)$
 - C. $O(H)$
 - D. None of above

9. Applications of heap are_____
 - A. Priority queue implementation
 - B. Heap sort
 - C. Order statistics
 - D. All of above

10. Priority queue types are_____
 - A. Max
 - B. Min
 - C. Descending order
 - D. All of above

11. Which is not Priority queue operation?
 - A. Delete
 - B. Peeking from the Priority Queue
 - C. Isfull
 - D. Extract-Max/Min from the Priority Queue

12. What are the methods used to implement Priority Queue?
 - A. Arrays,
 - B. Linked list,
 - C. Heap data structure and binary search tree
 - D. All of above

13. Which is most efficient way to implementing the priority queue?
 - A. Arrays,
 - B. Linked list,
 - C. Heap data structure
 - D. Binary search tree

14. What are the applications of Priority Queue

- A. Dijkstra's algorithm for shortest path
- B. It is used in prim's algorithm
- C. It is used in heap sort
- D. All of above

15. What is the time complexity to insert a node based on key in a priority queue?

- A. $O(n \log n)$
- B. $O(n)$
- C. $O(1)$
- D. $O(n^2)$

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. C | 2. C | 3. D | 4. B | 5. A |
| 6. C | 7. C | 8. A | 9. D | 10. D |
| 11. C | 12. D | 13. D | 14. D | 15. B |

Review Questions

1. Discuss heap properties.
2. "A heap can be implemented as an array by recording its elements in top-down left-to-right manner". Describe in detail.
3. "Binary search property is different from heap property". Justify.
4. Describe priority heap with example.
5. What are the applications of Priority Queue?
6. Represent the max heap and min heap for the data 3, 8, 20, 28, 42, 54.
7. Differentiate between max heap and min heap with example.



Further Readings

- Burkhard Monien, Data Structures and Efficient Algorithms, Thomas Ottmann, Springer.
- Kruse, Data Structure & Program Design, Prentice Hall of India, New Delhi.
- Mark Allen Weis, Data Structure & Algorithm Analysis in C, Second Ed., Addison-Wesley Publishing.
- RG Dromey, How to Solve it by Computer, Cambridge University Press.
- Lipschutz. S. (2011). Data Structures with C. Delhi: Tata McGraw hill
- Reddy. P. (1999). Data Structures Using C. Bangalore: Sri Nandi Publications
- Samantha. D (2009). Classic Data Structures. New Delhi: PHI Learning Private Limited



Web Links

www.en.wikipedia.org

www.web-source.net

www.webopedia.com

<https://www.programiz.com/dsa/heap-data-structure>

<https://www.javatpoint.com/heap-data-structure>

https://www.tutorialspoint.com/data_structures_algorithms/heap_data_structure.htm

Unit 09: More on Heaps

CONTENTS

- Objectives
- Introduction
- 9.1 Heap sort
- 9.2 Complexity of the Heap Sort
- 9.3 Heap Sort Applications
- 9.4 Advantages of Heap Sort
- 9.5 Binomial Heap
- 9.6 Operations of Binomial Heap
- 9.7 Fibonacci Heap
- 9.8 Operations on a Fibonacci Heap
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to:

- Understand basics of heap sort
- Learn binomial heaps
- Discuss fibonacci heaps

Introduction

A heap is a complete binary tree, and a binary tree is one in which each node can have no more than two children. A complete binary tree is one in which all levels except the last, i.e., the leaf node, are completely filled and all nodes are justified to the left.

The elements of a heap sort are processed by generating a min-heap or max-heap with the items of the provided array. The ordering of an array in which the root element reflects the array's minimal or maximum element is known as min-heap or max-heap. Heapsort is a well-liked and efficient sorting method. The idea behind heap sort is to remove elements from the heap part of the list one by one and then insert them into the sorted part.

9.1 Heap Sort

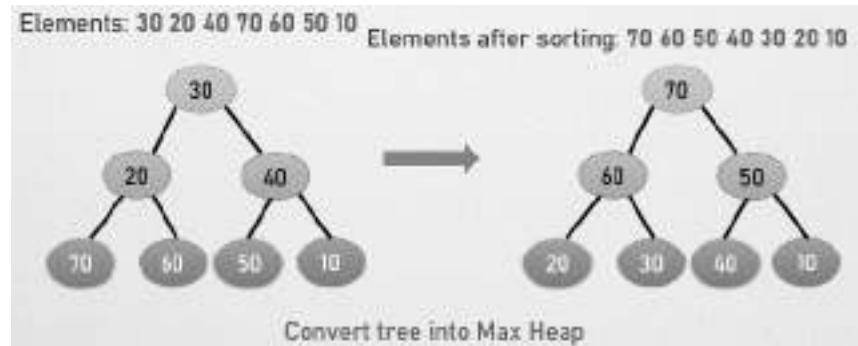
Heap sort is a sorting technique based upon Binary Heap data structure. It is a comparison-based sorting technique. It processes the elements by creating the min heap or max heap using the elements of the array. Heap sort basically recursively performs two main operations -

Build a heap H, using the elements of array.

Repeatedly delete the root element of the heap formed in 1st phase.

Steps for Heap Sort

- Construct a Binary Tree from the list of Elements.
- Transform the Binary Tree into Max Heap / Min Heap.
- Delete the root element from Max Heap / Min Heap
- Reducing the size of heap by 1
- Heapify the root of the tree.
- Put the deleted element into the Sorted list.
- Repeat the same until Min Heap becomes empty.



Algorithm

```

HeapSort(arr)
BuildMaxHeap(arr)
for i = length(arr) to 2
    swap arr[1] with arr[i]
    heap_size[arr] = heap_size[arr] - 1
    MaxHeapify(arr,1)
End

```

BuildMaxHeap(arr)

```

BuildMaxHeap(arr)
heap_size(arr) = length(arr)
for i = length(arr)/2 to 1
    MaxHeapify(arr,i)
End

```

MaxHeapify(arr,i)

```

MaxHeapify(arr,i)
L = left(i)
R = right(i)
if L < heap_size[arr] and arr[L] > arr[i]
    largest = L
else
    largest = i
if R < heap_size[arr] and arr[R] > arr[largest]
    largest = R

```

```

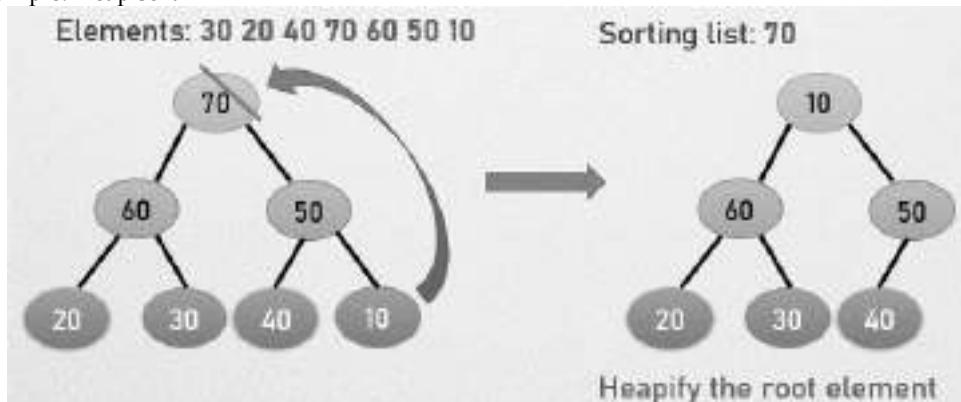
if largest != i
swap arr[i] with arr[largest]
MaxHeapify(arr,largest)
End

```

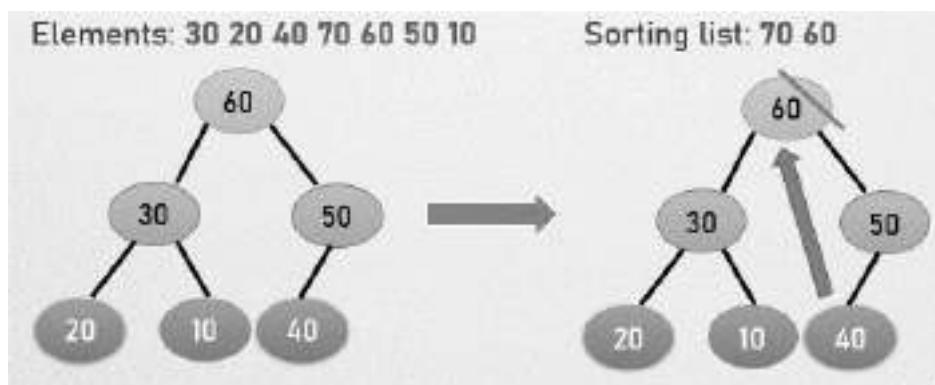
Heap sort first converts the initial array into a heap. The heapsort algorithm uses 'heapify' method to complete the task. The heapify algorithm, as given in the above code, receives a binary tree as input and converts it to a heap. Then, the root is compared with its two children, and the larger child is swapped with it. This may result in one of the left or right sub-trees losing the heap property. As a result, the heapify algorithm is recursively applied to the suitable sub-tree rooted at the node whose value was swapped with the root. This process continues until a leaf node is reached, or until the heap property is satisfied in the sub-tree.



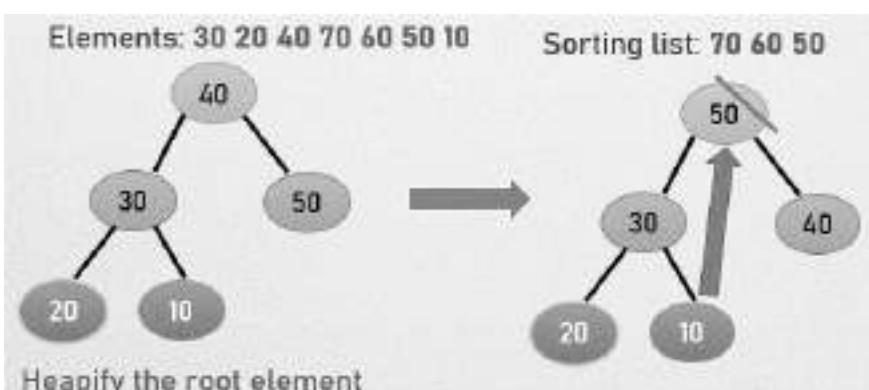
Example: Heap sort

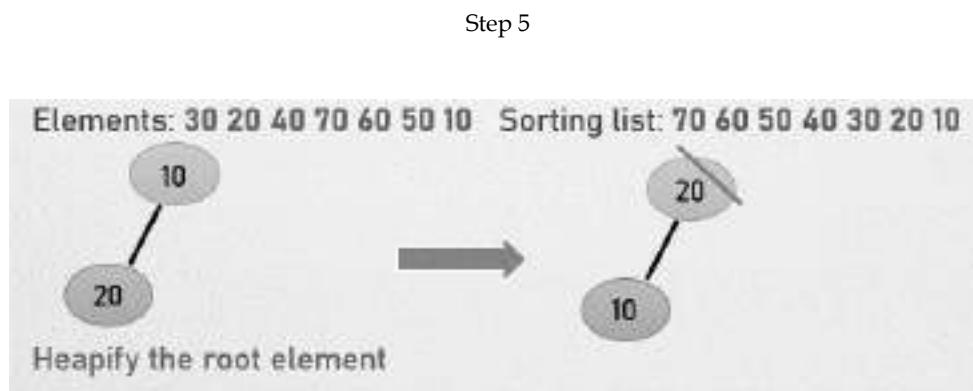
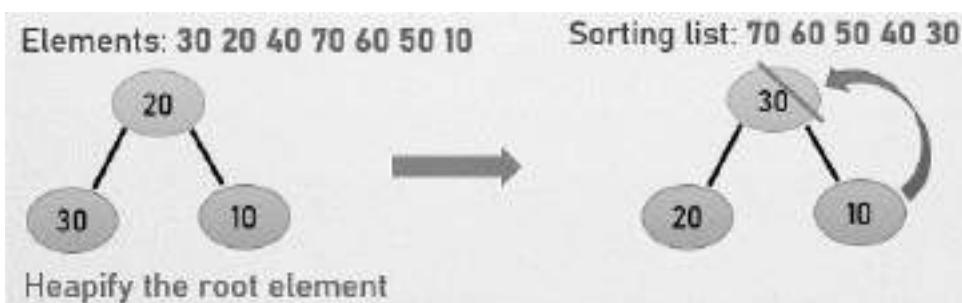
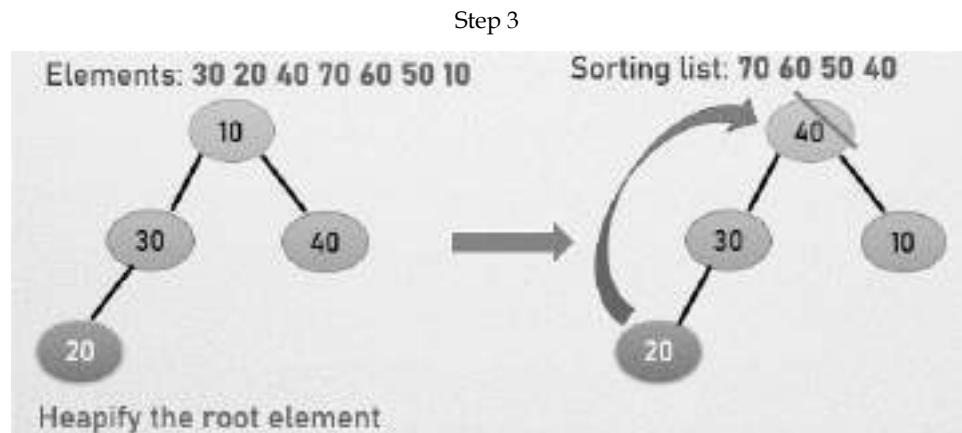


Step 1



Step 2





Step 6

**Lab Exercise:** Heap sort implementation

```
#include <iostream>
using namespace std;
void heapify(int a[], int n, int i)
{
    int largest = i; // Initialize largest as root
    int left = 2 * i + 1; // left child
    int right = 2 * i + 2; // right child
    // If left child is larger than root
    if (left < n && a[left] > a[largest])
        largest = left;
    // If right child is larger than largest so far
    if (right < n && a[right] > a[largest])
        largest = right;
    // If largest is not root
    if (largest != i)
    {
        swap(a[i], a[largest]);
        heapify(a, n, largest);
    }
}
```

```
// If right child is larger than root
```

```
if (right < n && a[right] > a[largest])
```

```
    largest = right;
```

```
// If root is not largest
```

```
if (largest != i) {
```

```
    // swap a[i] with a[largest]
```

```
    int temp = a[i];
```

```
    a[i] = a[largest];
```

```
    a[largest] = temp;
```

```
    heapify(a, n, largest);
```

```
}
```

```
}
```

```
/*Function to implement the heap sort*/
```

```
void heapSort(int a[], int n)
```

```
{
```

```
for (int i = n / 2 - 1; i >= 0; i--)
```

```
    heapify(a, n, i);
```

```
    // One by one extract an element from heap
```

```
for (int i = n - 1; i >= 0; i--) {
```

```
    /* Move current root element to end*/
```

```
    // swap a[0] with a[i]
```

```
    int temp = a[0];
```

```
    a[0] = a[i];
```

```
    a[i] = temp;
```

```
    heapify(a, i, 0);
```

```
}
```

```
}
```

```
/* function to print the array elements */
```

```
void printArr(int a[], int n)
```

```
{
```

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

```
{
```

```
    cout<<a[i]<<" ";
```

```
}
```

```
}
```

```
int main()
```

```
{
```

```

int a[] = {47, 9, 22, 42, 27, 25, 0};
int n = sizeof(a) / sizeof(a[0]);
cout<<"Before sorting array elements are - \n";
printArr(a, n);
heapSort(a, n);
cout<<"\nAfter sorting array elements are - \n";
printArr(a, n);
return 0;
}

```

9.2 Complexity of the Heap Sort

Worst Case: $O(n \log n)$

Best Case: $O(n \log n)$

Average Case: $O(n \log n)$

9.3 Heap Sort Applications

Systems concerned with security and embedded systems such as Linux Kernel use Heap Sort because of the $O(n \log n)$ upper bound on Heapsort's running time and constant $O(1)$ upper bound on its auxiliary storage.

Although Heap Sort has $O(n \log n)$ time complexity even for the worst case, it doesn't have more applications (compared to other sorting algorithms like Quick Sort, Merge Sort). However, its underlying data structure, heap, can be efficiently used if we want to extract the smallest (or largest) from the list of items without the overhead of keeping the remaining items in the sorted order. For e.g Priority Queues.

K sorted array

K largest or smallest elements in an array

9.4 Advantages of HeapSort

Efficiency: As the number of objects to sort grows, the time required to conduct Heap sort grows logarithmically, whereas alternative methods may grow exponentially slower. This is a very efficient sorting algorithm.

Memory usage: Memory usage is modest because it requires no additional memory space to work other than what is required to keep the initial list of objects to be sorted.

Simplicity: Because it does not involve difficult computer science concepts like recursion, it is easier to understand than other equally efficient sorting algorithms.

9.5 Binomial Heap

A binomial Heap is a collection of Binomial Trees that satisfies the heap properties, i.e., min heap. It supports quicker merging of two heaps in $O(\log n)$. A min heap is a heap in which each node has a value lesser than the value of its child nodes.

A Binomial tree is a tree in which B_k is an ordered tree defined recursively, where k is defined as the order of the binomial tree. The binomial tree B_0 consists of a single node. The binomial tree B_k consists of two binomial trees B_{k-1} that are linked together, the root of one is the leftmost child of the root of the other.

If the binomial tree is represented as B₀ then the tree consists of a single node. In general terms, B_k consists of two binomial trees, i.e., B_{k-1} and B_{k-1} are linked together in which one tree becomes the left sub tree of another binomial tree.

Binomial Tree B₀

If B₀, k= 0, there would be only one node in the tree



B₀

Binomial Tree B₁

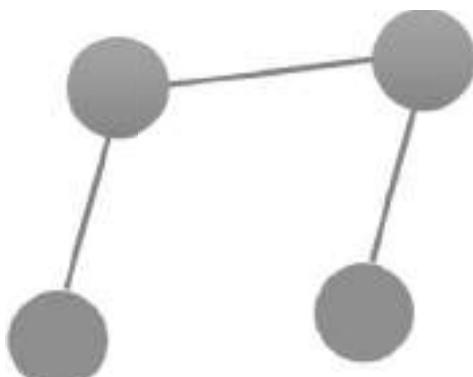
If B₁, k= 1, means k-1 equal to 0. Therefore, there would be two binomial trees of B₀ in which one B₀ becomes the left sub tree of another B₀.



B₁

Binomial Tree B₂

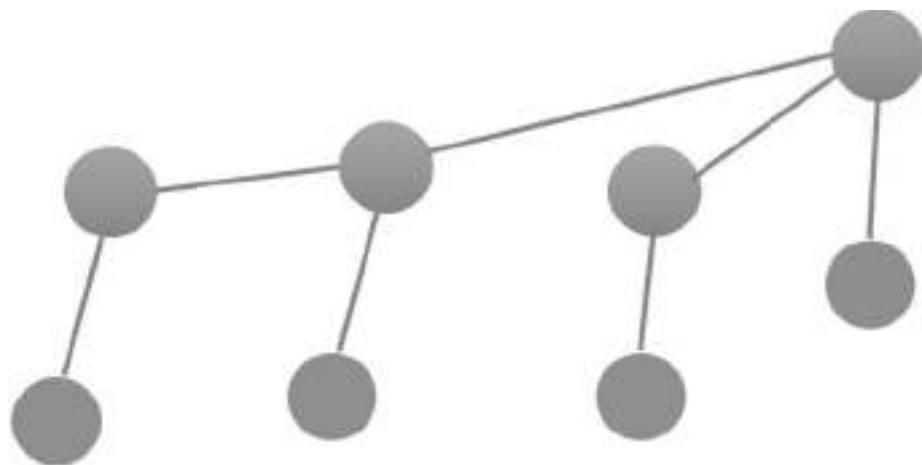
If B₂, k= 2, means k-1 equal to 1. Therefore, there would be two binomial trees of B₁ in which one B₁ becomes the left sub tree of another B₁.



B₂

Binomial Tree B₃

If B₃ , k= 3, means k-1 equal to 2. Therefore, there would be two binomial trees of B₂ in which one B₂ becomes the left sub tree of another B₂.



9.6 Operations of Binomial Heap

- Union of two binomial heap
- Finding the minimum key
- Creating a new binomial heap
- Inserting a node
- Extracting minimum key
- Decreasing a key
- Deleting a node

Union of two binomial heap

Merging in a heap can be done by comparing the keys at the roots of two trees, and the root node with the larger key will become the child of the root with a smaller key than the other. The time complexity for finding a union is $O(\log n)$.

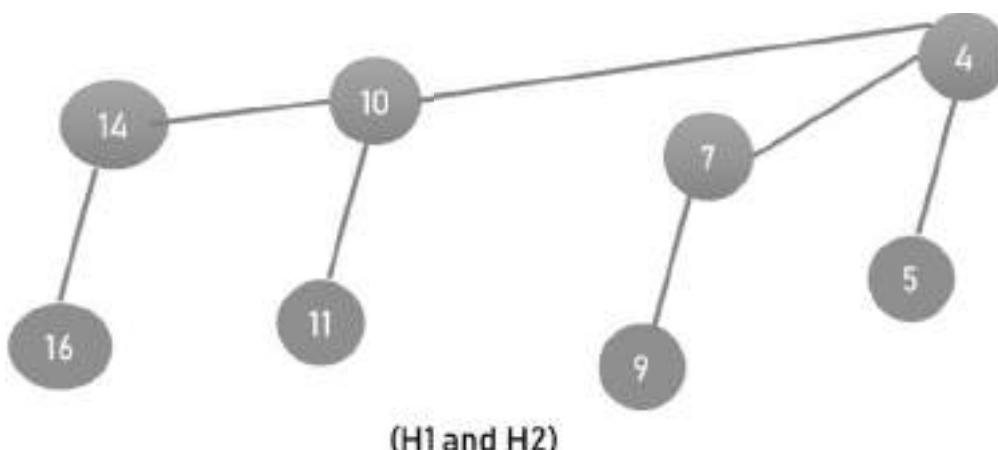
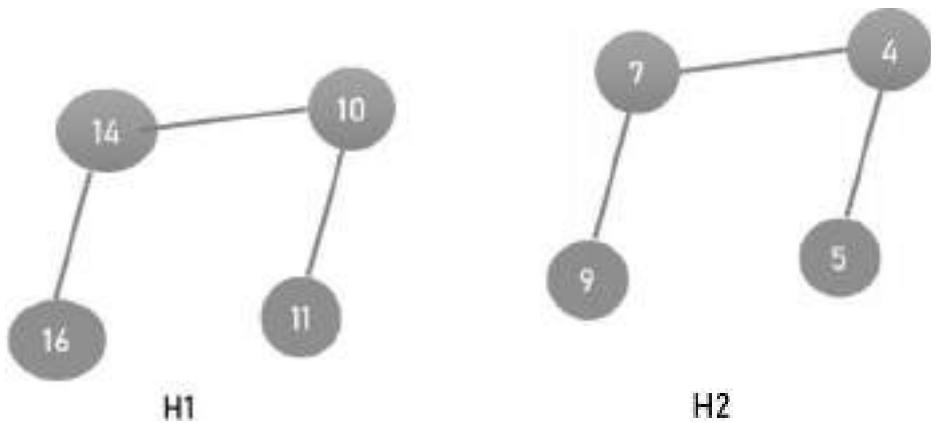
Case 1: If $\text{degree}[x]$ is not equal to $\text{degree}[\text{next } x]$ then move pointer ahead.

Case 2: if $\text{degree}[x] = \text{degree}[\text{next } x] = \text{degree}[\text{sibling}(\text{next } x)]$ then

Move pointer ahead.

Case 3: If $\text{degree}[x] = \text{degree}[\text{next } x]$ but not equal to $\text{degree}[\text{sibling}[\text{next } x]]$ and $\text{key}[x] < \text{key}[\text{next } x]$ then remove $[\text{next } x]$ from root and attached to x .

Case 4: If $\text{degree}[x] = \text{degree}[\text{next } x]$ but not equal to $\text{degree}[\text{sibling}[\text{next } x]]$ and $\text{key}[x] > \text{key}[\text{next } x]$ then remove x from root and attached to $[\text{next } x]$.



Find minimum

To find the minimum element of the heap, find the minimum among the roots of the binomial trees. It requires $O(\log n)$ time. It can be optimized to $O(1)$ by maintaining a pointer to minimum key root.

Decrease Key

We compare the decreases key with its parent and if parent's key is more, we swap keys and recur for the parent. Swap process stop when we either reach a node whose parent has a smaller key or we hit the root node. Time complexity of decrease Key() is $O(\log n)$.

Extract minimum key

First find this element, remove it from its binomial tree, and obtain a list of its sub trees. Transform this list of sub trees into a separate binomial heap by reordering them from smallest to largest order. Then merge this heap with the original heap. This operation requires $O(\log n)$ time.

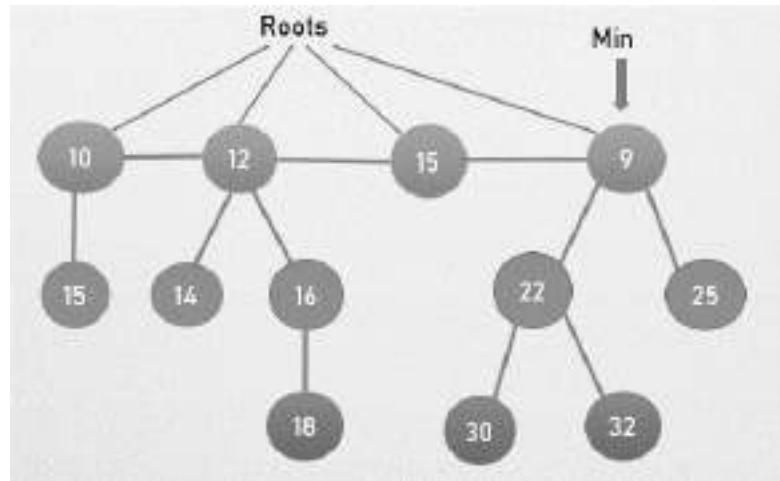
9.7 Fibonacci Heap

A Fibonacci heap is a circular doubly linked list, with a pointer to the minimum key, but the elements of the list are not single keys. Instead, we collect keys together into structures called binomial heaps. Binomial heaps are trees that satisfy the heap property — every node has a smaller key than its children.

Fibonacci heap data structure is collection of trees which follow min heap or max heap property. In a Fibonacci heap, a node can have more than two children or no children at all.

Properties of a Fibonacci Heap

- A pointer is maintained at the minimum element node.
- The trees within a Fibonacci heap are unordered but rooted.
- It is a set of min heap-ordered trees.
- It consists of a set of marked nodes.



The child nodes of a parent node are connected to each other through a circular doubly linked list. Deleting a node from the tree takes $O(1)$ time. The concatenation of two such lists takes $O(1)$ time.

Fibonacci heaps have a faster amortized running time than other heap types. Fibonacci heaps have a less rigid structure as compared to binomial heaps. Fibonacci heaps are used to implement the priority queue element in Dijkstra's algorithm. The reduced time complexity of Decrease-Key has importance in Dijkstra and Prim algorithms. With Binary Heap, time complexity of these algorithms is $O(V\log V + E\log V)$. If Fibonacci Heap is used, then time complexity is improved to $O(V\log V + E)$.

9.8 Operations on a Fibonacci Heap

- Insertion
- Find Min
- Union
- Extract Min
- Decrease a key
- Delete a node

Insertion

- Create a new node for the element.
- Check if the heap is empty.
- If the heap is empty, set the new node as a root node and mark it min.
- Else, insert the node into the root list and update min.

Algorithm: Insertion

```
insert(H, x)
degree[x] = 0
```

```

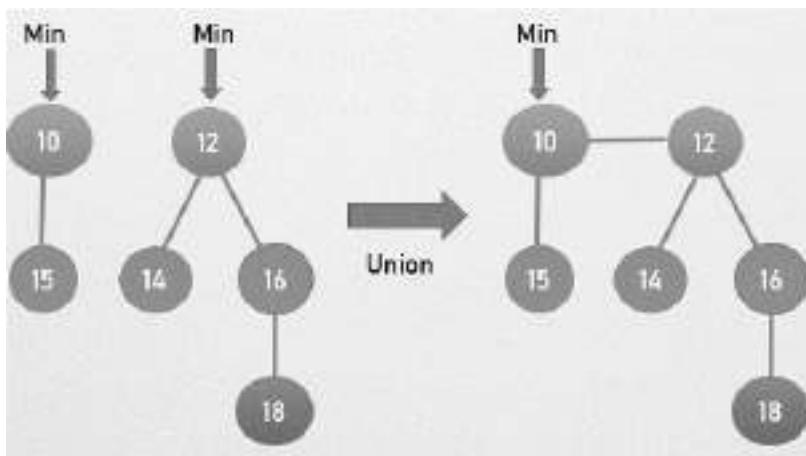
p[x] = NIL
child[x] = NIL
left[x] = x
right[x] = x
mark[x] = FALSE
concatenate the root list containing x with root list H
if min[H] == NIL or key[x] < key[min[H]]
    then min[H] = x
n[H] = n[H] + 1

```

Union

Steps for Union of two Fibonacci heaps.

- Concatenate the roots of both the heaps.
- Update min by selecting a minimum key from the new root lists.



Extract Min

In extract min minimum value is removed from the heap and the tree is re-adjusted.

Steps for Extract Min

- Delete the min node.
- Set the min-pointer to the next root in the root list.
- Create an array of size equal to the maximum degree of the trees in the heap before deletion.
- Do the following (steps 5-7) until there are no multiple roots with the same degree.
 - Map the degree of current root (min-pointer) to the degree in the array.
 - Map the degree of next root to the degree in array.
 - If there are more than two mappings for the same degree, then apply union operation to those roots such that the min-heap property is maintained (i.e. the minimum is at the root).

Decrease Key

In decreasing a key operation, the value of a key is decreased to a lower value.

Decrease the value of the node 'x' to the new chosen value.

CASE 1 - If min heap order not violated,

Update min pointer if necessary.

CASE 2 - If min heap order violated and parent of 'x' is unmarked,

Cut off the link between 'x' and its parent.

Mark the parent of 'x'.

Add tree rooted at 'x' to the root list and update min pointer if necessary.

CASE 3 - If min heap order is violated and parent of 'x' is marked,

Cut off the link between 'x' and its parent p[x].

Add 'x' to the root list, updating min pointer if necessary.

Cut off link between p[x] and p[p[x]].

Add p[x] to the root list, updating min pointer if necessary.

If p[p[x]] is unmarked, mark it.

Else, cut off p[p[x]] and repeat steps 4.2 to 4.5, taking p[p[x]] as 'x'.

Deleting a Node

This process makes use of decrease-key and extract-min operations. The following steps are followed for deleting a node.

Let k be the node to be deleted.

Apply decrease-key operation to decrease the value of k to the lowest possible value (i.e. $-\infty$).

Apply extract-min operation to remove this node.



Lab Exercise: Operations on a Fibonacci

```
#include <cmath>
#include <cstdlib>
#include <iostream>
using namespace std;

struct node {
    int n;
    int degree;
    node *parent;
    node *child;
    node *left;
    node *right;
    char mark;
    char C;
};

class FibonacciHeap {
private:
    int nH;

    node *H;
public:
    node *InitializeHeap();
    int Fibonacci_link(node *, node *, node *);
}
```

```

node *Create_node(int);
node *Insert(node *, node *);
node *Union(node *, node *);
node *Extract_Min(node *);
int Consolidate(node *);
int Display(node *);
node *Find(node *, int);
int Decrease_key(node *, int, int);
int Delete_key(node *, int);
int Cut(node *, node *, node *);
int Cascase_cut(node *, node *);
FibonacciHeap() { H = InitializeHeap(); }
};

// Initialize heap
node *FibonacciHeap::InitializeHeap() {
    node *np;
    np = NULL;
    return np;
}

// Create node
node *FibonacciHeap::Create_node(int value) {
    node *x = new node;
    x->n = value;
    return x;
}

// Insert node
node *FibonacciHeap::Insert(node *H, node *x) {
    x->degree = 0;
    x->parent = NULL;
    x->child = NULL;
    x->left = x;
    x->right = x;
    x->mark = 'F';
    x->C = 'N';
    if (H != NULL) {
        (H->left)->right = x;
        x->right = H;
        x->left = H->left;
        H->left = x;
        if (x->n < H->n)
            H = x;
    }
}

```

```

} else {
    H = x;
}
nH = nH + 1;
return H;
}

// Create linking
int FibonacciHeap::Fibonacci_link(node *H1, node *y, node *z) {
    (y->left)->right = y->right;
    (y->right)->left = y->left;
    if (z->right == z)
        H1 = z;
    y->left = y;
    y->right = y;
    y->parent = z;
    if (z->child == NULL)
        z->child = y;
    y->right = z->child;
    y->left = (z->child)->left;
    ((z->child)->left)->right = y;
    (z->child)->left = y;
    if (y->n < (z->child)->n)
        z->child = y;
    z->degree++;
}

// Union Operation
node *FibonacciHeap::Union(node *H1, node *H2) {
    node *np;
    node *H = InitializeHeap();
    H = H1;
    (H->left)->right = H2;
    (H2->left)->right = H;
    np = H->left;
    H->left = H2->left;
    H2->left = np;
    return H;
}

// Display the heap
int FibonacciHeap::Display(node *H) {
    node *p = H;
    if (p == NULL) {

```

```

cout << "Empty Heap" << endl;
return 0;
}

cout << "Root Nodes: " << endl;
do {
    cout << p->n;
    p = p->right;
    if (p != H) {
        cout << "-->";
    }
} while (p != H && p->right != NULL);
cout << endl;
}

// Extract min
node *FibonacciHeap::Extract_Min(node *H1) {
    node *p;
    node *ptr;
    node *z = H1;
    p = z;
    ptr = z;
    if (z == NULL)
        return z;
    node *x;
    node *np;
    x = NULL;
    if (z->child != NULL)
        x = z->child;
    if (x != NULL) {
        ptr = x;
        do {
            np = x->right;
            (H1->left)->right = x;
            x->right = H1;
            x->left = H1->left;
            H1->left = x;
            if (x->n < H1->n)
                H1 = x;
            x->parent = NULL;
            x = np;
        } while (np != ptr);
    }
}

```

```

(z->left)->right = z->right;
(z->right)->left = z->left;
H1 = z->right;

if (z == z->right && z->child == NULL)
    H = NULL;
else {
    H1 = z->right;
    Consolidate(H1);
}
nH = nH - 1;
return p;
}

// Consolidation Function
int FibonacciHeap::Consolidate(node *H1) {
    int d, i;
    float f = (log(nH)) / (log(2));
    int D = f;
    node *A[D];
    for (i = 0; i <= D; i++)
        A[i] = NULL;
    node *x = H1;
    node *y;
    node *np;
    node *pt = x;
    do {
        pt = pt->right;
        d = x->degree;
        while (A[d] != NULL)
        {
            y = A[d];
            if (x->n > y->n)
            {
                np = x;
                x = y;
                y = np;
            }
            if (y == H1)
                H1 = x;
            Fibonacci_link(H1, y, x);
        }
        A[d] = NULL;
    }
    while (d < D);
    return H1;
}

```

```

if (x->right == x)
    H1 = x;
A[d] = NULL;
d = d + 1;
}
A[d] = x;
x = x->right;
}

while (x != H1);
H = NULL;
for (int j = 0; j <= D; j++) {
    if (A[j] != NULL) {
        A[j]->left = A[j];
        A[j]->right = A[j];
        if (H != NULL) {
            (H->left)->right = A[j];
            A[j]->right = H;
            A[j]->left = H->left;
            H->left = A[j];
            if (A[j]->n < H->n)
                H = A[j];
        } else {
            H = A[j];
        }
        if (H == NULL)
            H = A[j];
        else if (A[j]->n < H->n)
            H = A[j];
    }
}
}

// Decrease Key Operation
int FibonacciHeap::Decrease_key(node *H1, int x, int k) {
    node *y;
    if (H1 == NULL) {
        cout << "The Heap is Empty" << endl;
        return 0;
    }
    node *ptr = Find(H1, x);
    if (ptr == NULL) {

```

```

cout << "Node not found in the Heap" << endl;
return 1;
}
if (ptr->n < k) {
    cout << "Entered key greater than current key" << endl;
    return 0;
}
ptr->n = k;
y = ptr->parent;
if (y != NULL && ptr->n < y->n) {
    Cut(H1, ptr, y);
    Cascase_cut(H1, y);
}
if (ptr->n < H->n)
    H = ptr;
return 0;
}
// Cutting Function
int FibonacciHeap::Cut(node *H1, node *x, node *y)
{
if (x == x->right)
    y->child = NULL;
(x->left)->right = x->right;
(x->right)->left = x->left;
if (x == y->child)
    y->child = x->right;
y->degree = y->degree - 1;
x->right = x;
x->left = x;
(H1->left)->right = x;
x->right = H1;
x->left = H1->left;
H1->left = x;
x->parent = NULL;
x->mark = 'F';
}
// Cascade cut
int FibonacciHeap::Cascase_cut(node *H1, node *y) {
node *z = y->parent;
if (z != NULL) {
    if (y->mark == 'F') {

```

```

y->mark = 'T';
} else
{
    Cut(H1, y, z);
    Cascase_cut(H1, z);
}
}

// Search function
node *FibonacciHeap::Find(node *H, int k) {
    node *x = H;
    x->C = 'Y';
    node *p = NULL;
    if (x->n == k) {
        p = x;
        x->C = 'N';
        return p;
    }
    if (p == NULL) {
        if (x->child != NULL)
            p = Find(x->child, k);
        if ((x->right)->C != 'Y')
            p = Find(x->right, k);
    }
    x->C = 'N';
    return p;
}

// Deleting key
int FibonacciHeap::Delete_key(node *H1, int k) {
    node *np = NULL;
    int t;
    t = Decrease_key(H1, k, -5000);
    if (!t)
        np = Extract_Min(H);
    if (np != NULL)
        cout << "Key Deleted" << endl;
    else
        cout << "Key not Deleted" << endl;
    return 0;
}

```

```

int main() {
    int n, m, l;
    FibonacciHeap fh;
    node *p;
    node *H;
    H = fh.InitializeHeap();
    p = fh.Create_node(7);
    H = fh.Insert(H, p);
    p = fh.Create_node(3);
    H = fh.Insert(H, p);
    p = fh.Create_node(17);
    H = fh.Insert(H, p);
    p = fh.Create_node(24);
    H = fh.Insert(H, p);
    fh.Display(H);
    p = fh.Extract_Min(H);
    if (p != NULL)
        cout << "The node with minimum key: " << p->n << endl;
    else
        cout << "Heap is empty" << endl;
    m = 26;
    l = 16;
    fh.Decrease_key(H, m, l);
    m = 16;
    fh.Delete_key(H, m);
}

```

Complexities

Insertion	O(1)
Find Min	O(1)
Union	O(1)
Extract Min	O(log n)
Decrease Key	O(1)
Delete Node	O(log n)

Summary

- Heap sort is sorting technique based upon Binary Heap data structure. It is comparison-based sorting technique.
- The elements of a heap sort are processed by generating a min-heap or max-heap with the items of the provided array.

- Advantages of heapsort are Efficiency, Memory usage and Simplicity
- A binomial Heap is a collection of Binomial Trees that satisfies the heap properties, i.e., min heap.
- A Binomial tree is a tree in which B_k is an ordered tree defined recursively, where k is defined as the order of the binomial tree.
- Fibonacci heap data structure is collection of trees which follow min heap or max heap property. In a Fibonacci heap, a node can have more than two children or no children at all.

Keywords

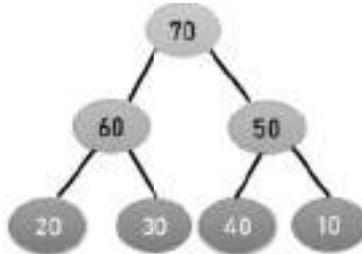
Max heap	Min heap
Binomial Heap	Heap sort
Extract Min	Decrease Key

Self Assessment

1. Heap sort is_____
 - A. It is based upon Binary Heap data structure.
 - B. It is comparison-based sorting technique.
 - C. It processes the elements by creating the min heap or max heap using the elements of the array.
 - D. All of above
2. Elements arranged in descending order_____
 - A. Max heap
 - B. Min heap
 - C. Both Max and Min heap
 - D. None of above
3. Heapify is part of_____
 - A. Max heap
 - B. Min heap
 - C. Both Max and Min heap
 - D. None of above
4. Elements arranged in ascending order_____
 - A. Max heap
 - B. Min heap
 - C. Both Max and Min heap
 - D. None of above
5. Complexity of the Heap Sort in worst case is_____
 - A. $(\log 1)$
 - B. $(\log n)$

- C. $(n \log n)$
- D. None of above

6. Given graph is example of ____



- A. Min heap
- B. Max heap
- C. Both max and min heap
- D. None of above

7. Binomial Heap is a collection of ____

- A. Binary trees.
- B. AVL trees.
- C. Binomial trees.
- D. None of above

8. In binomial tree B1 what are the numbers of nodes.

- A. 0
- B. 1
- C. 2
- D. 3

9. Operations of Binomial Heap ____

- A. Finding the minimum key
- B. Creating a new binomial heap
- C. Inserting a node
- D. All of above

10. What is value of K in binomial tree B3?

- A. 1
- B. 2
- C. 3
- D. None of above

11. Properties of a Fibonacci Heap are ____

- A. It is a set of min heap-ordered trees.
- B. It consists of a set of marked nodes.

- C. The trees within a Fibonacci heap are unordered but rooted.
 D. All of above
12. Which is not Fibonacci Heap operation.
 A. Union
 B. Extract Min
 C. Peek
 D. Decrease a key
13. A pointer is maintained in Fibonacci Heap at the _____ element node
 A. Maximum
 B. Minimum
 C. Both minimum and maximum
 D. None of above
14. The child nodes of a parent node are connected to each other through_____
 A. Doubly linked list
 B. Singly linked list
 C. Circular doubly linked list
 D. None of above
15. Deleting a node from the tree in Fibonacci Heap takes _____ time.
 A. (1)
 B. (0)
 C. ($\log n$)
 D. None of above

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. D | 2. A | 3. C | 4. B | 5. C |
| 6. B | 7. C | 8. C | 9. D | 10. C |
| 11. D | 12. D | 13. B | 14. C | 15. C |

Review Questions

1. What are the steps for heap sort operation?
2. Write algorithm for heap sort.
3. Explain complexity of heap sort.
4. Define binomial Heap with suitable example.
5. Discuss different operations of binomial heap
6. Describe insert and union operations in Fibonacci Heap.
7. Explain different cases of Decrease Key.



Further Readings

- Burkhard Monien, Data Structures and Efficient Algorithms, Thomas Ottmann, Springer.
- Kruse, Data Structure & Program Design, Prentice Hall of India, New Delhi.
- Mark Allen Weis, Data Structure & Algorithm Analysis in C, Second Ed., Addison-Wesley Publishing.
- RG Dromey, How to Solve it by Computer, Cambridge University Press.
- Lipschutz. S. (2011). Data Structures with C. Delhi: Tata McGraw hill
- Reddy. P. (1999). Data Structures Using C. Bangalore: Sri Nandi Publications
- Samantha. D (2009). Classic Data Structures. New Delhi: PHI Learning Private Limited



Web Links

www.en.wikipedia.org

www.web-source.net

www.webopedia.com

<https://www.tutorialspoint.com/fibonacci-heaps-in-data-structure>

<https://www.cl.cam.ac.uk/teaching/1415/Algorithms/fibonacci.pdf>

<http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap20.htm>

<http://www.cs.toronto.edu/~anikolov/CSC265F18/binomial-heaps.pdf>

Unit 10: Graphs

CONTENTS

- Objectives
- Introduction
- 10.1 Graphs
- 10.2 Graph Terminology
- 10.3 Types of Graphs
- 10.4 Representations of Graphs
- 10.5 Connected Components
- 10.6 Spanning Tree
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to:

- Understand basics of graphs
- Learn basic graph terminology
- Discuss adjacency matrix and linked adjacency chains
- learn spanning trees

Introduction

In this unit, we introduce you to an important mathematical structure called Graph. Graphs have found applications in subjects as diverse as Sociology, Chemistry, Geography and Engineering Sciences. They are also widely used in solving games and puzzles. In computer science, graphs are used in many areas one of which is computer design. In day-to-day applications, graphs find their importance as representations of many kinds of physical structure.

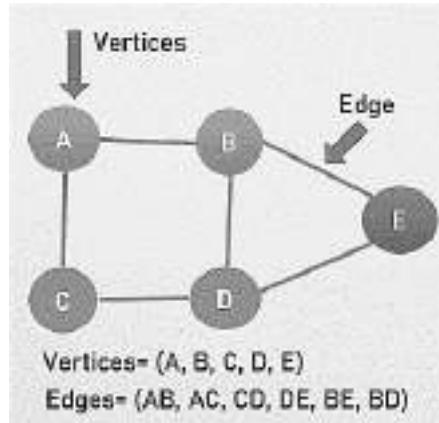
We use graphs as models of practical situations involving routes: the vertices represent the cities and edges represent the roads or some other links, specially in transportation management, Assignment problems and many more optimization problems. Electric circuits are another obvious example where interconnections between objects play a central role. Circuit's elements like transistors, resistors, and capacitors are intricately wired together. Such circuits can be represented and processed within a computer in order to answer simple questions like "Is everything connected together?" as well as complicated questions like "If this circuit is built, will it work?"

10.1 Graphs

A Graph G consists of a set V of vertices (nodes) and a set E of edges (arcs). We write $G=(V,E)$. V is a finite and non empty set of vertices. E is a set of pairs of vertices; these pairs are called edges. Therefore

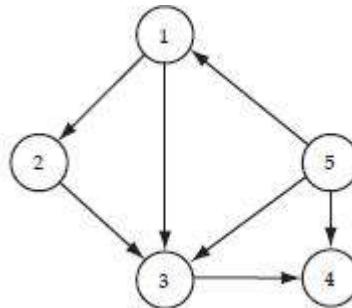
$V(G)$, read as V of G , is set of vertices, and $E(G)$, read as E of G , is set of edges.

An edge $e = (v,w)$, is a pair of vertices v and w , and is said to be incident with v and w . It is a pictorial representation of a set of objects where objects are connected by links. A graph may be pictorially represented as given in Figure



In an undirected graph, pair of vertices representing any edge is unordered. Thus (v,w) and (w,v) represent the same edge. In a directed graph each edge is an ordered pair of vertices, i.e. each edge is represented by a directed pair. If $e = (v,w)$, then v is tail or initial vertex and w is head or final vertex. Subsequently (v,w) and (w,v) represent two different edges.

A directed graph may be pictorially represented as given in Figure



Directed graph

The direction is indicated by an arrow. The set of vertices for this graph remains the same as that of the graph in the earlier example, i.e.

$$V(G) = \{1, 2, 3, 4, 5\}$$

However the set of edges would be

$$E(G) = \{(1,2), (2,3), (3,4), (5,4), (5,1), (1,3), (5,3)\}$$

10.2 Graph Terminology

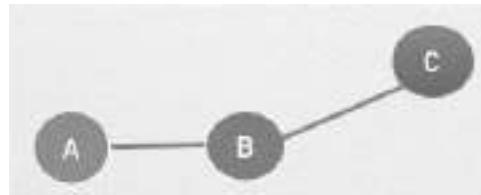
A good deal of nomenclature is associated with graphs. Most of the terms have straight forward definitions, and it is convenient to put them in one place even though we would not be using some of them until later.

- Vertices
- Edges
- Path
- Closed path
- Degree of the Node
- Adjacent Nodes/ Adjacency

Vertices: Each node of the graph is represented as a vertex.

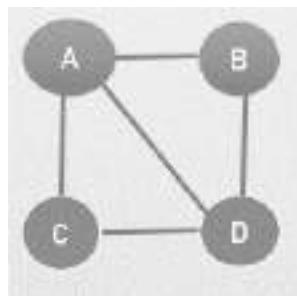
Edge: it is used to represent the relationships between various nodes in a graph. An edge between two nodes expresses a one-way or two-way relationship between the nodes.

Path: Path represents a sequence of edges between the two vertices. E.g. ABC



Closed Path: A path will be called as closed path if the initial node is same as terminal node.

Degree of the Node: A degree of a node is the number of edges that are connected with that node.
Degree of A=3.



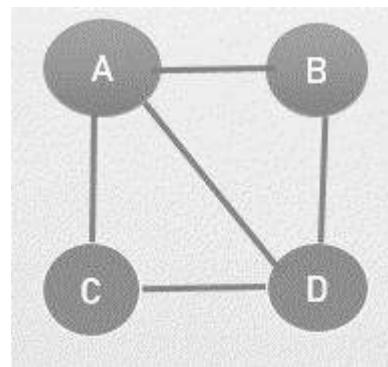
Adjacent Nodes/ Adjacency: if two nodes are connected to each other through an edge are called as neighbors or adjacent nodes.

10.3 Types of Graphs

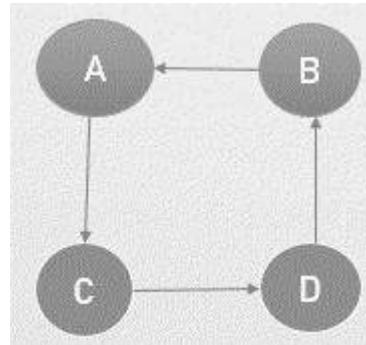
- Undirected Graph
- Directed Graph
- Weighted Graph
- Un-weighted Graph
- Complete Graph
- Finite Graph
- Trivial Graph
- Multi Graph
- Pseudo Graph
- Connected Graph
- Labeled Graphs
- Disconnected Graph

Undirected graph

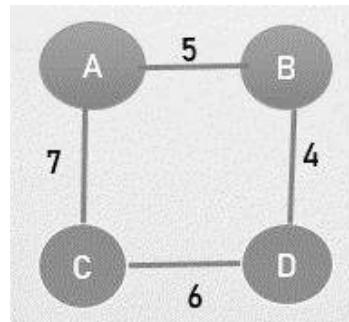
An undirected graph nodes are connected and all the edges are bi-directional i.e. the edges do not point in any specific direction.

**Directed graph**

A directed graph is a graph in which all the edges are uni-directional i.e. the edges point in a single direction. It is also called a digraph.

**Weighted graph**

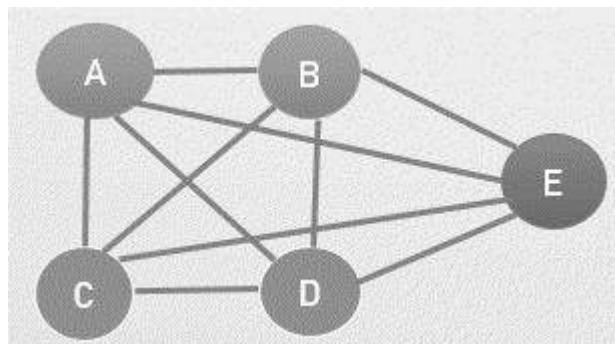
In weighted graph edges or path have values or cost. All the values seen associated with the edges are called weights.

**Un-weighted graph**

In un-weighted graph there is no value or weight associated with the edge. By default, all the graphs are un-weighted.

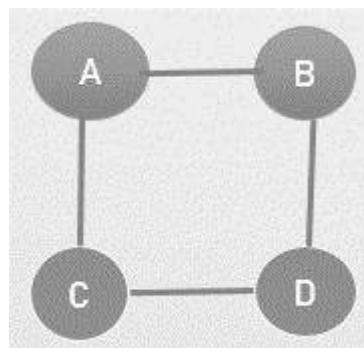
Complete graph

A complete graph is the one in which every node is connected with all other nodes. A complete graph contain $n(n-1)/2$ edges where n is the number of nodes in the graph.



Finite graph

The graph $G=(V, E)$ is called a finite graph if the number of vertices and edges in the graph is limited in number



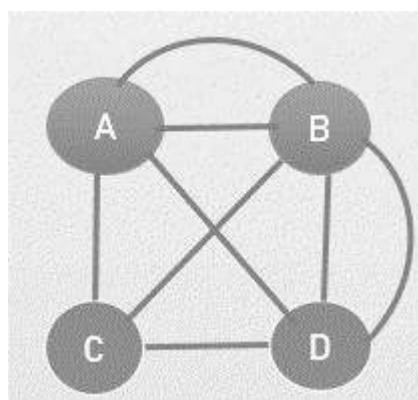
Trivial Graph

A graph $G= (V, E)$ is trivial if it contains only a single vertex and no edges.



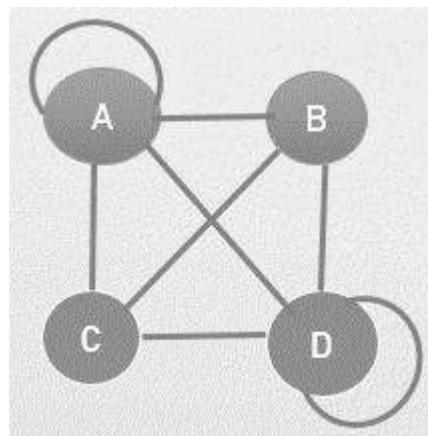
Multi Graph

If there are numerous edges between a pair of vertices in a graph $G= (V, E)$, the graph is referred to as a multi graph. There are no self-loops in a Multi graph.



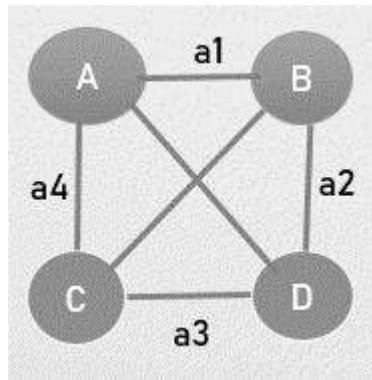
Pseudo graph

If a graph $G= (V, E)$ contains a self-loop besides other edges, it is a pseudo graph.



Labeled graph

A graph $G=(V, E)$ is called a labeled graph if its edges are labeled with some name or data.



10.4 Representations of Graphs

Graph is a mathematical structure and finds its application in many areas of interest in which problems need to be solved using computers. Thus, this mathematical structure must be represented as some kind of data structures. Two such representations are, commonly used.

There are various ways to represent a graph. A simple representation is given by an adjacency list, which specifies all vertices adjacent to each vertex of the graph. This list can be implemented as a table, in which case it is called a star representation, which can be forward or reverse.

Another representation is a matrix, which comes in two forms: an adjacency matrix and an incidence matrix. An adjacency matrix of graph $G = (V,E)$ is a binary $|V| \times |V|$ matrix such that each entry of this matrix.

These are:

1. Adjacent Matrix
2. Adjacency List representation.

The choice of representation depends on the application and function to be performed on the graph.

Adjacent Matrix

Two vertices is called adjacent or neighbor if it support at least one common edge. A finite graph can be represented in the form of a square matrix. Boolean value (0,1) of the matrix indicates if there is a direct path between two vertices.

It is also called 2D matrix that is used to map the association between the graph nodes. If a graph has n number of vertices, then the adjacency matrix of that graph is $n \times n$, and each entry of the matrix represents the number of edges from one vertex to another.

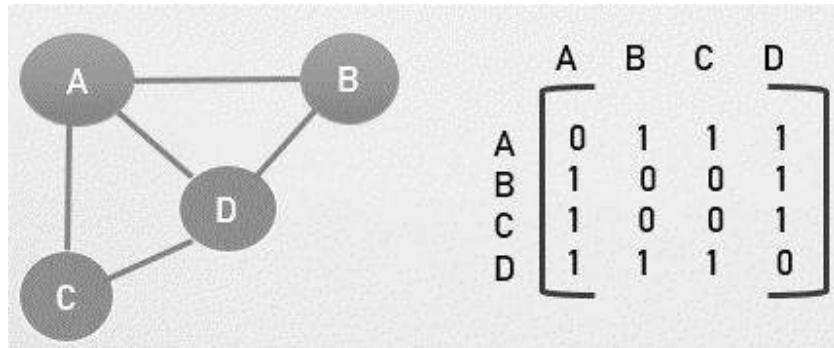
Adjacency Matrix Representation

The adjacency matrix A for a graph $G = (V, E)$ with n vertices, is an $n \times n$ matrix of bits, such that A

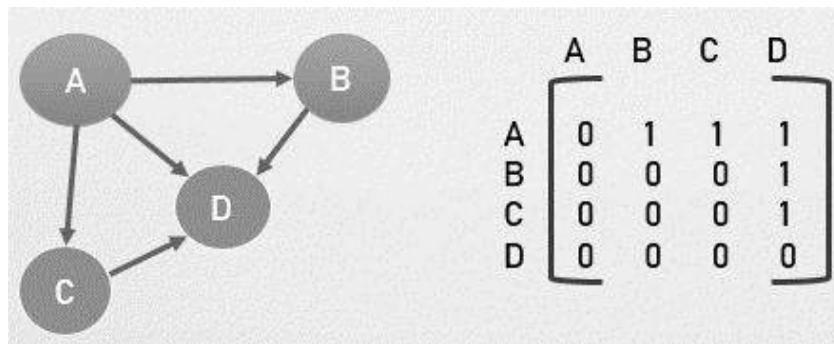
$A_{ij} = 1$, iff there is an edge from v_i to v_j and

$A_{ij} = 0$, if there is no such edge.

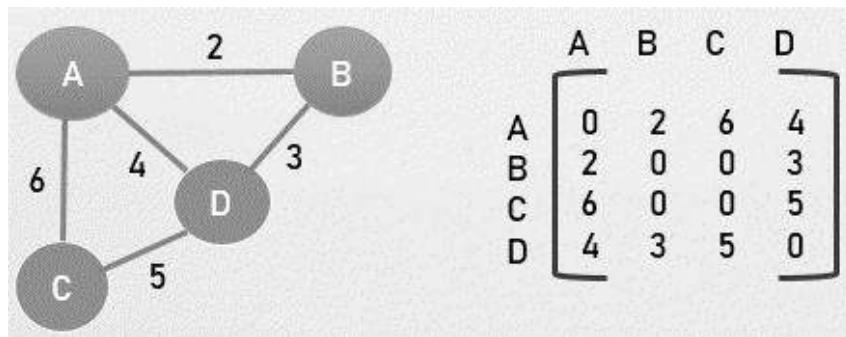
Undirected Graph Representation



Directed Graph Representation



Undirected Weighted Graph

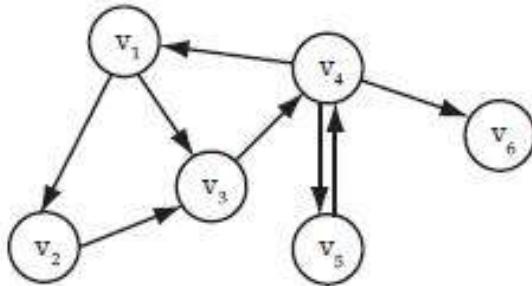


Applications: Adjacency Matrix

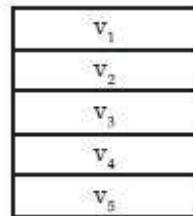
- Navigation tasks
- It is used to represent finite graphs
- Creating routing table in networks

Adjacency List Representation

In this representation, we store a graph as a linked structure. We store all the vertices in a list and then for each vertex, we have a linked list of its adjacent vertices.

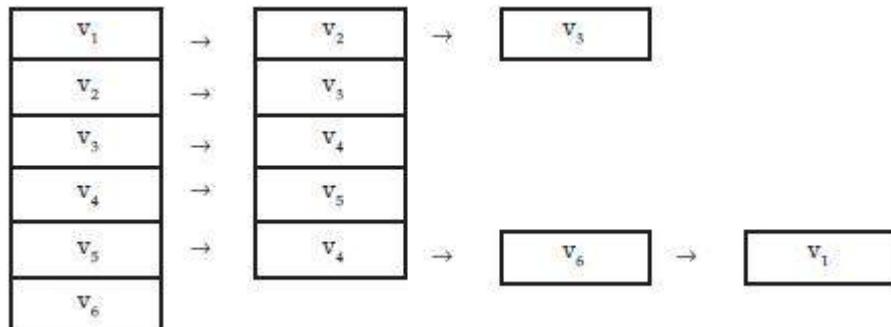


The adjacency list representation needs a list of all of its nodes, i.e.



And for each node a linked list of its adjacent nodes.

Therefore we shall have



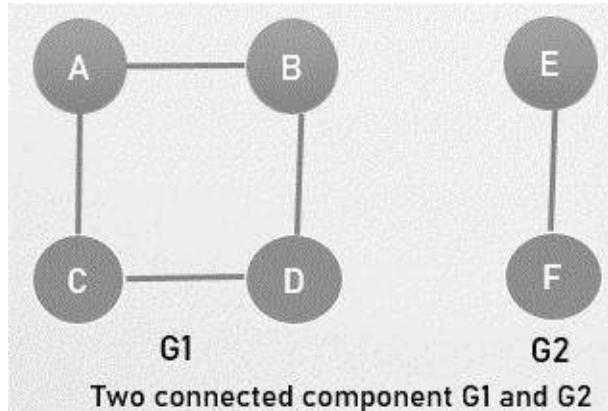
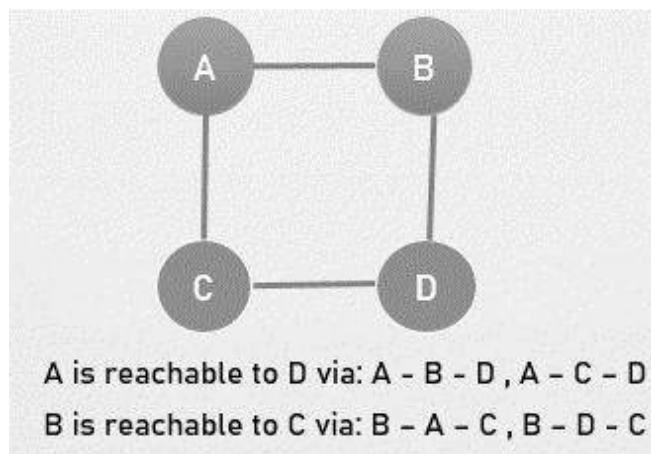
Adjacency List Structure for Graph

The adjacency list representation is better for sparse graphs because the space required is $O(V + E)$, as contrasted with the $O(V^2)$ required by the adjacency matrix representation.

10.5 Connected Components

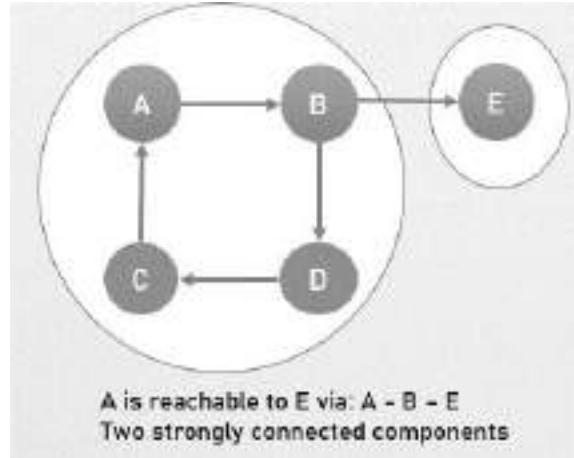
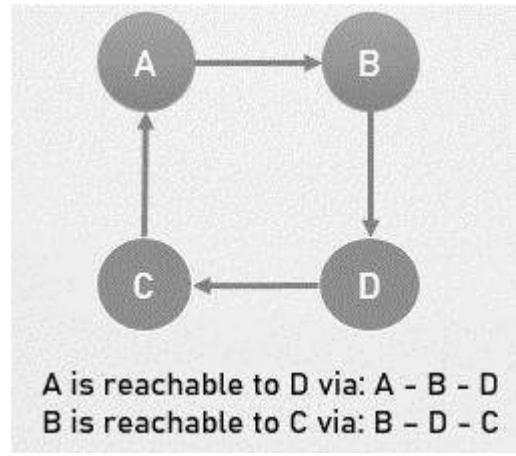
Component of an undirected graph is a sub graph in which each pair of nodes is connected with each other via a path. Every vertex of the graph lies in a connected component that consists of all the vertices that can be reached from that vertex, together with all the edges that join those vertices.

We can use traversal algorithm, depth-first or breadth-first, to find the connected components of an undirected graph.



Strongly Connected Component

For a directed graph, a strongly connected component has a directed path between any two nodes.



10.6 Spanning Tree

A spanning tree is a tree that connects all the vertices of a graph with the minimum possible number of edges. Thus, a spanning tree is always connected. Also, a spanning tree never contains a cycle. A spanning tree is always defined for a graph and it is always a subset of that graph. Thus, a disconnected graph can never have a spanning tree.

A spanning tree is a sub-graph of an undirected connected graph, which has all the vertices covered with minimum possible number of edges. If a vertex is missed, then it is not a spanning tree. A spanning tree does not have cycles and it cannot be disconnected.

Every undirected and connected graph has a minimum of one spanning tree. Consider a graph having V vertices and E number of edges. Then, we will represent the graph as $G(V, E)$. Its spanning tree will be represented as $G'(V, E')$ where $E' \subseteq E$ and the number of vertices remain the same. So, a spanning tree G' is a subgraph of G whose vertex set is the same but edges may be different.

Spanning Trees Terminologies

Undirected graph: An undirected graph is a graph in which all the edges do not point to any particular direction.

Connected graph: A connected graph is a graph in which a path always exists from a vertex to any other vertex. A graph is connected if we can reach any vertex from any other vertex by following edges in either direction.

Directed graph: A directed graph is defined as a graph in which set of V vertices and set of Edges, each connecting two different vertices, but it is not mandatory that node points in the opposite direction also.

Properties of Spanning Tree

- An undirected connected graph can have more than one spanning tree.
- All the possible spanning trees of a graph have the same number of edges and vertices.
- The spanning tree does not have any cycle / loops.
- Any connected and undirected graph will always have at least one spanning tree.
- Spanning tree is always minimally connected. Removing one edge from the spanning tree will make the graph disconnected
- A spanning tree is maximally acyclic. Adding one edge to the spanning tree will create a cycle or loop.

Mathematical Properties of Spanning Tree

- Spanning tree has $n-1$ edges, where n is the number of nodes (vertices).
- A complete graph can have maximum n^{n-2} number of spanning trees.
- From a complete graph, by removing maximum $e - n + 1$ edges, we can construct a spanning tree.



Example: If we have $n = 4$, the maximum number of possible spanning trees is equal to $4^{4-2} = 16$. Thus, 16 spanning trees can be formed from a complete graph with 4 vertices.

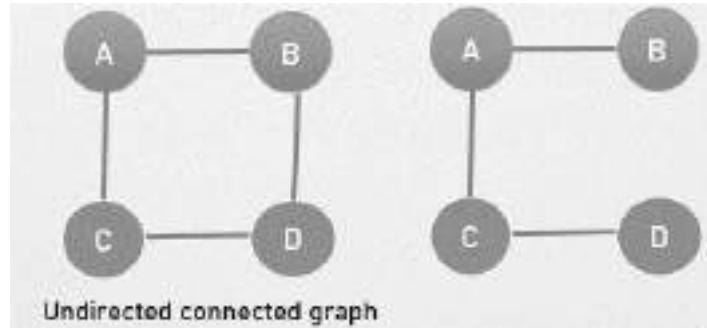


Fig. a

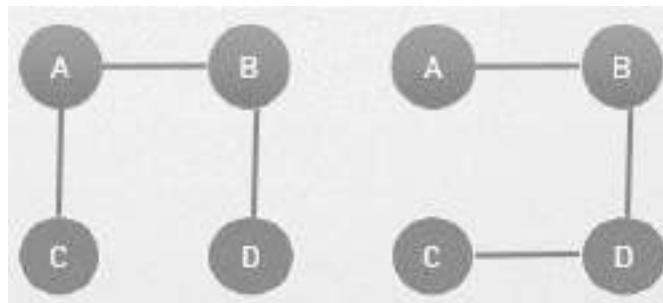


Fig. b

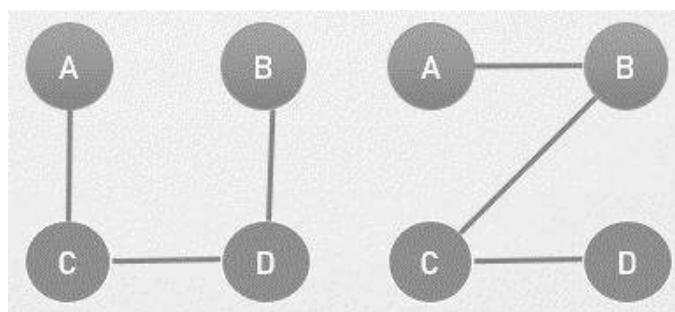


Fig. c

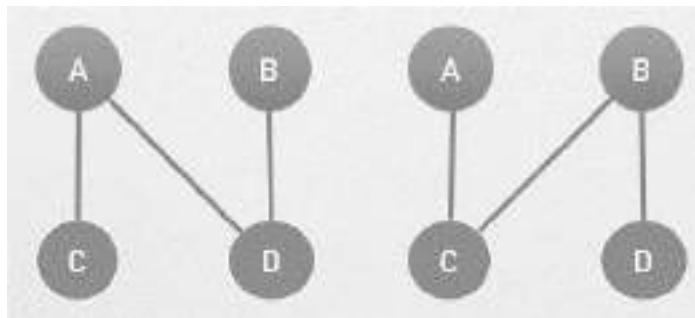


Fig. d

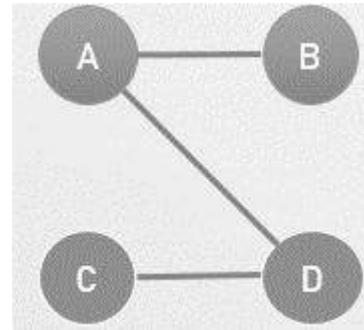


Fig. e

Application of Spanning Tree

It is used to find a minimum path to connect all nodes in a graph

Computer Network Routing Protocol

Cluster Analysis

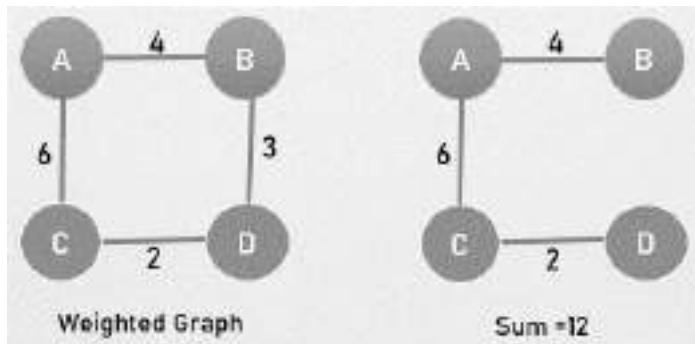
Minimum Spanning Tree

A minimum spanning tree is defined for a weighted graph. A spanning tree having minimum weight is defined as a minimum spanning tree. This weight depends on the weight of the edges. In real-world applications, the weight could be the distance between two points, cost associated with the edges or simply an arbitrary value associated with the edges.

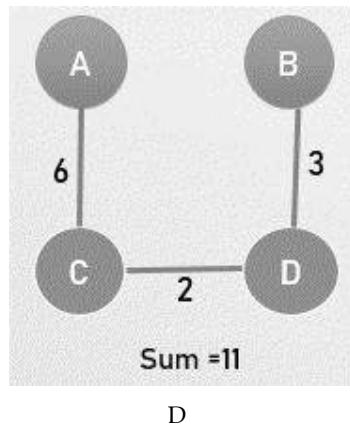
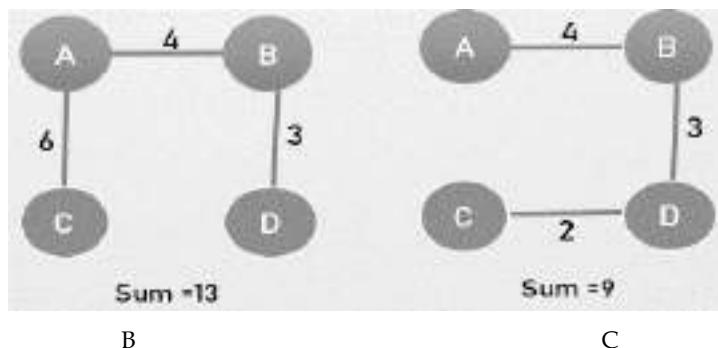
A minimum spanning tree is a spanning tree in which the sum of the weight of the edges is as minimum as possible.



Example: Minimum Spanning Tree



A



Minimum spanning tree = C, sum =9.

Summary

- A Graph G consists of a set V of vertex (nodes) and a set E of edges (arcs)
- An undirected graph nodes are connected and all the edges are bi-directional i.e. the edges do not point in any specific direction.
- If there are numerous edges between a pair of vertices in a graph $G = (V, E)$, the graph is referred to as a multi graph. There are no self-loops in a Multi graph.
- Two vertices is called adjacent or neighbor if it support at least one common edge. A finite graph can be represented in the form of a square matrix.
- Every undirected and connected graph has a minimum of one spanning tree.

Keywords

Vertices	Edges
Path	Closed path
Degree of the Node	Spanning tree

Self Assessment

- Graph is collection of _____
 - Vertices
 - Edges
 - Both vertices and edges

- D. None of above

- 2. Which is not part of graph.
 - A. Path
 - B. Extract Min
 - C. Edge
 - D. Closed path

- 3. Types of Graph are_____
 - A. Pseudo
 - B. Trivial
 - C. Disconnected
 - D. All of above

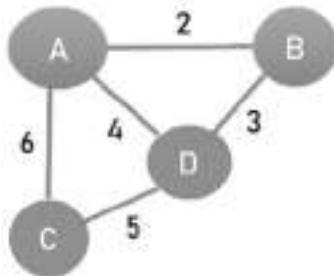
- 4. A complete graph is _____
 - A. Connected with all other nodes
 - B. Connected with bidirectional nodes
 - C. Connected with directional nodes
 - D. None of above

- 5. Graphs are commonly represent using ____
 - A. Adjacency Matrix
 - B. Adjacency List
 - C. Both Adjacency Matrix and Adjacency List
 - D. None of above

- 6. Two vertices is called adjacent.
 - A. If it there is no common edge
 - B. If it support at least two common edge
 - C. If it support at least one common edge
 - D. None of above

- 7. Applications of adjacency matrix are_____
 - A. Navigation tasks
 - B. It is used to represent finite graphs
 - C. Creating routing table in networks
 - D. All of above

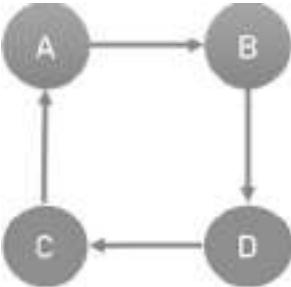
- 8. Image represent a _____



- A. Undirected Weighted Graph
 B. Directed Graph Representation
 C. Undirected Graph Representation
 D. None of above
9. To find the connected components of an undirected graph____
- A. Depth-first search algorithm
 B. Dijkstra's Algorithm
 C. Centrality Algorithms
 D. None of above

10. Strongly connected components are____
- A. Undirected path between any two nodes
 B. Directed path between any two nodes
 C. It support at least two common edge
 D. None of above

11. Graph represents ____



- A. Undirected Connected Component
 B. Bidirectional Connected Component
 C. Strongly Connected Component
 D. All of above

12. Which is not type of graph ____
- A. Connected
 B. Directed
 C. Centrality
 D. Bidirectional

13. Spanning tree is _____

- A. Have more than one spanning tree
- B. Have the same number of edges and vertices
- C. Does not have any cycle / loops
- D. All of above

14. Mathematical properties of spanning tree _____

- A. Has $n-1$ edges, $n =$ number of nodes
- B. Complete graph can have maximum n^{n-2} number of spanning trees
- C. All of above

15. Minimum Spanning Tree is _____

- A. The sum of the weight of the edges is as minimum as possible
- B. The sum of the weight of the edges is as maximum as possible
- C. The sum of the weight of the edges is as average as possible
- D. None of above

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. C | 2. B | 3. D | 4. A | 5. C |
| 6. C | 7. D | 8. A | 9. A | 10. B |
| 11. C | 12. C | 13. D | 14. D | 15. A |

Review Questions

1. Define graph and its different types.
2. Discuss edge and vertices with example.
3. How to find degree of node?
4. Differentiate between directed and weighted graph with example.
5. Give an example of Adjacency List representation.
6. How spanning tree is different from minimum spanning tree?
7. What are the applications of spanning tree?



Further Readings

- Burkhard Monien, Data Structures and Efficient Algorithms, Thomas Ottmann, Springer.
- Kruse, Data Structure & Program Design, Prentice Hall of India, New Delhi.
- Mark Allen Weles, Data Structure & Algorithm Analysis in C, Second Ed., Addison-Wesley Publishing.
- RG Dromey, How to Solve it by Computer, Cambridge University Press.
- Lipschutz. S. (2011). Data Structures with C. Delhi: Tata McGraw hill

- Reddy. P. (1999). Data Structures Using C. Bangalore: Sri Nandi Publications
- Samantha. D (2009). Classic Data Structures. New Delhi: PHI Learning Private Limited



Web Links

www.en.wikipedia.org

www.web-source.net

www.webopedia.com

<https://www.javatpoint.com/spanning-tree>

<https://www.programiz.com/dsa/graph>

Unit 11: More on Graphs

CONTENTS

- Objectives
- Introduction
- 11.1 Breadth First Search (BFS)
- 11.2 Depth First Search
- 11.3 Network Flow Problem
- 11.4 Ford-Fulkerson Algorithm
- 11.5 Floyd-Warshall Algorithm
- 11.6 Topological Sort
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to:

- Understand breadth first search
- Learn depth first search
- Discuss network flow problems and warshall's algorithm
- Learn topological sort

Introduction

Graph traversal entails visiting each vertex and edge in a predetermined order. You must verify that each vertex of the graph is visited exactly once when utilizing certain graph algorithms. The sequence in which the vertices are visited is crucial, and it may be determined by the algorithm or question you're working on. It's critical to keep track of which vertices have been visited throughout a traversal. Marking vertices is the most popular method of tracking them.

In Graph traversal visiting every vertex and edge exactly once in a well-defined order. In graph algorithms, you must ensure that each vertex of the graph is visited exactly once. The order in which the vertices are visited may depend upon the algorithm or type of problem going to solve.

Two common elementary algorithms for tree-searching are

- Breadth-first search (BFS)
- Depth-first search (DFS).

Both of these algorithms work on directed or undirected graphs. Many advanced graph algorithms are based on the ideas of BFS or DFS. Each of these algorithms traverses edges in the graph, discovering new vertices as it proceeds. The difference is in the order in which each algorithm discovers the edges.

11.1 Breadth First Search (BFS)

Breadth first search is a graph traversal algorithm that starts traversing the graph from root node and explores all the neighbouring nodes. Then, it selects the nearest node and explore all the unexplored nodes. The algorithm follows the same process for each of the nearest node until it finds the goal.

For using BFS algorithm user should know about data structure queue and its relevant operations like en-queue and de-queue.

Algorithm: Breadth First Search

Step 1: SET STATUS = 1 (ready state)

for each node in G

Step 2: Enqueue the starting node A

and set its STATUS = 2

(waiting state)

Step 3: Repeat Steps 4 and 5 until

QUEUE is empty

Step 4: Dequeue a node N. Process it

and set its STATUS = 3

(processed state).

Step 5: Enqueue all the neighbours of

N that are in the ready state

(whose STATUS = 1) and set

their STATUS = 2

(waiting state)

[END OF LOOP]

Step 6: EXIT



Example: Breadth First Search

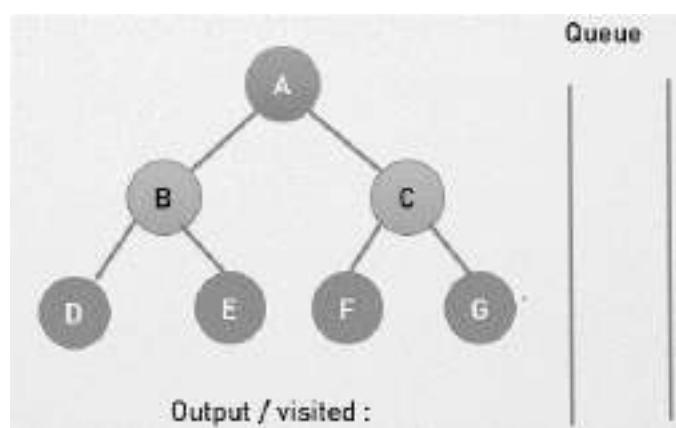


Fig (a)

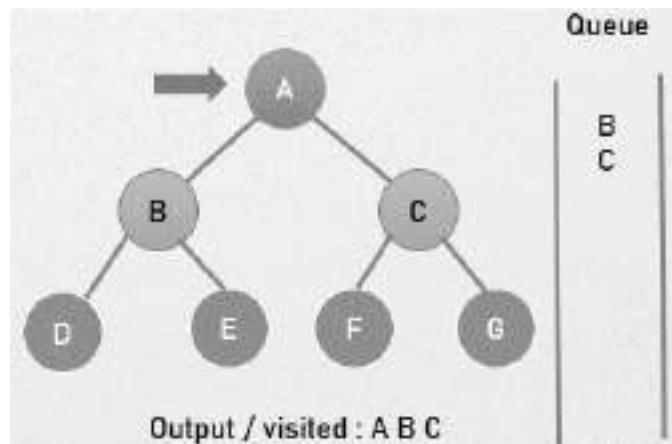


Fig (b)

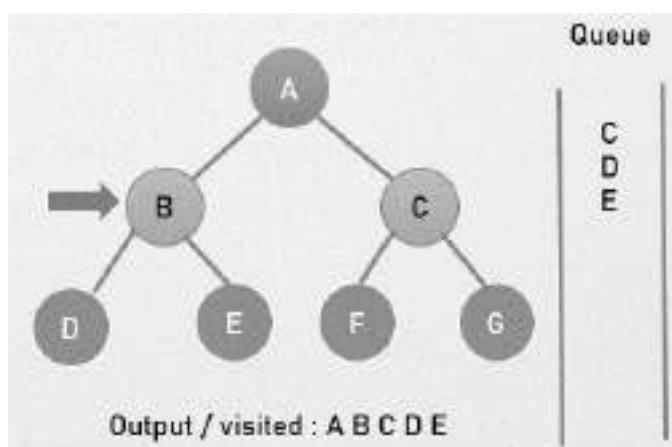


Fig (c)

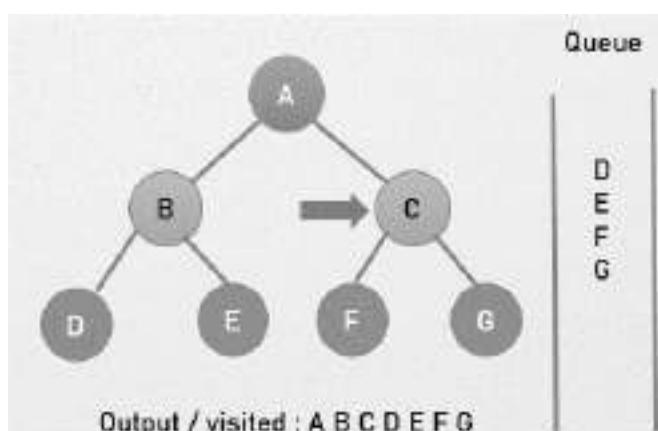


Fig (d)

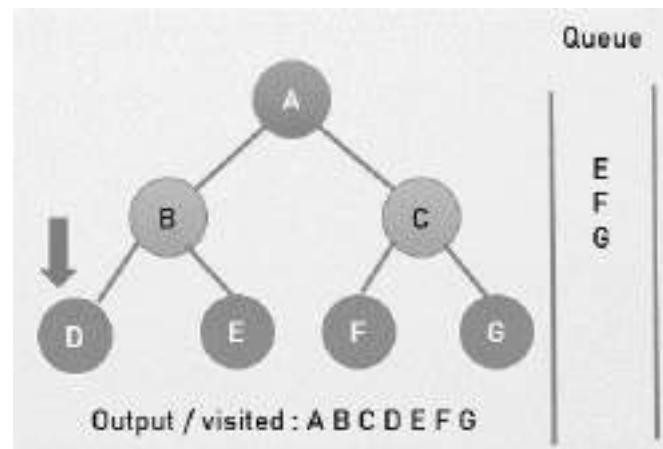


Fig (e)

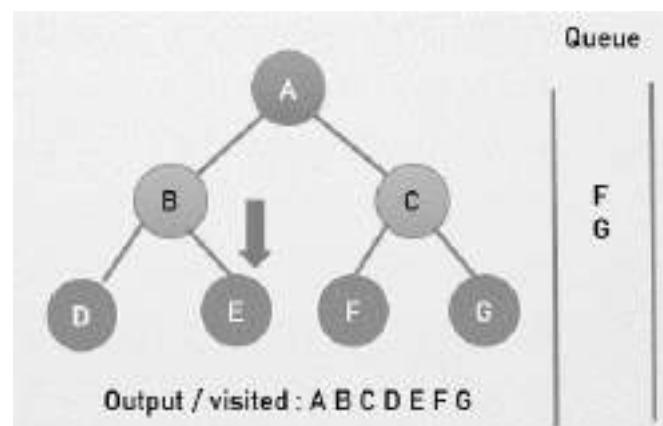


Fig (f)

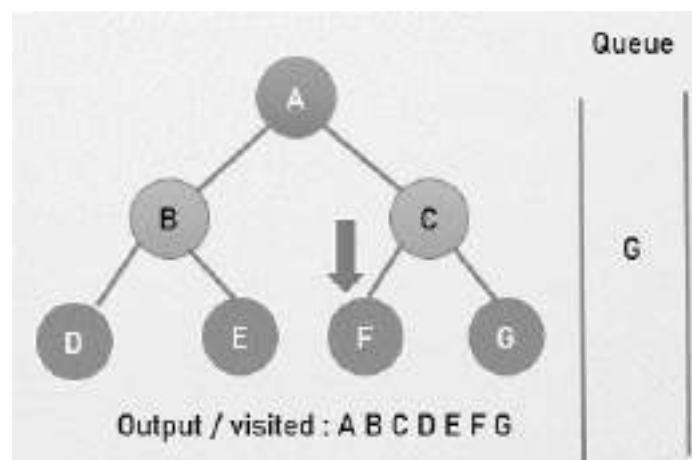


Fig (g)

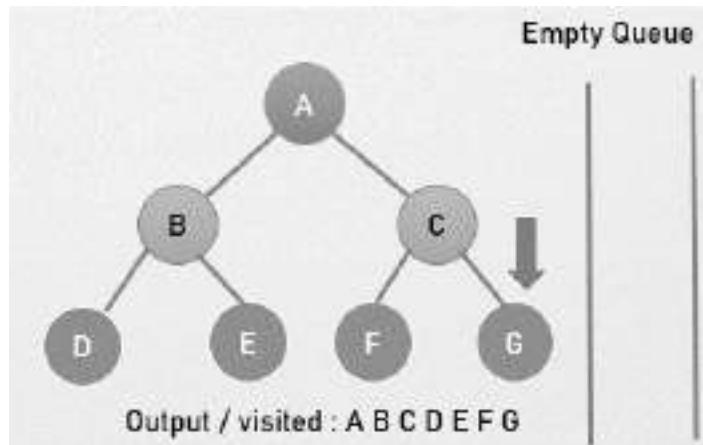


Fig (h)

Algorithm Complexity

The time complexity of the BFS algorithm is represented in the form of $O(V + E)$, where V is the number of nodes and E is the number of edges.

The space complexity of the algorithm is $O(V)$.

BFS Applications

- Path finding algorithms
- To build index by search index
- Cycle detection in an undirected graph
- For GPS navigation
- In minimum spanning tree
- Social networking websites

11.2 Depth First Search

Depth first search is another way of traversing graphs, which is closely related to preorder traversal of a tree. Recall that preorder traversal simply visits each node before its children. It is most easy to program as a recursive routine.

DFS traversal is a recursive algorithm for searching all the vertices/ nodes of a graph or tree using stack data structure. In Depth First Search (DFS) algorithm traverses a graph in a depth ward motion. The DFS algorithm use the concept of backtracking. Depth-first search (DFS): Finds a path between two vertices by exploring each possible path as far as possible before backtracking. Often implemented recursively. Many graph algorithms involve visiting or marking vertices.

Steps for DFS

Step 1 – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.

Step 2 – If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)

Step 3 – Repeat Rule 1 and Rule 2 until the stack is empty.

For using DFS algorithm user should know about data structure Stack (Last In First Out) and its relevant operations like Push and Pop.

Algorithm: Depth First Search

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until STACK is empty

Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)

Step 5: Push on the stack all the neighbours of N that are in the ready state (whose STATUS = 1) and set their

STATUS = 2 (waiting state)

[END OF LOOP]

Step 6: EXIT



Example: Depth First Search

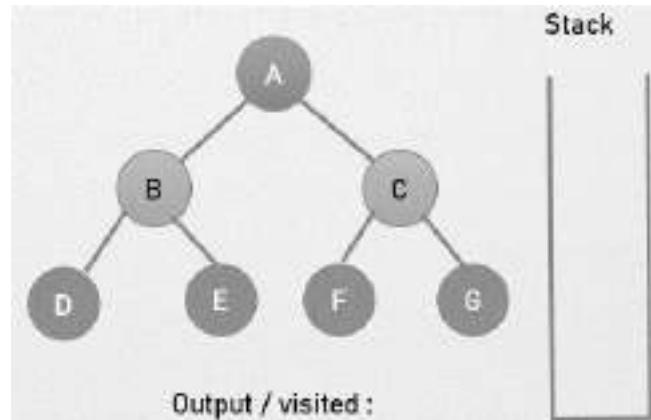


Fig (a)

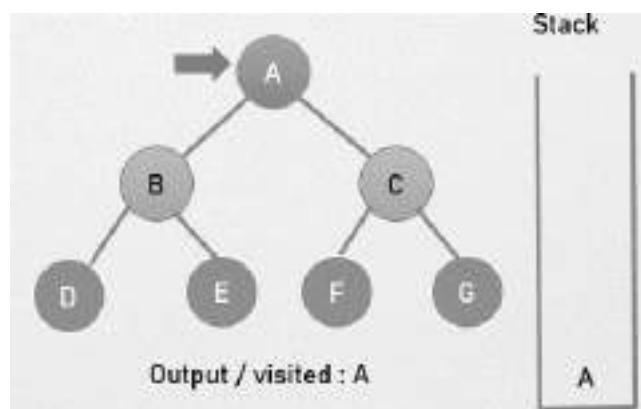


Fig (b)

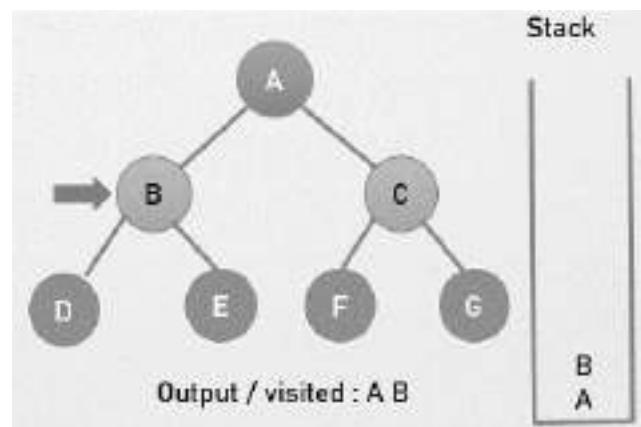


Fig (c)

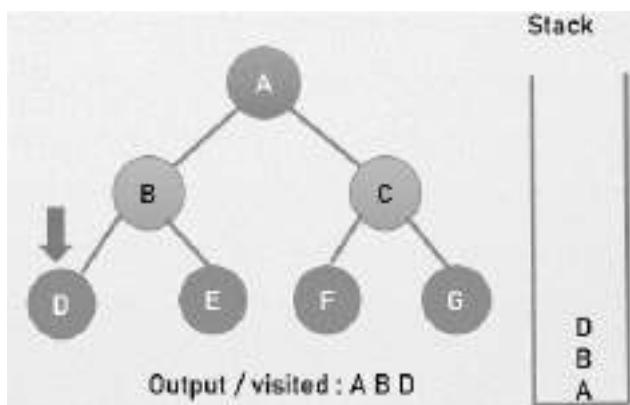


Fig (d)

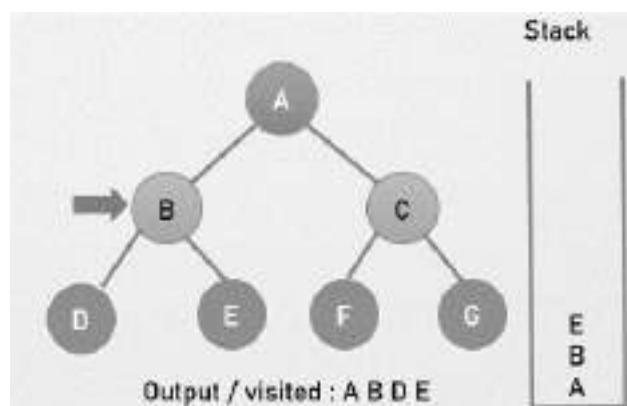


Fig (e)

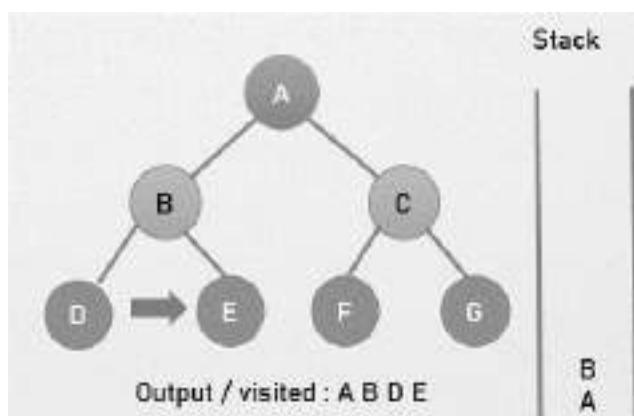


Fig (f)

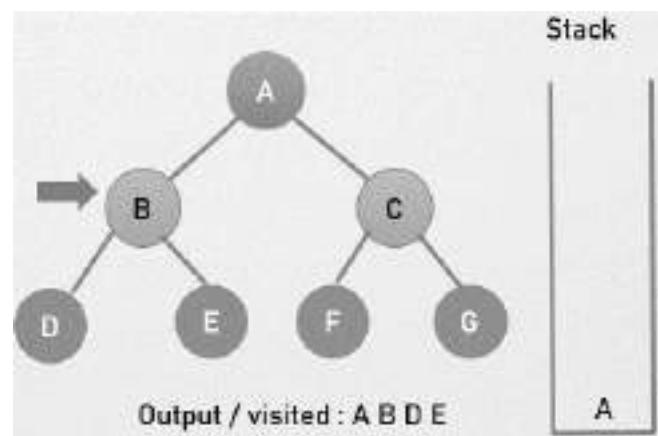


Fig (g)

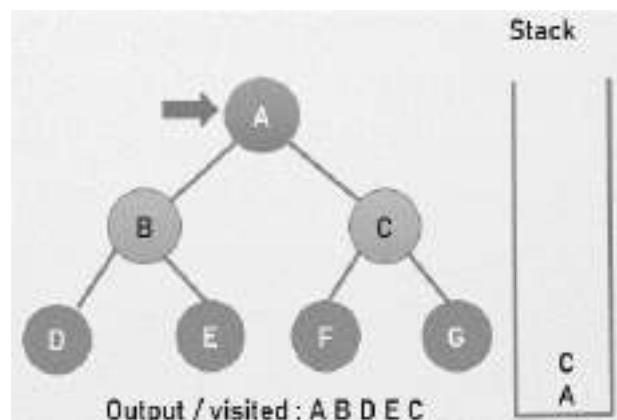


Fig (h)

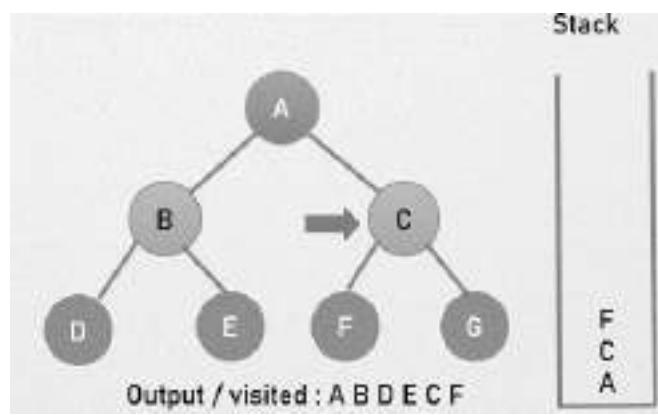


Fig (i)

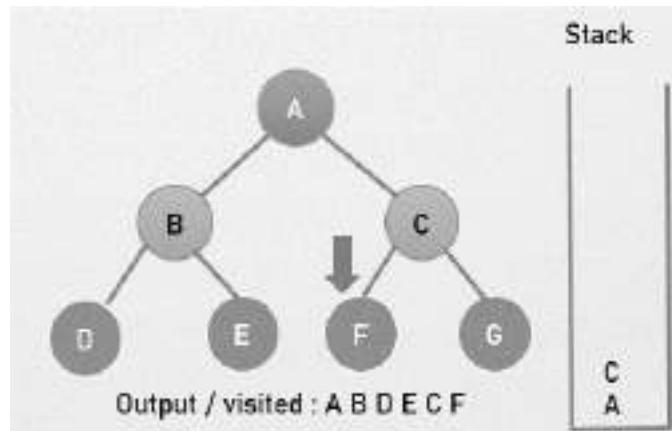


Fig (j)

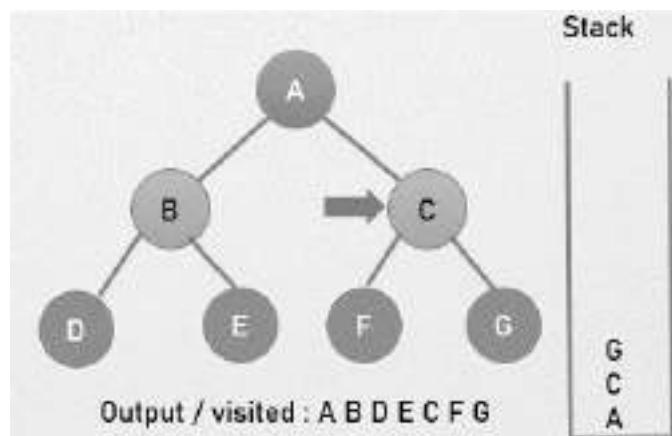


Fig (k)

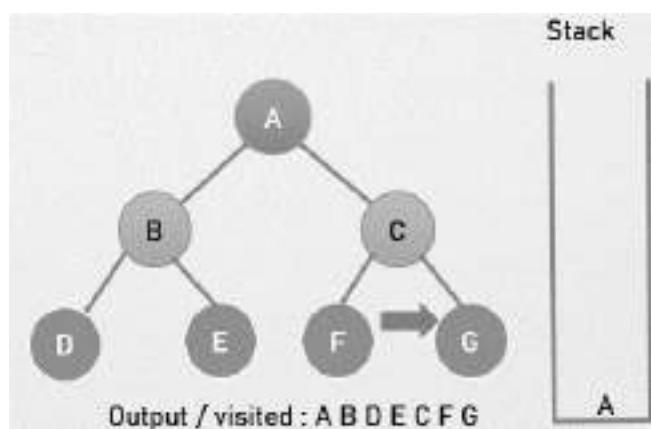


Fig (l)

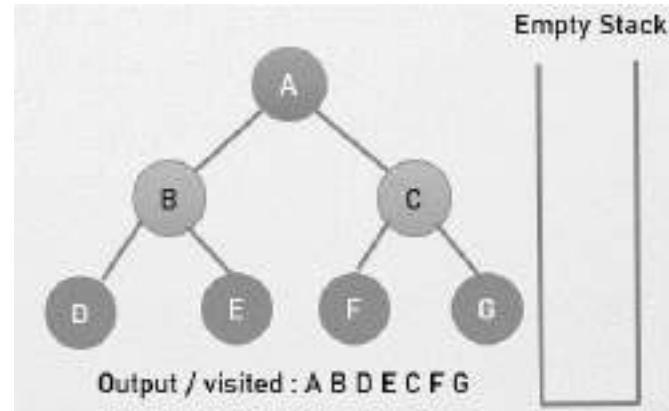


Fig (m)

Algorithm Complexity

Time complexity: $O(V + E)$, where V is the number of vertices and E is the number of edges in the graph.

Space Complexity: $O(V)$.

DFS Applications

- Mapping Routes and Network Analysis.
- Path Finding.
- Cycle detection in graphs.
- Topological Sorting.
- Solving puzzle.

11.3 Network Flow Problem

Network flow is an advanced branch of graph theory. The problem revolves around a special type of weighted directed graph with two special vertices: the source vertex, which has no incoming edge, and the sink vertex, which has no outgoing edge. By convention, the source vertex is usually labelled s and the sink vertex labelled t.

In graph theory, a flow network is a directed graph involving a source(s) and a sink(t) and several other nodes connected with edges. Each edge has an individual capacity which is the maximum limit of flow that edge could allow.

Network Flow Problems

Problem1: Given a flow network $G = (V, E)$, the maximum flow problem is to find a flow with maximum value.

Problem 2: The multiple source and sink maximum flow problem is similar to the maximum flow problem, except there is a set $\{s_1, s_2, s_3, \dots, s_n\}$ of sources and a set $\{t_1, t_2, t_3, \dots, t_n\}$ of sinks.

For any non-source and non-sink node, the input flow is equal to output flow.

For any edge (E_i) in the network, $0 \leq \text{flow } (E_i) \leq \text{capacity } (E_i)$.

Total flow out of the source node is equal total to flow in to the sink node.

Net flow in the edges follows skew symmetry

Problem: Maximize the total amount of flow from s to t. subject to two constraints

- Flow on edge e doesn't exceed $c(e)$
- For every node $v \neq s, t$, incoming flow is equal to outgoing flow

Types of network flow problems

Minimum-cost flow problem: in which the edges have costs as well as capacities and the goal is to achieve a given amount of flow (or a maximum flow) that has the minimum possible cost.

Multi-commodity flow problem: in which one must construct multiple flows for different commodities whose total flow amounts together respect the capacities.

Nowhere-zero flow: a type of flow studied in combinatorics in which the flow amounts are restricted to a finite set of nonzero values.

Maximum flow problem.

Algorithms for network flow problem

The **Ford-Fulkerson algorithm**, a greedy algorithm for maximum flow that is not in general strongly polynomial

Dinic's algorithm, a strongly polynomial algorithm for maximum flow

The **Edmonds-Karp algorithm**, a faster strongly polynomial algorithm for maximum flow.

The *network simplex algorithm*, a method based on linear programming but specialized for network flow.

The *out-of-kilter algorithm* for minimum-cost flow

The ***push-relabel maximum flow algorithm***, one of the most efficient known techniques for maximum flow.

11.4 Ford-Fulkerson Algorithm

It was developed by L. R. Ford, Jr. and D. R. Fulkerson in 1956. A simple and practical max-flow algorithm. Objective: find valid flow paths until there is none left, and add them up.

Terminology: Ford-Fulkerson Algorithm

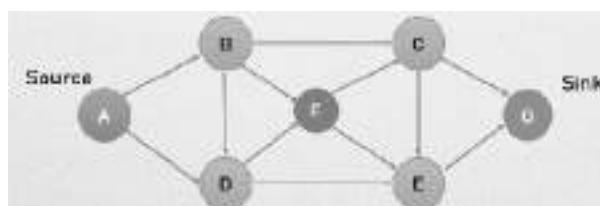
Source Sink

Bottleneck capacity Flow

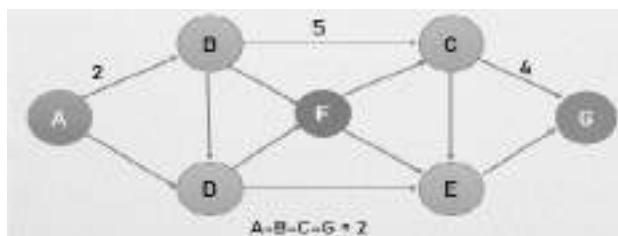
Augmenting path Residual capacity

The *source* vertex has all outward edges, no inward edges.

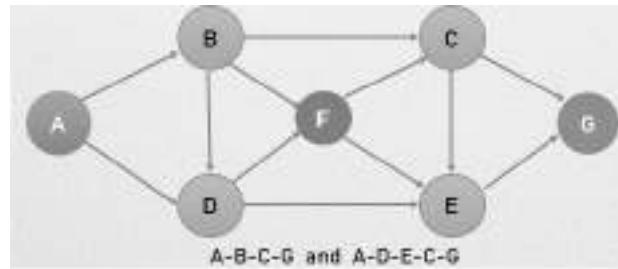
Sink have all inward edges, no outward edges.



Bottleneck capacity of a path is the minimum capacity of any edge on the path.



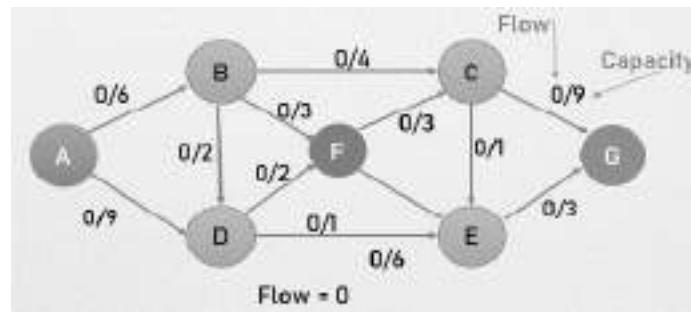
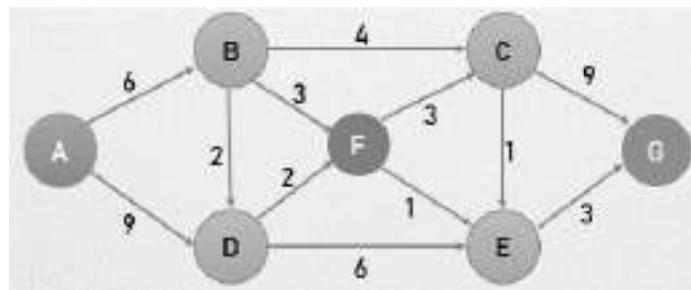
An **augmenting path** is a simple path from source to sink which do not include any cycles and that pass only through positive weighted edges.



Residual capacity: which is equal to original capacity of the edge minus current flow. Residual capacity is basically the current capacity of the edge.

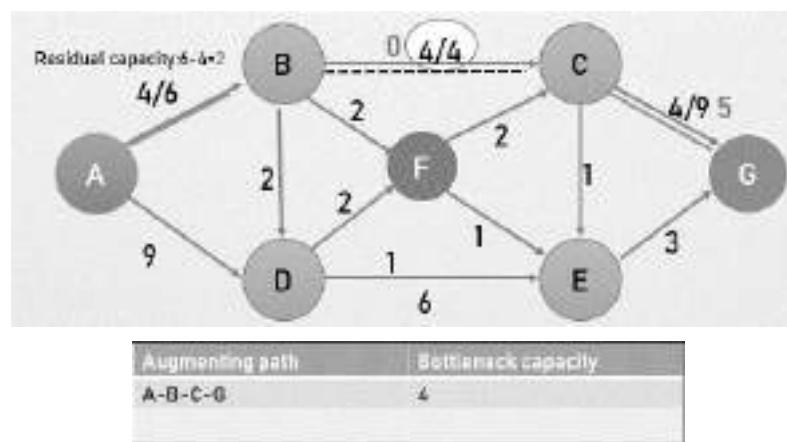


Example: Ford-Fulkerson Algorithm



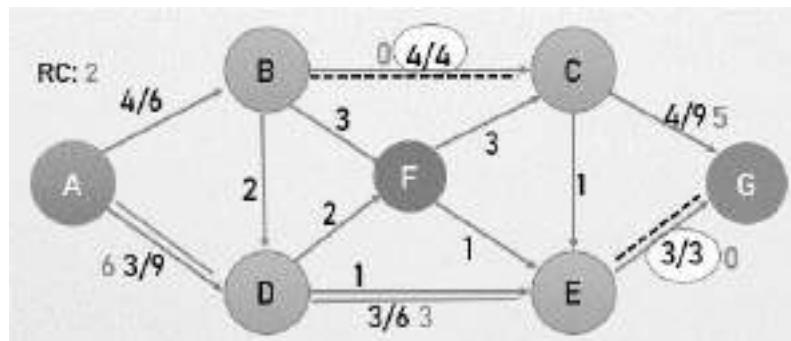
Path: A-B-C-G

Flow = 4



Path: A-D-E-G

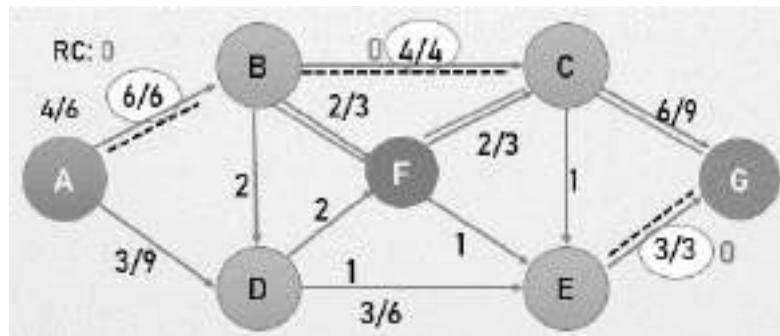
Flow = 4+3



Augmenting path	Bottleneck capacity
A-B-C-G	4
A-D-E-G	3

Path: A-B-F-C-G

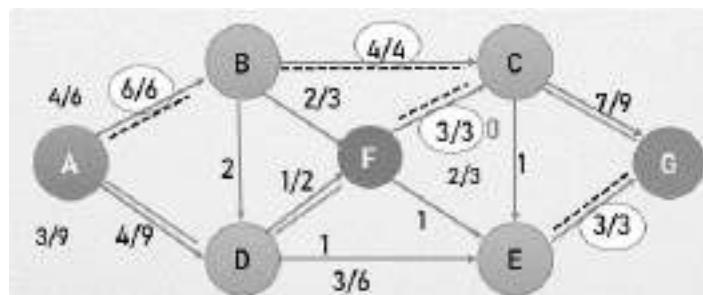
Flow = 4+3+2



Augmenting path	Bottleneck capacity
A-B-C-G	4
A-D-E-G	3
A-B-F-C-G	2

Path: A-D-F-C-G

Flow = 4+3+2+1 = 10



Augmenting path	Bottleneck capacity
A-B-C-G	4
A-D-E-G	3
A-B-F-C-G	2
A-D-F-C-G	1

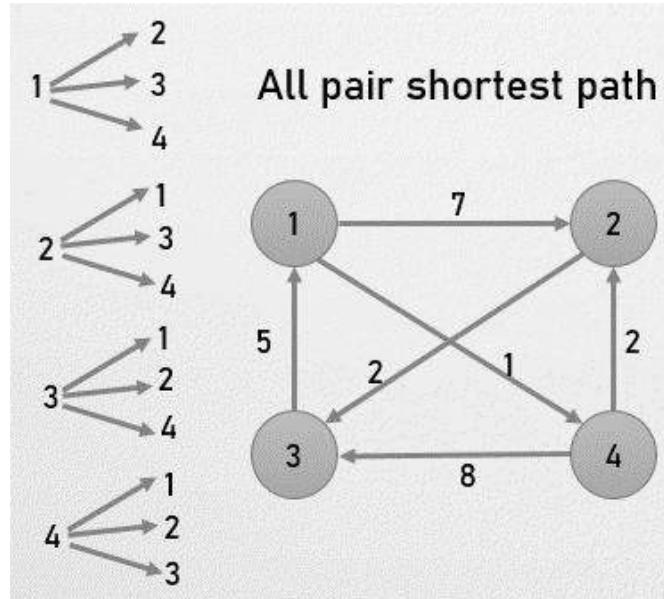
Ford-Fulkerson Applications

- Circulation with demands
- Water distribution pipeline
- Bipartite matching problem

11.5 Floyd-Warshall Algorithm

The Floyd Warshall Algorithm is used for solving the All Pairs Shortest Path problem. The problem is to finding the shortest path between all the pairs of vertices in a weighted directed Graph.

This algorithm works for both the directed and undirected weighted graphs. Floyd-Warshall algorithm follows the dynamic programming approach to find the shortest paths. Floyd-Warshall algorithm is also called as Floyd's algorithm, Roy-Floyd algorithm, Roy-Warshall algorithm, or WFI algorithm.



Floyd-Warshall Algorithm

$n = \text{no of vertices}$

$A = \text{matrix of dimension } n \times n$

for $k = 1$ to n

 for $i = 1$ to n

 for $j = 1$ to n

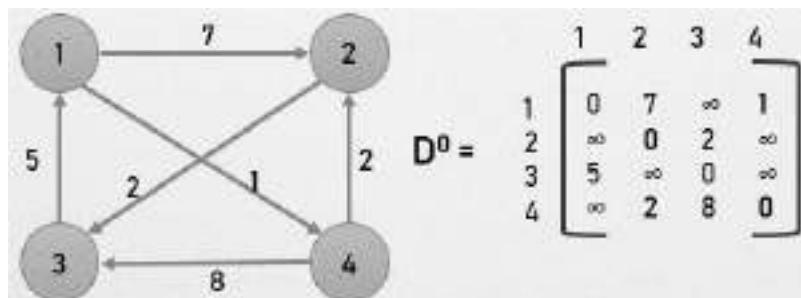
$A_{k[i, j]} = \min(A_{k-1}[i, j], A_{k-1}[i, k] + A_{k-1}[k, j])$

return A



Example: Floyd Warshall Algorithm

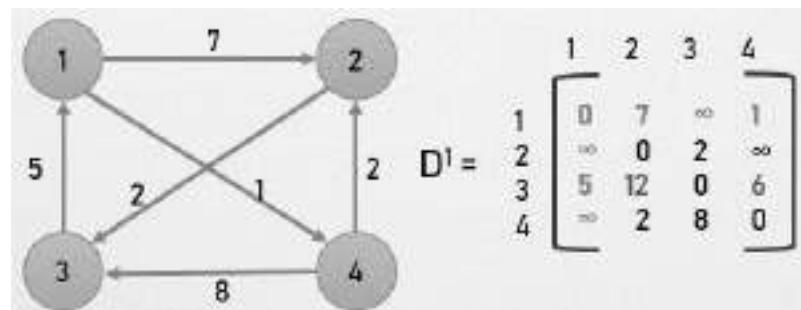
D1



$$D^1 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 0 & 7 & \infty & 1 \\ 2 & \infty & 0 & 2 & \infty \\ 3 & 5 & 12 & 0 & 6 \\ 4 & \infty & 2 & 8 & 0 \end{bmatrix}$$

2 to 3 = $(2-1),(1-3) == 2$
 2 to 4 = $(2-1),(1-4) == \infty$
 3 to 2 = $(3-1),(1-2) == 5+7 == 12$
 3 to 4 = $(3-1),(1-4) == 5+1 == 6$
 4 to 2 = $(4-1),(1-2) == 2$
 4 to 3 = $(4-1),(1-3) == 8$

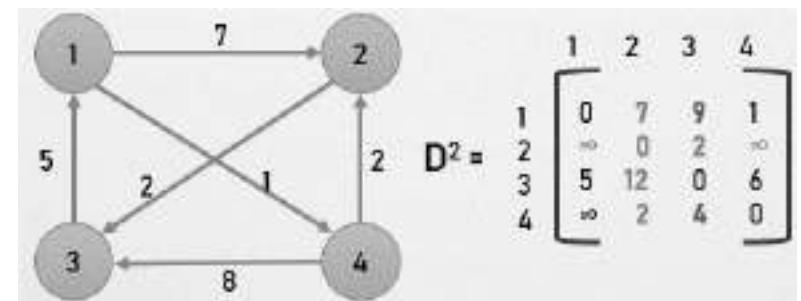
D2



$$D^2 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 0 & 7 & 9 & 1 \\ 2 & \infty & 0 & 2 & \infty \\ 3 & 5 & 12 & 0 & 6 \\ 4 & \infty & 2 & 4 & 0 \end{bmatrix}$$

1 to 3 = $(1-2),(2-3) == 7+2 == 9$
 1 to 4 = $(1-2),(2-4) == 1$
 3 to 1 = $(3-2),(2-1) == 5$
 3 to 4 = $(3-2),(2-4) == 6$
 4 to 1 = $(4-2),(2-4) == \infty$
 4 to 3 = $(4-2),(2-3) == 2+2 == 4$

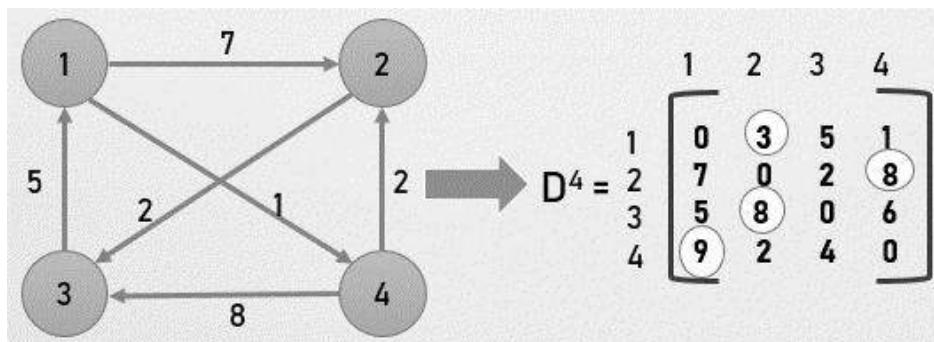
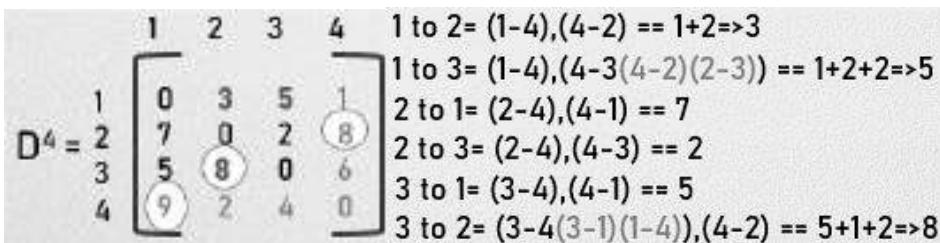
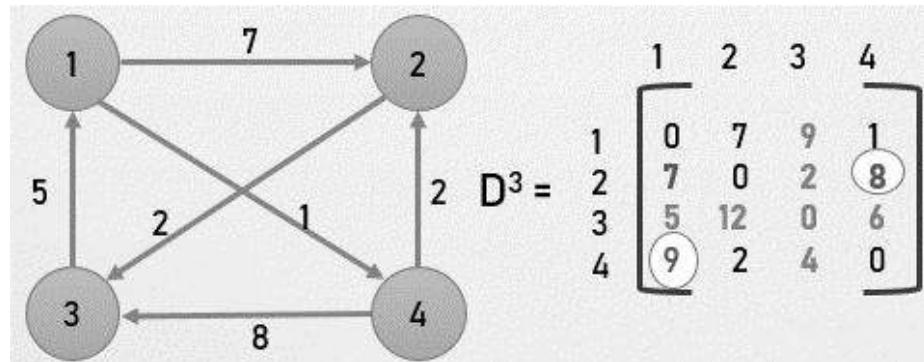
D3



$$D^3 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 0 & 7 & 9 & 1 \\ 2 & 7 & 0 & 2 & 8 \\ 3 & 5 & 12 & 0 & 6 \\ 4 & 9 & 2 & 4 & 0 \end{bmatrix}$$

1 to 2 = $(1-3),(3-2) == 7$
 1 to 4 = $(1-3),(3-4) == 1$
 2 to 1 = $(2-3),(3-1) == 7$
 2 to 4 = $(2-3),(3-4) == 2+5+1 == 8$
 4 to 1 = $(4-3),(3-1) == 2+5 == 9$
 4 to 2 = $(4-3),(3-2) == 2$

D4



All pair shortest path

Complexity

Time complexity = $O(|V|^3)$

Space complexity = $O(|V|^2)$

Applications: Floyd Warshall Algorithm

To find the transitive closure of directed graphs

To find the Inversion of real matrices

To find the shortest path in a directed graph

For testing whether an undirected graph is bipartite

11.6 Topological Sort

Topological Sort is a linear ordering of the vertices in such a way that if there is an edge in the DAG (directed acyclic graph) going from vertex 'u' to vertex 'v', then 'u' comes before 'v' in the ordering.

Topological Sorting is possible if and only if the graph is a Directed Acyclic Graph. There may exist multiple different topological orderings for a given directed acyclic graph.

Steps for topological sort

Step 1: Find the indegree for every vertex.

Step 2: Place the vertices whose indegree is 0 on the empty queue.

Step 3: Dequeue the vertex V and decrement the indegree's of all its adjacent vertices.

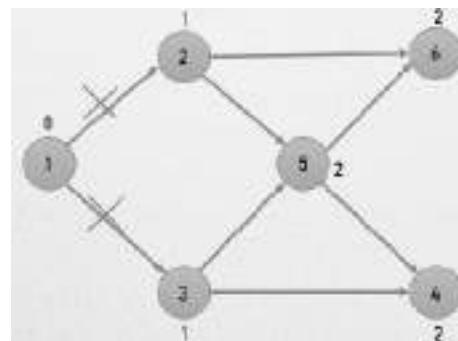
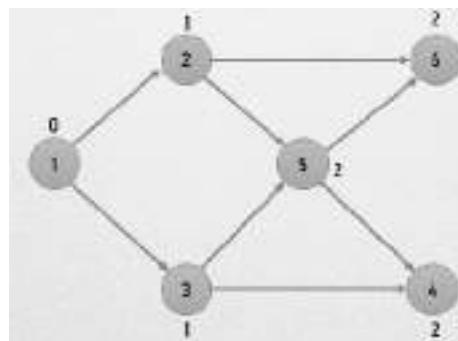
Step 4: Enqueue the vertex on the queue, if its degree falls to zero.

Step 5: Repeat from step 3 until the queue becomes empty.

Step 6: The topological ordering is the order in which the vertices dequeued.

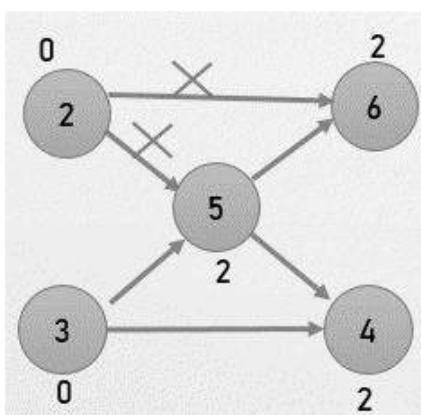


Example: Topological sort

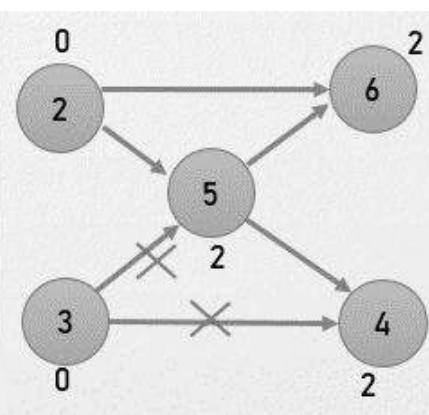


In-degree of vertex

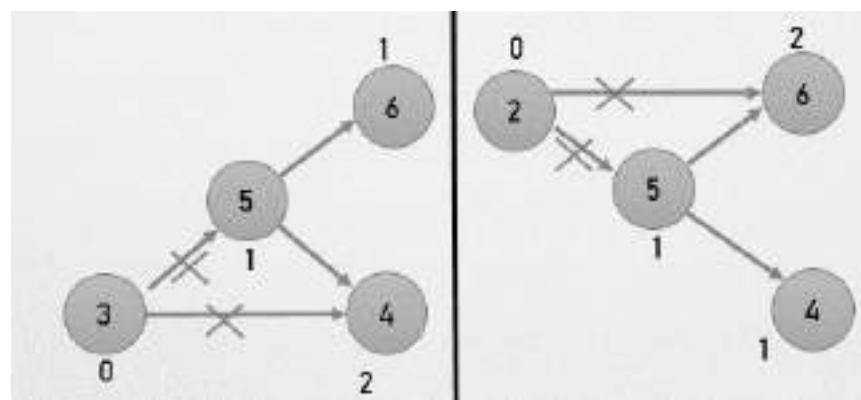
Visited vertex: 1



Visited vertex: 1 2

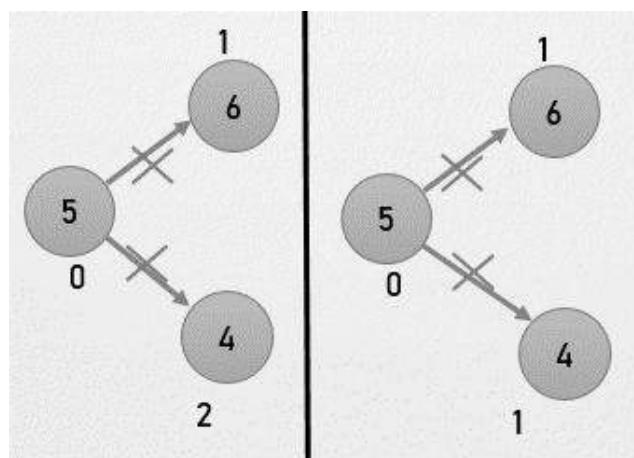


Visited vertex: 1 3



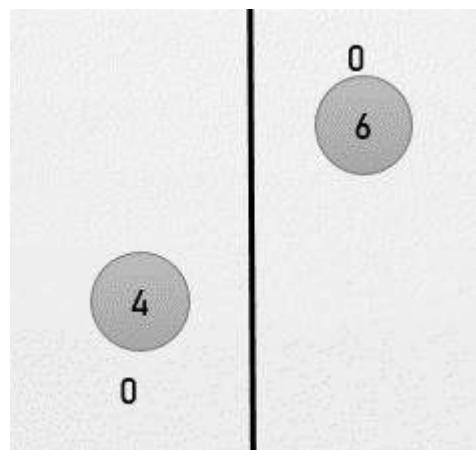
Visited vertex: 1 2 3

Visited vertex: 1 3 2



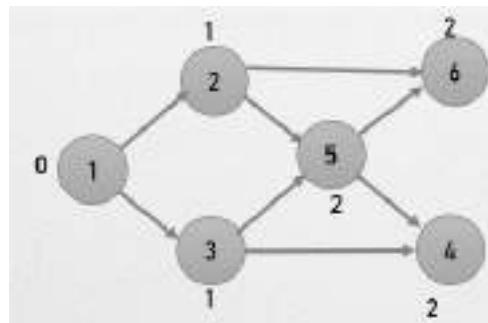
Visited vertex: 1 2 3 5

1



Visited vertex: 1 2 3 5 6 4

Visited vertex: 1 3 2 5 4 6



Visited vertex: 1 2 3 5 6 4

Visited vertex: 1 3 2 5 4 6

Visited vertex: 1 2 3 5 4 6

Visited vertex: 1 3 2 5 6 4

Applications of Topological Sort

Instruction Scheduling

Determining the order of compilation tasks to perform in make files

Scheduling jobs from the given dependencies among jobs

Data Serialization

Summary

- Graphs provide in excellent way to describe the essential features of many applications.
- Graphs are mathematical structures and are found to be useful in problem solving. They may be implemented in many ways by the use of different kinds of data structures.
- Graph traversals, Depth first as well as Breadth First, are also required in many applications.
- Breadth first search is a graph traversal algorithm that starts traversing the graph from root node and explores all the neighboring nodes.
- DFS traversal is a recursive algorithm for searching all the vertices/ nodes of a graph or tree using stack data structure.
- The Floyd Warshall Algorithm is used for solving the All Pairs Shortest Path problem

Keywords

Network Flow BFS

DFS Floyd Warshall Algorithm

Topological sort network simplex algorithm

Self Assessment

1. Which is not Graph traversal algorithm____

- Breadth First Search
- Depth First Search
- Euclidean algorithm
- Dijkstra's Algorithm

2. Time complexity of the BFS algorithm is ____
 - A. Log n
 - B. (V + E)
 - C. (log 1)
 - D. $\log n$

3. Breadth First Search algorithm use ____ data structure.
 - A. Stack
 - B. Queue
 - C. Linked list
 - D. All of above

4. Depth First Search applications are____
 - A. Path Finding.
 - B. Cycle detection in graphs.
 - C. Topological Sorting.
 - D. All of above

5. Space complexity of the BFS algorithm is ____
 - A. Log n
 - B. (V + E)
 - C. (V)
 - D. $\log n$

6. Breadth First Search algorithm use ____ data structure.
 - A. Queue
 - B. Linked list
 - C. Stack
 - D. All of above

7. Network flow Problems defines____
 - A. To find a flow with minimum value.
 - B. To find a flow with maximum value.
 - C. To find a flow with average value.
 - D. All of above

8. Which is not network flow problem____
 - A. Multi-commodity
 - B. Minimum-cost
 - C. Travelling salesman problem
 - D. Nowhere-zero

9. Algorithm used for network flow problem are ____

- A. Dinic's algorithm
- B. Edmonds-Karp algorithm
- C. Out-of-kilter
- D. All of above

10. Floyd Warshall Algorithm is used for ____

- A. To find a flow with average value.
- B. Travelling salesman problem
- C. All Pairs Shortest Path problem.
- D. All of above

11. Time complexity of Floyd-Warshall Algorithm ____

- A. $\log n$
- B. $O(|V|^3)$
- C. $O(|V|^2)$
- D. All of above

12. Space complexity of Floyd-Warshall Algorithm ____

- A. $(\log 2)$
- B. $\log 1$
- C. $O(|V|^3)$
- D. $O(|V|^2)$

13. Topological Sorting is possible if ____

- A. Graph is a Directed cyclic Graph
- B. Graph is a Directed Acyclic Graph
- C. Graph is an undirected Acyclic Graph.
- D. All of above

14. Applications of topological sort are ____

- A. Data Serialization
- B. Instruction Scheduling
- C. Scheduling jobs from the given dependencies among jobs
- D. All of above

15. The In-degree of starting vertex in topological sort graph is ____

- A. 1
- B. 0
- C. 2
- D. 4

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. C | 2. B | 3. B | 4. D | 5. C |
| 6. C | 7. B | 8. C | 9. D | 10. C |
| 11. B | 12. D | 13. B | 14. D | 15. B |

Review Questions

1. Discuss graph traversal with example.
2. Why queue and stack data structure is used with BFS and DFS?
3. Give an example of BFS with example.
4. Describe different types of network flow problem.
5. What are the applications of topological sort?
6. Discuss all pair shortest path problem.
7. Define Bottleneck capacity and Augmenting path.

**Further Readings**

Burkhard Monien, Data Structures and Efficient Algorithms, Thomas Ottmann, Springer.

Kruse, Data Structure & Program Design, Prentice Hall of India, New Delhi.

Mark Allen Weles, Data Structure & Algorithm Analysis in C, Second Ed., Addison-Wesley Publishing.

RG Dromey, How to Solve it by Computer, Cambridge University Press.

Lipschutz. S. (2011). Data Structures with C. Delhi: Tata McGraw hill

Reddy. P. (1999). Data Structures Using C. Bangalore: Sri Nandi Publications

Samantha. D (2009). Classic Data Structures. New Delhi: PHI Learning Private Limited

**Web Links**

www.en.wikipedia.org

www.web-source.net

https://www.brainkart.com/article/Topological-Sort_10158

<https://www.geeksforgeeks.org/topological-sorting>

<https://www.tutorialspoint.com/difference-between-bfs-and-dfs>

Unit 12: Hashing Techniques

CONTENTS

- Objectives
- Introduction
- 12.1 Hashing
- 12.2 Steps to Implement Hashing
- 12.3 Hash Table
- 12.4 Hash Function
- 12.5 Division Method
- 12.6 Mid Square Method
- 12.7 Digit Folding Method
- 12.8 Multiplication Method
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Question
- Further Readings

Objectives

After studying this unit, you will be able to:

- Learn basics of hashing
- Understand linear list representation
- Learn hash table representation
- Discuss hash functions and its methods

Introduction

The search time of each algorithm depend on the number n of elements of the collection S of the data. A searching technique called Hashing or Hash addressing which is essentially independent of the number n .

Hashing is the transformation of a string of characters into a usually shorter fixed-length value or key that represents the original string. Hashing is used to index and retrieve items in a database because it is faster to find the item using the shorter hashed key than to find it using the original value. It is also used in many encryption algorithms.

A Hash Function is a Unary Function that is used by Hashed Associative Containers: it maps its argument to a result of type sized. A Hash Function must be deterministic and stateless. That is, the return value must depend only on the argument, and equal arguments must yield equal results.

12.1 Hashing

In many applications we require to use a data object called symbol table. A symbol table is nothing but a set of pairs (name, value). Where value represents collection of attributes associated with

thename, and this collection of attributes depends upon the program element identified by the name. For example, if a name x is used to identify an array in a program, then the attributes associated with x are the number of dimensions, lower bound and upper bound of each dimension, and the element type. Therefore a symbol table can be thought of as a linear list of pairs (name, value), and hence you can use a list of data object for realizing a symbol table. A symbol table is referred to or accessed frequently either for adding the name, or for storing the attributes of the name, or for retrieving the attributes of the name. Therefore, accessing efficiency is a prime concern while designing a symbol table. Hence the most common way of getting a symbol table implemented is to use a hash table. Hashing is a method of directly computing the index of the table by using some suitable mathematical function called hash function. The hash function operates on the name to be stored in the symbol table, or whose attributes are to be retrieved from the symbol table. If h is a hash function and x is a name, then $h(x)$ gives the index of the table where x along with its attributes can be stored. If x is already stored in the table, then $h(x)$ gives the index of the table where it is stored to retrieve the attributes of x from the table. There are various methods of defining a hash function like a division method. In this method, you take the sum of the values of the characters, divide it by the size of the table, and take the remainder. This gives us an integer value lying in the range of 0 to $(n - 1)$ if the size of the table is n . The other method is a mid-square method. In this method, the identifier is first squared and then the appropriate number of bits from the middle of square is used as the hash value. Since the middle bits of the square usually depend on all the characters in the identifier, it is expected that different identifiers will result into different values. The number of middle bits that you select depends on the table size. Therefore if r is the number of middle bits that you use to form hash value, then the table size will be 2^r . Hence when you use this method the table size is required to be power of 2. Another method is folding in which the identifier is partitioned into several parts, all but the last part being of the same length. These parts are then added together to obtain the hash value.

To store the name or to add attributes of the name, you compute hash value of the name, and place the name or attributes as the case may be, at that place in the table whose index is the hash value of the name. For retrieving the attribute values of the name kept in the symbol table, I apply the hash function to the name to obtain index of the table where you get the attributes of the name. Hence you find that no comparisons are required to be done. Hence the time required for the retrieval is independent of the table size. Therefore, retrieval is possible in a constant amount of time, which will be the time taken for computing the hash function. Therefore, hash table seems to be the best for realization of the symbol table, but there is one problem associated with the hashing, and it is of collisions. Hash collision occurs when the two identifiers are mapped into the same hash value. This happens because a hash function defines a mapping from a set of valid identifiers to the set of those integers, which are used as indices of the table. Therefore, you see that the domain of the mapping defined by the hash function is much larger than the range of the mapping, and hence the mapping is of many to one nature. Therefore, when I implement a hashtable a suitable collision handling mechanism is to be provided which will be activated when there is a collision.

Collision handling involves finding out an alternative location for one of the two colliding symbols. For example, if x and y are the different identifiers and if $h(x) = h(y)$, x and y are the colliding symbols. If x is encountered before y , then the i th entry of the table will be used for accommodating symbol x , but later on when y comes there is a hash collision, and therefore you have to find out an alternative location either for x or y . This means you find out a suitable alternative location and either accommodate y in that location, or you can move x to that location and place y in the i th location of the table. There are various methods available to obtain an alternative location to handle the collision. They differ from each other in the way search is made for an alternative location. The following are the commonly used collision handling techniques.

Terminology: Hash table

Data bucket - Data buckets are memory locations where the records are stored. It is also known as unit of storage.

Hash index - It is an address of the data block. A hash function could be a simple mathematical function to even a complex mathematical function.

Linear Probing - Linear probing is a fixed interval between probes. In this method, the next available data block is used to enter the new record, instead of overwriting on the older record.

Double Hashing -Double hashing is a computer programming method used in hash tables to resolve the issues of has a collision.

Quadratic probing- It helps you to determine the new bucket address. It helps you to add Interval between probes by adding the consecutive output of quadratic polynomial to starting value given by the original computation.

Time complexity

Time complexity in linear search is $O(n)$

Time complexity in binary search is $O(\log n)$

Time complexity in hashing is $O(1)$

12.2 Steps to Implement Hashing

An element is converted into an integer by using a hash function. This element can be used as an index to store the original element, which falls into the hash table.

The element is stored in the hash table where it can be quickly retrieved using hashed key.

hash = hash func(key)

index = hash % array size

The hash is independent of the array size and it is then reduced to an index (a number between 0 and array size - 1) by using the modulo operator (%).

Operations of a hash table

Search – Searches an element in a hash table.

Insert – inserts an element in a hash table.

Delete – Deletes an element from a hash table.

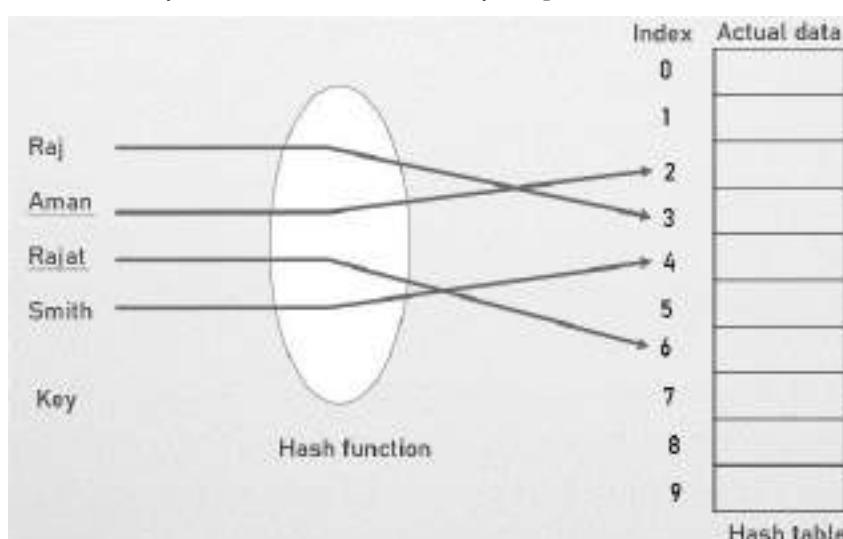
12.3 Hash Table

A Hash table is a data structure that stores information, and the information has basically two components.

- key and value.

The hash table can be implemented with the help of an associative array

It uses a hash function to compute an index into an array of buckets or slots from which the desired value can be found. It is an array of list where each list is known as bucket. It contains value based on the key. Hash table is synchronized and contains only unique elements.



12.4 Hash Function

A hash function is any function that can be used to map a data set of an arbitrary size to a data set of a fixed size, which falls into the hash table. The values returned by a hash function are called hash values, hash codes, hash sums, or hashes. An efficient hash function should be built such that the index value of the added item is distributed equally across the table.

An effective collision resolution technique should be created to generate an alternate index for a key whose hash index corresponds to a previously inserted position in a hash table.

Characteristics of hash function

Uniform Distribution: For distribution throughout the constructed table.

Fast: The generation of hash should be very fast, and should not produce any considerable overhead.

Less collisions: Collisions occur when pairs of elements are mapped to the same hash value. These should be avoided.

Some of the methods of defining hash function are discussed below:

1. **Modular arithmetic:** In this method, first the key is converted to integer, then it is divided by the size of index range, and the remainder is taken to be the hash value. The spread achieved depends very much on the modulus. If modulus is power of small integers like 2 or 10, then many keys tend to map into the same index, while other indices remain unused. The best choice for modulus is often but not always a prime number, which usually has the effect of spreading the keys quite uniformly.

2. **Truncation:** This method ignores part of key, and use the remainder part directly as hash value. (considering non-numeric fields as their numerical code) If the keys for example are eight digit numbers and the hash table has 1000 entries, then the first, second, and fifth digit from right might make hash value. So 62538194 maps to 394. It is a fast method, but often fails to distribute keys evenly.

3. **Folding:** In this method, the identifier is partitioned into several parts all but the last part being of the same length. These parts are then added together to obtain the hash value. For example an eight digit integer can be divided into groups of three, three, and two digits. The groups are then added together, and truncated if necessary to be in the proper range of indices. Hence 62538149 maps to, $625 + 381 + 94 = 1100$, truncated to 100. Since all information in the key can affect the value of the function, folding often achieves a better spread of indices than truncation.

4. **Mid square method:** In this method, the identifier is squared (considering non-numeric fields as their numerical code), and then the appropriate number of bits from the middle depend on all the characters in the identifier, it is expected that different identifiers will result in different values. The number of middle bits that we select depends on table size. Therefore if r is the number of middle bits used to form hash value, then the table size will be 2^r , hence when you use mid square method the table size should be a power of 2.

12.5 Division Method

The hash function can be described as the hash function divides the value k by M and then uses the remainder obtained. The division method involves mapping a key k into one of m slots by taking the remainder of k divided by m as expressed in the hash function.

$$h(K) = k \bmod M$$

Here,

k is the key value, and

M is the size of the hash table.



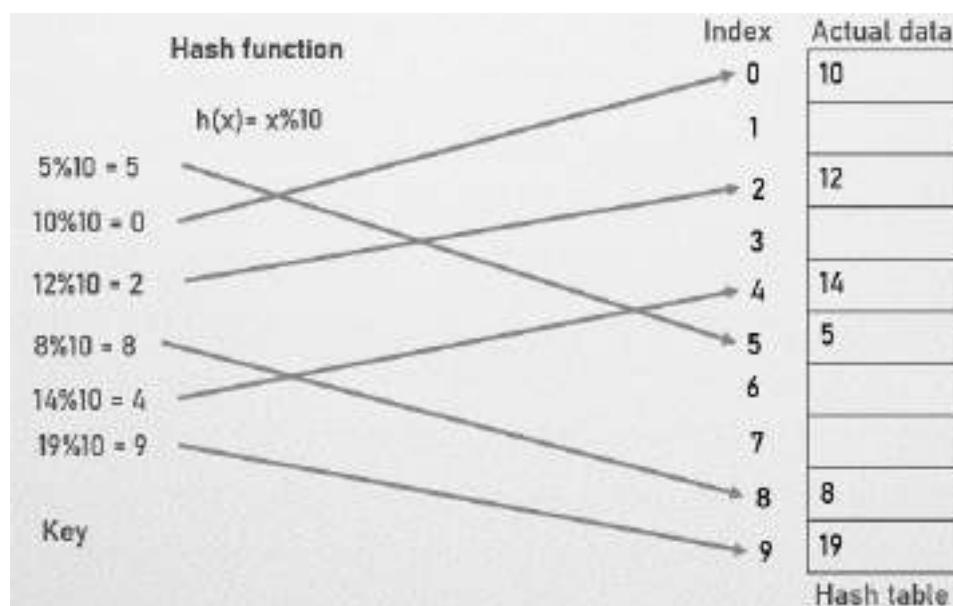
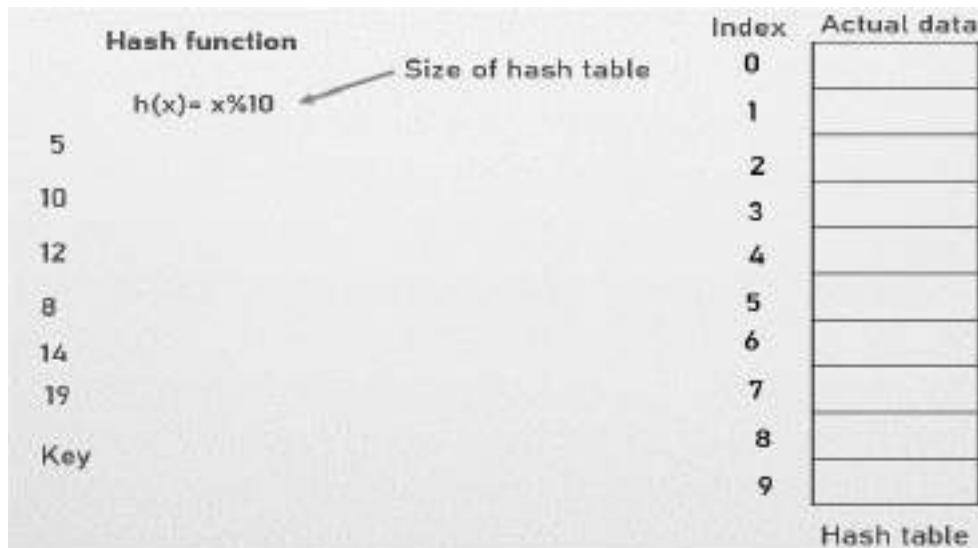
Example:

$m=30; k=80$

$$h(k) = k \bmod m = 20$$



Example:



Division method: pros

Any key will indeed map to one of m slots (as we want from a hash function, mapping a key to one of m slots)

Fast and simple

Division method: cons

Need to avoid certain values of m to avoid bias (as in the even number example)

12.6 Mid Square Method

In this technique, squares the key value. Then, some digits from the middle are extracted. These extracted digits form a number which is taken as the new number for address.

Limitation:

The size of key2 is too large.

E.g. $2025^2 = 4100625$



Example:

$H(key) = key^2$		Index	Actual data
		0	10
10	$10 \times 10 = 100$	1	
15		2	
12		3	
22		4	
14		5	
		6	
		7	
		8	
		9	
			Hash table

$H(key) = key^2$		Index	Actual data
		0	10
10	$10 \times 10 = 100$	1	
15	$15 \times 15 = 225$	2	15
12	$12 \times 12 = 144$	4	12
22	$22 \times 22 = 484$	5	
14	$14 \times 14 = 196$	7	
		8	22
		9	14
			Hash table

12.7 Digit Folding Method

The folding method for constructing hash functions begins by dividing the item into equal-size pieces (the last piece may not be of equal size). These pieces are then added together to give the resulting hash value.

Fold shift

Fold boundary

Unit 12: Hashing Techniques

-The left and right numbers are folded on fixed boundary between them and the centre.

-The two outside values are then reversed.

Case 1: if hash table size is 100 (0-99) and sum in 3 digits, then we will ignore last carry, if any.

Or

Case 2: if hash table size is 100 (0-99) and sum in 3 digits, then we need to perform the extra step of dividing by 100(size of table) and keeping the remainder.

Folding method: case 1

Key:	122122	Index	Actual data
Parts:	12 21 22	0	
Sum:	55	1	
Hash value:	55	55	122122
		
		99	
			Hash table

Folding method: case 2

Key:	234567	Index	Actual data
Parts:	23 45 67	0	
Sum:	135	1	
Ignore last carry		35	234567
Hash value:	35	
		99	
			Hash table

Folding method: case 3

Key:	234567	Index	Actual data
Parts:	23 45 67	0	
		1	
		
Sum: 135 mod 100 = 35		35	234567
Sum mod with table size			
Hash value: 35			
		
		99	
			Hash table

Fold boundary: case 1

Key:	142123	Index	Actual data
Parts:	14 21 23	0	
	41 21 32	1	
		
Sum: 94		94	142123
Hash value: 58			
		
		99	
			Hash table

Fold boundary: case 2

Key:	354027	Index	Actual data
Parts:	35 40 27	0	
	53 40 72 value reversed	1	
Sum:	165	
Ignore last carry		65	354027
Hash value:	65	
		99	
			Hash table

12.8 Multiplication Method

$h(k) = \text{floor}(\text{table size} * (\text{key} * A)) \% \text{size}$

Or

$$h(k) = \text{floor}(n(kA \bmod 1))$$

K= key

A is constant value between 0 and 1

Knuth recommends $A = 0.6180339887\dots$ (Golden Ratio)

1) Choose constant

2) Multiply key k by A

3) Extract fractional part of kA (this gives us a number between 0 and 1)

4) Multiply fractional part by m and take floor of the multiplication (this transforms a number between 0 and 1, to a discrete number between 0 and $m-1$ that we can map to slot in the hash table)



Example: Multiplication Method

$k=34$

Size of table = 100

$A=0.618033$

$$h(34) = \text{floor}(100(34 * 0.618033)) \% 100$$

$$= \text{floor}(3461.80) \% 100$$

$$= 3461 \% 100$$

$$= 61$$

Pros: Indeed maps any key into 1 of m slots, as we expect from a hash function

Choice of m not as critical as in division method

There is also a bit implementation (not discussed)

Cons:

Slower than division method

Summary

- Hash functions are mostly used in hash tables, to quickly locate a data record (for example, a dictionary definition) given its search key (the headword).
- Specifically, the hash function is used to map the search key to the index of a slot in the table where the corresponding record is supposedly stored.
- A hash function is any function that can be used to map a data set of an arbitrary size to a data set of a fixed size, which falls into the hash table.
- The folding method for constructing hash functions begins by dividing the item into equal-size pieces (the last piece may not be of equal size).

Keywords

Hash function	Division method
Mid square method	Fold boundary
Hash table	Hashes
Modular arithmetic	

Self Assessment

1. Time complexity in hashing is ____
 - (log o)
 - (log n)
 - (1)
 - None of above
2. The values returned by a hash function are called ____
 - Hash values
 - Hash codes
 - Hash sums
 - All of above
3. Methods for calculating the hash function are ____
 - Division method
 - Folding method
 - Mid square method
 - All of above
4. Hashing is process of ____
 - Encrypt data
 - Converting an input of any length into a fixed size string
 - Calculate mean of number
 - None of above

5. Hash table components are_

- A. Key
- B. Value
- C. Both key and value
- D. None of above

6. Which operator is used in hashing ____

- A. +
- B. *
- C. %
- D. /

7. Which is not characteristics of hash function ____

- A. Less collisions
- B. Uniform Distribution
- C. Static
- D. All of above

8. Which is not a methods for calculating the hash function ____

- A. Folding
- B. Mid square
- C. Square
- D. Division

9. $(h(x)= x \% 10)$, 10 represent in statement __

- A. Size of hash function
- B. Size of hash key
- C. Size of hash table
- D. None of above

10. $(h(x)= x \% 10)$, x represent in statement _

- A. Key value
- B. Hash table size
- C. Hash function
- D. None of above

11. Fold shift and Fold boundary methods used in ____

- A. Division method
- B. Mid square method
- C. Multiplication Method
- D. None of above

12. $h(k) = \text{floor}(n(kA \bmod 1))$, statement represent which method ____

- A. Division method
- B. Multiplication Method
- C. Mid square method
- D. Pairing method

13. Knuth recommends value for A is ____

- A. 0.61803
- B. 0.62347
- C. 0.71803
- D. None of above

14. Golden ratio is recommended by __

- A. Smith K
- B. Knuth
- C. George S
- D. None of above

15. In multiplication method the value of A is between __

- A. 2 and 4
- B. 2 and 3
- C. 0 and 1
- D. None of above

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. C | 2. D | 3. D | 4. B | 5. C |
| 6. C | 7. C | 8. C | 9. C | 10. A |
| 11. D | 12. B | 13. A | 14. B | 15. C |

Review Question

1. Discuss hashing.
2. What is the significance of hashing in data structure?
3. Define hash function with suitable example.
4. Give an example mid square method.
5. Differentiate between division method and multiplication method.
6. Define Linear Probing and data bucket.



Further Readings

Burkhard Monien, Data Structures and Efficient Algorithms, Thomas Ottmann,

Springer.

Kruse, Data Structure & Program Design, Prentice Hall of India, New Delhi.

Mark Allen Weles, Data Structure & Algorithm Analysis in C, Second Ed., Addison-Wesley Publishing.

RG Dromey, How to Solve it by Computer, Cambridge University Press.

Lipschutz. S. (2011). Data Structures with C. Delhi: Tata McGraw hill

Reddy. P. (1999). Data Structures Using C. Bangalore: Sri Nandi Publications

Samantha. D (2009). Classic Data Structures. New Delhi: PHI Learning Private Limited



Web Links

www.en.wikipedia.org

www.web-source.net

<https://www.tutorialspoint.com/Hash-Functions-and-Hash-Tables>

<https://www.ee.ryerson.ca/~courses/coe428/structures/hash.html>

<https://www.techopedia.com/definition/19744/hash-function>

<https://www.cs.hmc.edu/~geoff/classes/hmc.cs070.200101/homework10/hashfuncs.html>

Unit 13: Collision Resolution

CONTENTS

- Objectives
- Introduction
- 13.1 Collision Resolution
- 13.2 Separate Chaining
- 13.3 Open Addressing
- 13.4 Linear Probing
- 13.5 Quadratic Probing
- Summary
- Keywords
- Self Assessment
- Answers for Self Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to:

- Discuss separate chaining
- Understand open addressing-linear probing
- Learn quadratic probing

Introduction

The implementation of hash tables is frequently called hashing. Hashing is a technique used for performing insertions, deletions, and finds in constant average time. Tree operations that require any ordering information among the elements are not supported efficiently. Thus, operations such as findMin, findMax, and the printing of the entire table in sorted order in linear time are not supported.

Note that straightforward hashing is not without its problems, because for almost all hash functions, more than one key can be assigned to the same position. For example, if the hash function h_1 applied to names returns the ASCII value of the first letter of each name (i.e., $h_1(\text{name}) = \text{name}[0]$), then all names starting with the same letter are hashed to the same position. This problem can be solved by finding a function that distributes names more uniformly in the table. For example, the function h_2 could add the first two letters (i.e., $h_2(\text{name}) = \text{name}[0] + \text{name}[1]$), which is better than h_1 . But even if all the letters are considered (i.e., $h_3(\text{name}) = \text{name}[0] + \dots + \text{name}[\text{strlen(name)} - 1]$), the possibility of hashing different names to the same location still exists. The function h_3 is the best of the three because it distributes the names most uniformly for the three defined functions, but it also tacitly assumes that the size of the table has been increased. If the table has only 26 positions, which is the number of different values returned by h_1 , there is no improvement using h_3 instead of h_1 . Therefore, one more factor can contribute to avoiding conflicts between hashed keys, namely, the size of the table. Increasing this size may lead to better hashing, but not always! These two factors –hash function and table size– may minimize the number of collisions, but they cannot completely eliminate them. The problem of collision has to be dealt with in a way that always guarantees a solution.

13.1 Collision Resolution

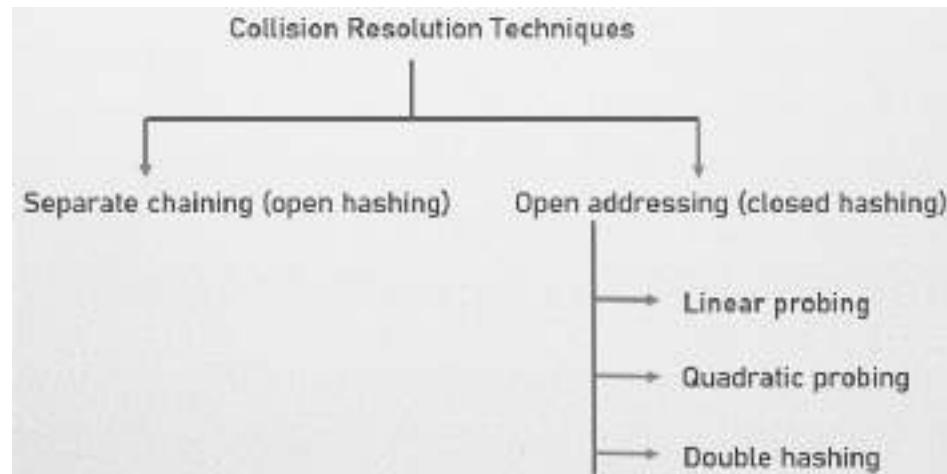
When two keys or hash values compete with a single hash table slot, then Collision occur. To resolve collision we use collision resolution techniques. Collisions can be reduced with a selection of a good hash function.

Collision Resolution Techniques

There are two types of collision resolution techniques.

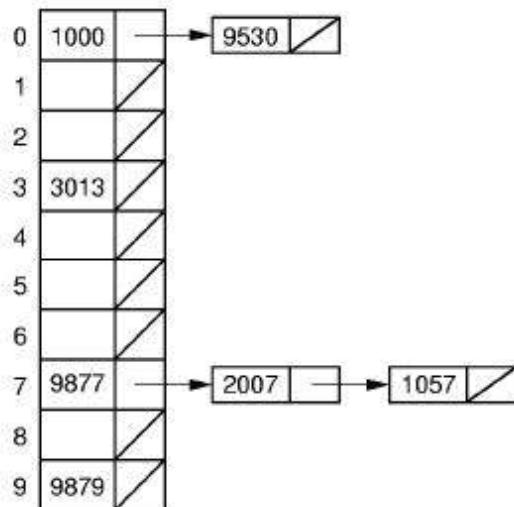
Separate chaining (open hashing)

Open addressing (closed hashing)



Open Hashing

The simplest form of open hashing defines each slot in the hash table to be the head of a linked list. All records that hash to a particular slot are placed on that slot's linked list. The figure below illustrates a hash table where each slot stores one record and a link pointer to the rest of the list.



Records within a slot's list can be ordered in several ways: by insertion order, by key value order, or by frequency-of-access order. Ordering the list by key value provides an advantage in the case of an unsuccessful search, because I know to stop searching the list once you encounter a key that is greater than the one being searched for. If records on the list are unordered or ordered by frequency, then an unsuccessful search will need to visit every record on the list.

Given a table of size M storing N records, the hash function will (ideally) spread the records evenly among the M positions in the table, yielding on average N/M records for each list. Assuming that the table has more slots than there are records to be stored, you can hope that few slots will contain

more than one record. In the case where a list is empty or has only one record, a search requires only one access to the list. Thus, the average cost for hashing should be $\Theta(1)$. However, if clustering causes many records to hash to only a few of the slots, then the cost to access a record will be much higher because many elements on the linked list must be searched.

Open hashing is most appropriate when the hash table is kept in main memory, with the lists implemented by a standard in-memory linked list. Storing an open hash table on disk in an efficient way is difficult, because members of a given linked list might be stored on different disk blocks. This would result in multiple disk accesses when searching for a particular key value, which defeats the purpose of using hashing.

Let:

1. U be the universe of keys:

(a) Integers

(b) Character strings

(c) Complex bit patterns

2. B the set of hash values (also called the buckets or bins). Let $B = \{0, 1, \dots, m - 1\}$ where $m > 0$ is a positive integer.

A hash function $h: U \rightarrow B$ associates buckets (hash values) to keys.

Two main issues:

Collisions

If x_1 and x_2 are two different keys, it is possible that $h(x_1) = h(x_2)$. This is called a collision. Collision resolution is the most important issue in hash table implementations.

Hash Functions

Choosing a hash function that minimizes the number of collisions and also hashes uniformly is another critical issue.

Closed Hashing

1. All elements are stored in the hash table itself

2. Avoids pointers; only computes the sequence of slots to be examined.

3. Collisions are handled by generating a sequence of rehash values.

$h: U \times U \rightarrow \{0, 1, 2, \dots, m - 1\}$

universe of primary keys probe number

4. Given a key x , it has a hash value $h(x, 0)$ and a set of rehash values

$h(x, 1), h(x, 2), \dots, h(x, m-1)$

5. I require that for every key x , the probe sequence

$\langle h(x, 0), h(x, 1), h(x, 2), \dots, h(x, m-1) \rangle$

be a permutation of $\langle 0, 1, \dots, m-1 \rangle$.

This ensures that every hash table position is eventually considered as a slot for storing a record with a key value x .

Search (x, T)

Search will continue until you find the element x (successful search) or an empty slot (unsuccessful search).

Delete (x, T)

1. No delete if the search is unsuccessful.

2. If the search is successful, then put the label DELETED (different from an empty slot).

Insert (x, T)

1. No need to insert if the search is successful.
2. If the search is unsuccessful, insert at the first position with a DELETED tag.

13.2 Separate Chaining

In this technique, a linked list is created from the slot in which collision has occurred, after which the new key is inserted into the linked list. This linked list of slots looks like a chain, so it is called separate chaining.

Separate chaining, is to keep a list of all elements that hash to the same value. We can use the Standard Library list implementation. If space is tight, it might be preferable to avoid their use (since these lists are doubly linked and waste space).

Performance of Chaining

Load factor $\alpha = n/m$

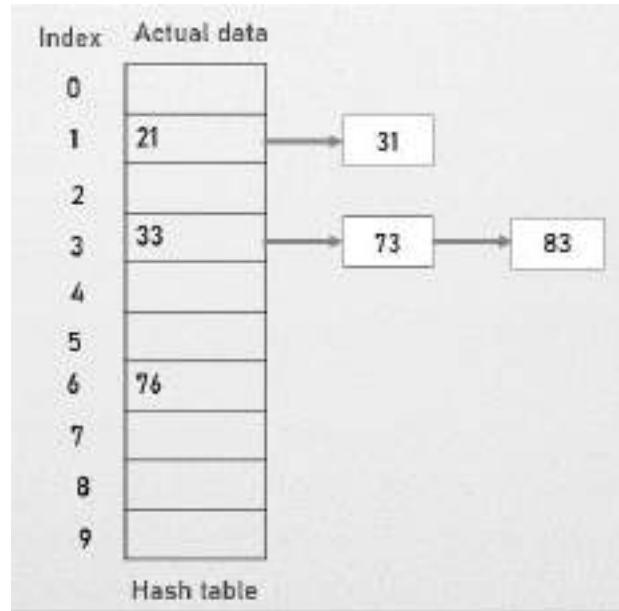
m = Number of slots in hash table

n = Number of keys to be inserted in hash table

Expected time to search = $O(1 + \alpha)$

Expected time to delete = $O(1 + \alpha)$

Time to insert = $O(1)$



Time complexity

For Searching

In worst case, all the keys might map to the same bucket of the hash table.

In such a case, all the keys will be present in a single linked list.

Sequential search will have to be performed on the linked list to perform the search.

Worst case complexity for searching is $O(n)$.

For Deletion

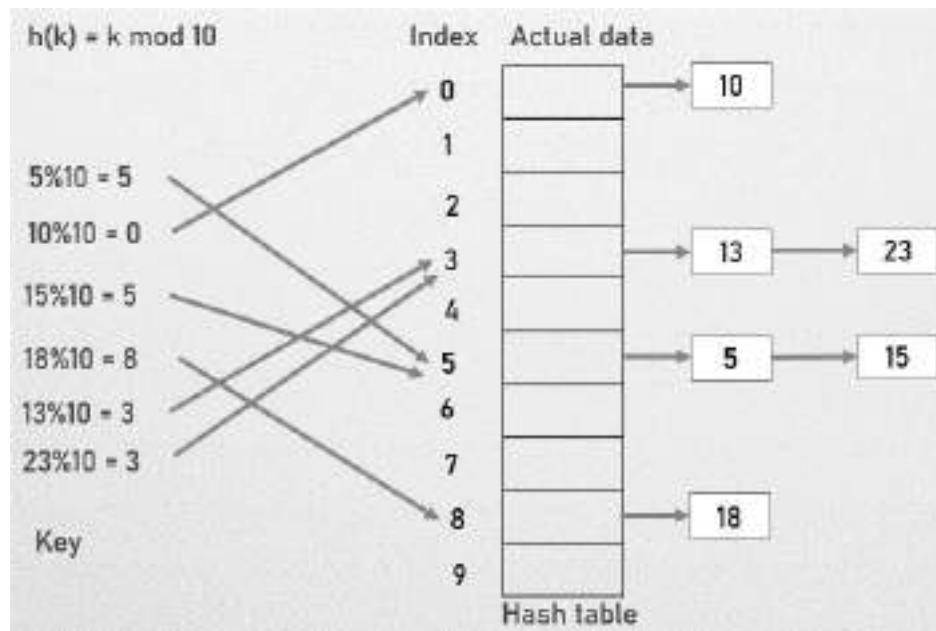
In worst case, the key might have to be searched first and then deleted.

In worst case, time taken for searching is $O(n)$.

Worst case complexity for deletion is $O(n)$.



Example: Separate chaining



Advantages of separate chaining

It is easy to implement.

The hash table never fills full, so we can add more elements to the chain.

It is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.

It is less sensitive to the function of the hashing.

Disadvantages of separate chaining

The cache performance of chaining is not good.

The memory wastage is too much in this method.

It requires more space for element links.

If the chain becomes long, then search time can become $O(n)$ in the worst case.

13.3 Open Addressing

The open addressing technique requires a hash table with fixed and known size. All elements are stored in the hash table itself. The size of the table must be greater than or equal to the total number of keys. During insertion, if a collision is encountered, alternative cells are tried until an empty bucket is found.

In case of collision:

Probing is performed until an empty bucket is found.

Once an empty bucket is found, the key is inserted.

Probing is performed in accordance with the technique used for open addressing.

Operations in Open Addressing

Insert (k): Keep probing until an empty slot is found. Once an empty slot is found, insert k.

Search (k): Keep probing until slot's key doesn't become equal to k or an empty slot is reached.

Delete: The key is first searched and then deleted. After deleting the key, that particular bucket is marked as "deleted".

Closed Hashing methods:

- Linear Probing
- Quadratic probing
- Double hashing

13.4 Linear Probing

In linear probing fixed sized hash table is used and when hash collision situation occur then, we linearly traverse the table in a cyclic manner to find the next empty slot. In this approach searches are performed sequentially so it's known as linear probing.

In this, when the collision occurs, we perform a linear probe for the next slot, and this probing is performed until an empty slot is found. In linear probing, the worst time to search for an element is $O(\text{table size})$. The linear probing gives the best performance of the cache but its problem is clustering. The main advantage of this technique is that it can be easily calculated.

Let $\text{hash}(x)$ be the slot index computed using a hash function and n be the table size

If slot $\text{hash}(x) \% n$ is full, then we try $(\text{hash}(x) + 1) \% n$

If $(\text{hash}(x) + 1) \% n$ is also full, then we try

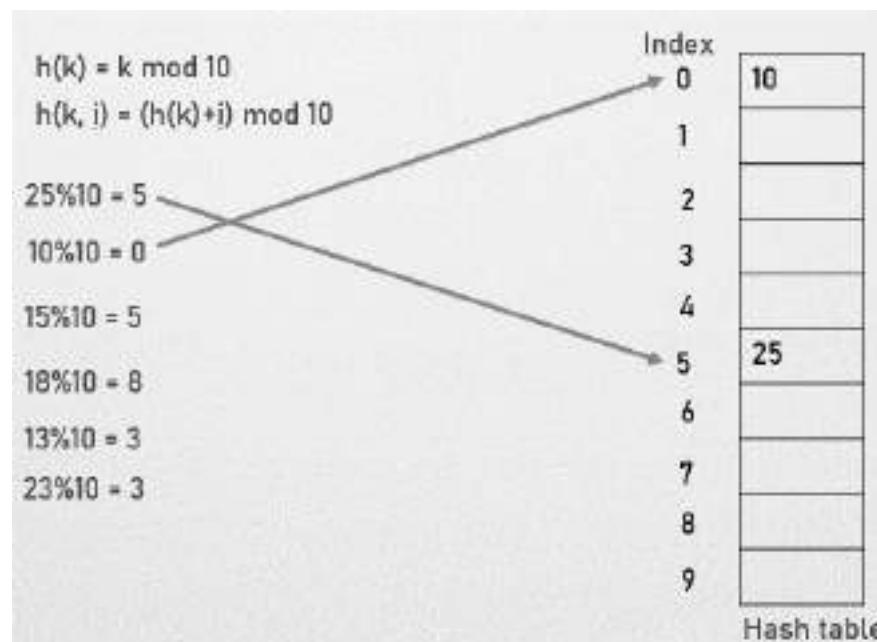
$(\text{hash}(x) + 2) \% n$

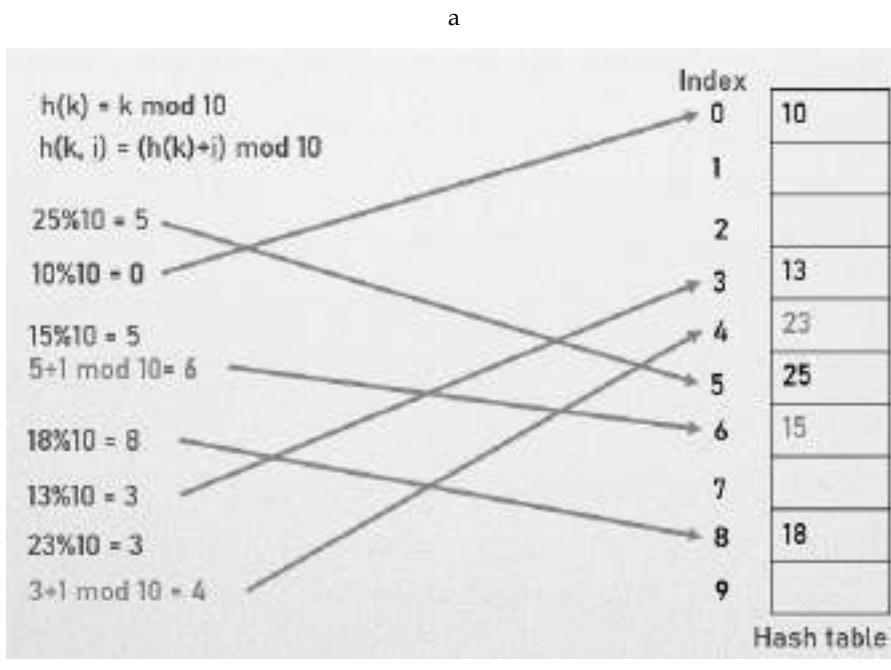
If $(\text{hash}(x) + 2) \% n$ is also full, then we try

$(\text{hash}(x) + 3) \% n$ so on...



Example: Linear Probing





Advantage

It is easy to compute.

Disadvantage

The clustering is major problem with linear probing

Many consecutive elements form groups.

It takes too much time to find an empty slot.

Time complexity

Worst time to search for an element is O(table size).

13.5 Quadratic Probing

Quadratic probing is a collision resolution method that eliminates the primary clustering problem of linear probing. Quadratic probing is what you would expect – the collision function is quadratic.

It is an open-addressing scheme. Here we look for i^2 slot in i th iteration if the given hash value x collides in the hash table. It is used to eliminate the primary clustering problem of linear probing.

In quadratic probing the sequence is that $H+1^2, H+2^2, H+3^2, \dots, H+K^2$

The hash function for quadratic probing is

$$h_i(X) = (\text{Hash}(X) + F(i)^2) \% \text{Table Size} \text{ for } i = 0, 1, 2, 3, \dots \text{etc.}$$

If the slot $\text{hash}(x) \% S$ is full, then we try

$$(\text{hash}(x) + 1*1) \% S.$$

If $(\text{hash}(x) + 1*1) \% S$ is also full, then we try $(\text{hash}(x) + 2*2) \% S$.

If $(\text{hash}(x) + 2*2) \% S$ is also full, then we try $(\text{hash}(x) + 3*3) \% S$.

This process continues until an empty slot is found.

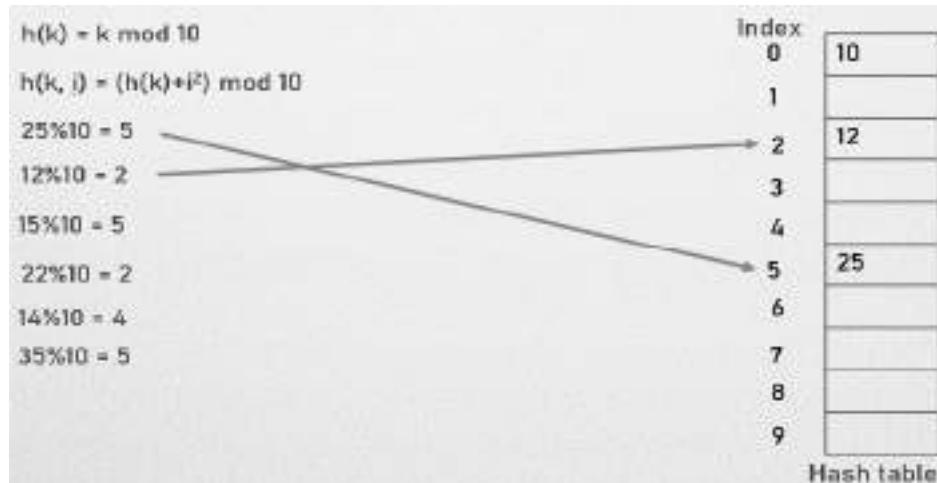
In Quadratic Probing to get slot your table size must meet these requirements:

- Be a prime number

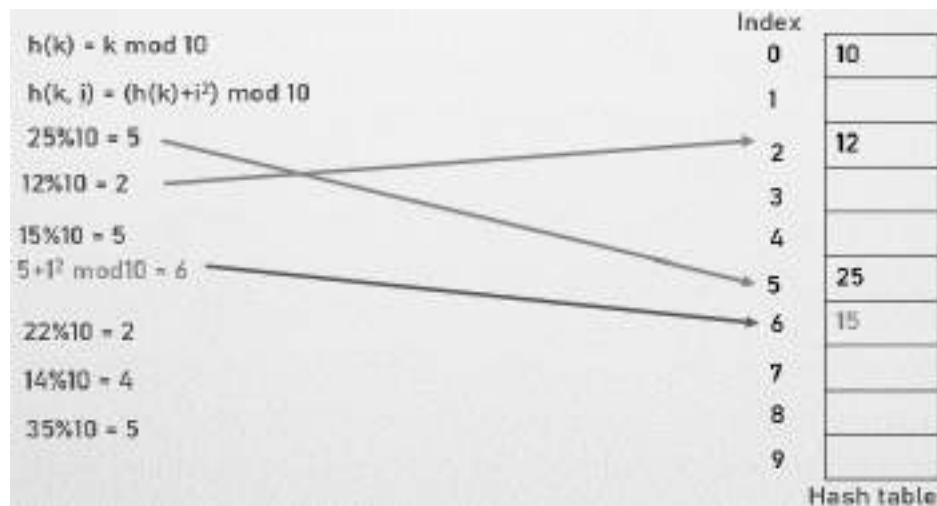
- never be more than half full (even by one element)



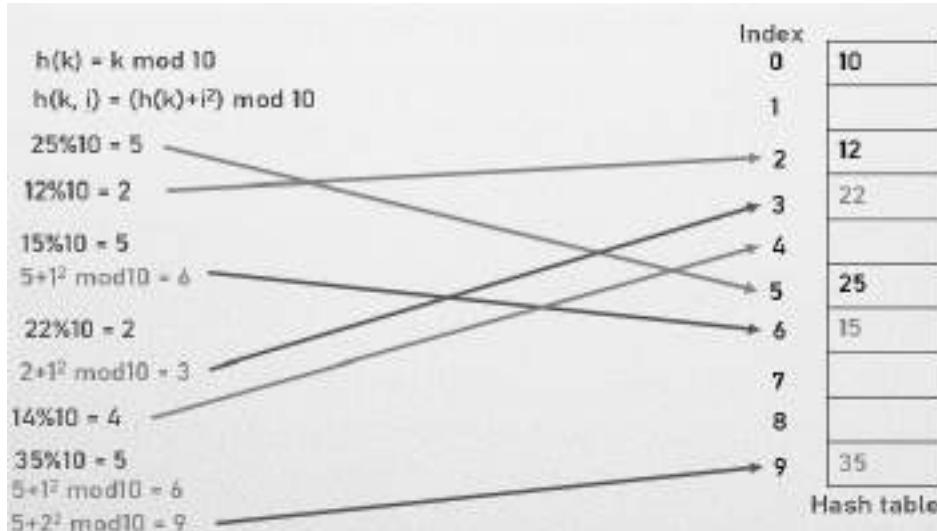
Example : Quadratic probing



a



b



c

Advantage:

Primary clustering problem resolved

Disadvantage:

Secondary clustering

No guarantee for finding slots

Separate Chaining Vs. Open Addressing

Separate Chaining	Open Addressing
Keys are stored inside the hash table as well as outside the hash table.	All the keys are stored only inside the hash table. No key is present outside the hash table.
The number of keys to be stored in the hash table can even exceed the size of the hash table.	The number of keys to be stored in the hash table can never exceed the size of the hash table.
Deletion is easier.	Deletion is difficult.
Extra space is required for the pointers to store the keys outside the hash table.	No extra space is required.
Cache performance is poor. This is because of linked lists which store the keys outside the hash table.	Cache performance is better. This is because here no linked lists are used.
Some buckets of the hash table are never used which leads to wastage of space.	Buckets may be used even if no key maps to those particular buckets.

Summary

- When two keys or hash values compete with a single hash table slot, then Collision occur.
- To resolve collision we use collision resolution techniques. Collisions can be reduced with a selection of a good hash function.
- Hash functions are mostly used in hash tables, to quickly locate a data record (for example, a dictionary definition) given its search key (the headword).
- Specifically, the hash function is used to map the search key to the index of a slot in the table where the corresponding record is supposedly stored.
- In linear probing fixed sized hash table is used and when hash collision situation occur then, we linearly traverse the table in a cyclic manner to find the next empty slot.
- Quadratic probing is a collision resolution method that eliminates the primary clustering problem of linear probing.

Keywords

Separate chaining	Quadratic probing
Linear Probing	Open Addressing
Hash function	Load factor

Self Assessment

1. Which is part of collision resolution technique ____
 - A. Separate chaining
 - B. Open addressing
 - C. Double hashing
 - D. All of above

2. Open addressing includes ____
 - A. Linear probing
 - B. Quadratic probing
 - C. Double hashing
 - D. All of above

3. In Load factor $\alpha = n/m$, m represent ____
 - A. Number of keys
 - B. Number of slots in hash table
 - C. Hash function
 - D. None of above

4. In Load factor $\alpha = n/m$, n represent ____
 - A. Number of slots in hash table
 - B. Hash function
 - C. Number of keys to be inserted in hash table
 - D. None of above

5. Worst-case complexity for searching in Separate chaining is ____
 - A. Log (1)
 - B. (n)
 - C. log (0)
 - D. None of above

6. Worst-case complexity for deletion in Separate chaining is ____
 - A. (n)
 - B. log (1)
 - C. log (2)
 - D. None of above

7. Advantages of separate chaining.

- A. It is less sensitive to the function of the hashing
- B. The hash table never fills full, so we can add more elements to the chain
- C. It is easy to implement
- D. All of above

8. Quadratic probing is part of __

- A. Open hashing
- B. Closed hashing
- C. Linear probing
- D. None of above

9. Double hashing is part of __

- A. Open hashing
- B. Closed hashing
- C. Linear probing
- D. All of above

10. Which is not part of open addressing.

- A. Linear probing
- B. Quadratic probing
- C. Open hashing
- D. All of above

11. Operations in Open Addressing are __

- A. Insert
- B. Delete
- C. Search
- D. All of above

12. In open addressing __

- A. All elements are stored in the hash table itself
- B. The size of the table must be greater than or equal to the total number of keys.
- C. During insertion, if a collision is encountered, alternative cells are tried until an empty bucket is found.
- D. All of above

13. Clustering is major problem in __

- A. Linear probing
- B. Quadratic probing
- C. Double hashing
- D. None of above

14. Which one is most relevant to Quadratic Probing?

- A. $h(k) = k \bmod 10$
- B. $h(k, i) = (h(k)+i^2) \bmod 10$
- C. $h(k, i) = (h(k)+i) \bmod 10$
- D. None of above

15. Which one is not relevant to Linear Probing?

- A. $h(k, i) = (h(k)+i) \bmod 10$
- B. $h(k) = k \bmod 10$
- C. $h(k, i) = (h(k)+i^2) \bmod 10$
- D. None of above

Answers for Self Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. D | 2. D | 3. B | 4. C | 5. B |
| 6. A | 7. D | 8. B | 9. B | 10. C |
| 11. D | 12. D | 13. A | 14. B | 15. C |

Review Questions

1. Discuss significance of collision resolution.
2. Differentiate between open hashing and closed hashing.
3. Explain cluster problem and its solution in hashing.
4. What are the advantages of quadratic probing?
5. Give an example of linear probing.
6. Discuss load factor in hashing.



Further Readings

Burkhard Monien, Data Structures and Efficient Algorithms, Thomas Ottmann, Springer.

Kruse, Data Structure & Program Design, Prentice Hall of India, New Delhi.

Mark Allen Weles, Data Structure & Algorithm Analysis in C, Second Ed., Addison-Wesley Publishing.

RG Dromey, How to Solve it by Computer, Cambridge University Press.

Lipschutz. S. (2011). Data Structures with C. Delhi: Tata McGraw hill

Reddy. P. (1999). Data Structures Using C. Bangalore: Sri Nandi Publications

Samantha. D (2009). Classic Data Structures. New Delhi: PHI Learning Private Limited



Web Links

Unit 13: Collision Resolution

www.en.wikipedia.org

<https://www.gatevidyalay.com/collision-resolution-techniques-separate-chaining/>

<http://users.csc.calpoly.edu/~gfisher/classes/103/lectures/week5.2.html>

<https://www.tutorialandexample.com/collision-resolution-techniques-in-data-structure>

Unit 14: More on Hashing

CONTENTS

- Objectives
- Introduction
- 14.1 Double Hashing
- 14.2 Rehashing
- Summary
- Keywords
- Self Assessment
- Answers for Assessment
- Review Questions
- Further Readings

Objectives

After studying this unit, you will be able to:

- Discuss Double hashing techniques
- Understand load factor
- Learn rehashing

Introduction

When two keys or hash values compete with a single hash table slot, then Collision occur. To resolve collision we use collision resolution techniques. Collisions can be reduced with a selection of a good hash function.

The open addressing technique requires a hash table with fixed and known size. All elements are stored in the hash table itself. The size of the table must be greater than or equal to the total number of keys. During insertion, if a collision is encountered, alternative cells are tried until an empty bucket is found.

In case of collision: Probing is performed until an empty bucket is found. Once an empty bucket is found, the key is inserted. Probing is performed in accordance with the technique used for open addressing.

14.1 Double Hashing

Double Hashing is a hashing collision resolution technique in open addressed Hash tables. In double hashing, there are two hash functions. The second hash function is used to provide an offset value in case the first function causes a collision. Second hash function used to remove the collision when you encountered the collision.

Double hashing uses two hash functions, one to find the initial location to place the key and a second to determine the size of the jumps in the probe sequence. The i th probe is

$$h(k, i) = (h_1(k) + i * h_2(k)) \bmod m.$$

Keys that hash to the same location, are likely to hash to a different jump size, and so will have different probe sequences. Thus, double hashing avoids secondary clustering by providing as many as m^2 probe sequences. How do we ensure every location is checked? Since each successive probe is offset by $h_2(k)$, every cell is probed if $h_2(k)$ is relatively prime to m . Two possible ways to ensure

$h_2(k)$ is relatively prime to m are, either make $m = 2k$ and design $h_2(k)$ so it is always odd, or make m prime and ensure $h_2(k) < m$. Of course, $h_2(k)$ cannot equal zero.

Double hashing can be performed using:

$$(h_1(key) + i * h_2(key)) \bmod \text{TABLE_SIZE}$$

Here $h_1()$ and $h_2()$ are hash functions

First hash function:

$$h_1(key) = \text{key} \bmod \text{TABLE_SIZE}$$

Second hash function is :

$$h_2(key) = \text{PRIME} - (\text{key} \bmod \text{PRIME})$$

Where PRIME is a prime smaller than the TABLE_SIZE

A good second Hash function is: It must never evaluate to zero. Must make sure that all cells can be probed.



Example: Double hashing

Elements: 20, 25, 36, 16, 55, 17 table size= 10 prime number = 7

	Index
h1(k) = k mod 10	0
h2(key) = Prime - (key mod Prime)	1
h2(key) = 7 - (key mod 7)	2
(h1(key) + i * h2(key)) mod Table size	3
	4
	5
	6
	7
	8
	9
	Hash table

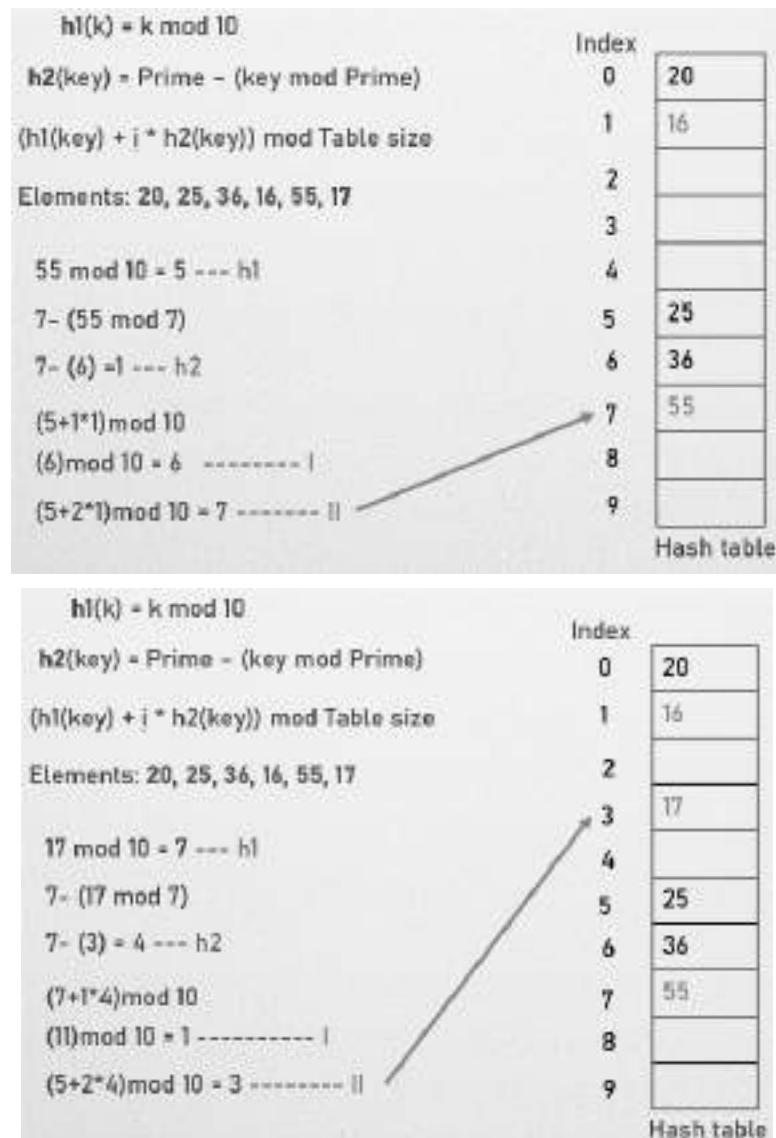
	Index
h1(k) = k mod 10	0
h2(key) = Prime - (key mod Prime)	1
(h1(key) + i * h2(key)) mod Table size	2
Elements: 20, 25, 36, 16, 55, 17	3
20 mod 10 = 0	4
	5
	6
	7
	8
	9
	Hash table

Unit 14: More on Hashing

$h1(k) = k \bmod 10$	Index	0	20
$h2(key) = \text{Prime} - (\text{key mod Prime})$	1		
$(h1(key) + i * h2(key)) \bmod \text{Table size}$	2		
Elements: 20, 25, 36, 16, 55, 17	3		
	4		
25 mod 10 = 5	5	25	
	6		
	7		
	8		
	9		
			Hash table

$h1(k) = k \bmod 10$	Index	0	20
$h2(key) = \text{Prime} - (\text{key mod Prime})$	1		
$(h1(key) + i * h2(key)) \bmod \text{Table size}$	2		
Elements: 20, 25, 36, 16, 55, 17	3		
	4		
36 mod 10 = 6	5	25	
	6	36	
	7		
	8		
	9		
			Hash table

$h1(k) = k \bmod 10$	Index	0	20
$h2(key) = \text{Prime} - (\text{key mod Prime})$	1	16	
$(h1(key) + i * h2(key)) \bmod \text{Table size}$	2		
Elements: 20, 25, 36, 16, 55, 17	3		
	4		
16 mod 10 = 6 --- h1	5	25	
$7 - (16 \bmod 7)$	6	36	
$7 - (2) = 5$ --- h2	7		
$(6+1*5)\bmod 10$	8		
$(11)\bmod 10 = 1$	9		
			Hash table



Double hashing highlights

- Computational cost is higher.
- No primary clustering.
- No secondary clustering.
- Double hashing can find the next free slot faster than the linear probing approach.
- Double hashing is used for uniform distribution of records throughout a hash table.
- Double hashing is useful if an application requires a smaller hash table.

14.2 Rehashing

This is another method of collision handling. In this method you find an alternative empty location by modifying the hash function, and applying the modified hash function to the colliding symbol. For example, if x is symbol and $h(x) = i$, and if the i th location is already occupied, then I modify the hash function h to h_1 , and find out $h_1(x)$, if $h_1(x) = j$, and j th location is empty, then I accommodate x in the j th location. Otherwise you once again modify h_1 to some h_2 and repeat the process till the collision gets handled. Once the collision gets handled we revert back to the original hash function before considering the next symbol.

It is process of re-calculating the hash code of already stored entries. The Hash table provides Constant time complexity of insertion and searching, provided the hash function is able to distribute the input load evenly. In case of Collision, the time complexity can go up to $O(N)$ in the worst case. Rehashing of a hash map is done when the number of elements in the map reaches the maximum threshold value.

When load factor increases to more than its predefined value, complexity increases. To overcome this problem, size of array is increased and all the values are hashed again and stored in new double size array to maintain a low load factor and complexity.

Load factor

Load factor is number of element (n) divide by number of bucket (m).

$$\text{Load factor } (\lambda) = n/m$$

$$-\lambda < 1 \text{ i.e. } m > n$$

if $\lambda < 1$ then no need to apply rehashing

if $\lambda > 1$ then we need to increase number of buckets

Increase in bucket size is known as rehashing. The Load Factor decides "when to increase the size of the hash Table."

Rehashing steps

-Increase number of buckets.

-Modify hash function

Hash function before rehashing : $x \bmod m$

after rehashing $x \bmod m'$

-apply changed hash function to existing elements.

m' calculation

$m' = \text{closest prime number of } 2m$

Example:

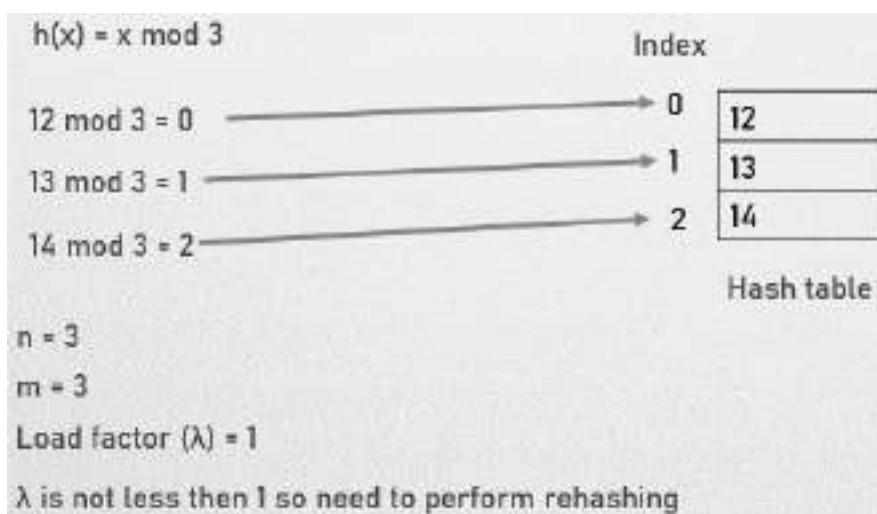
$$m=3 \quad m'=2(3)=6$$

Closest prime number = 5 or 7.



Example: Rehashing

Elements: 12, 13, 14 table size = 3



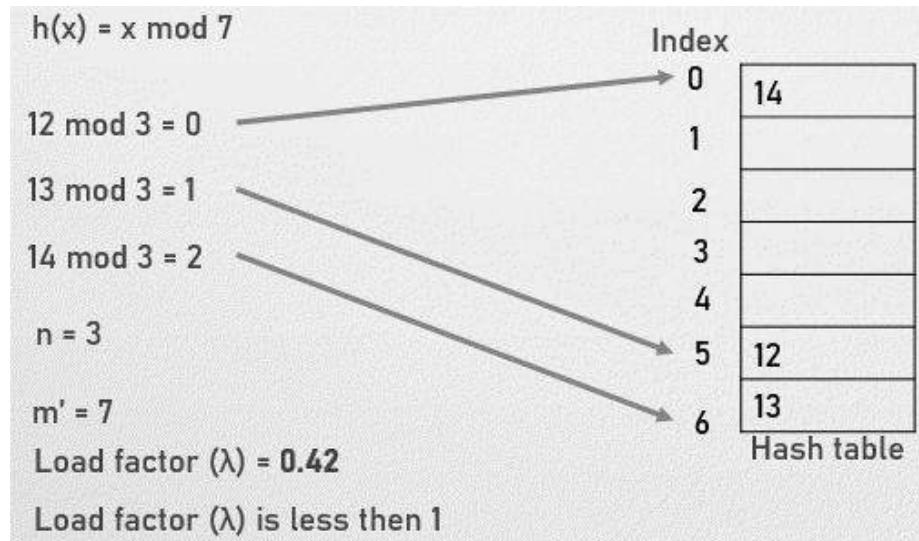
$$m' = 2n$$

$$m' = 2 \times 3 \Rightarrow 6$$

Nearest prime numbers are 5 and 7

$$m' = 7$$

$$x \bmod m' \Rightarrow x \bmod 7$$



A Comparison of Rehashing Methods

Linear Probing	m distinct probe sequences	Primary clustering
Quadratic Probing	m distinct probe sequences	No primary clustering; but secondary clustering
Double Probing	m^2 distinct probe sequences	No primary clustering No secondary clustering

Complexity of Rehashing

Time complexity – $O(n)$

Space complexity – $O(n)$

Summary

- Rehashing schemes use a second hashing operation when there is a collision.
- The open addressing technique requires a hash table with fixed and known size.
- Double Hashing is a hashing collision resolution technique in open addressed Hash tables.
- Rehashing is process of re-calculating the hash code of already stored entries.
- The load factor in HashMap is basically a measure that decides when exactly to increase the size of the HashMap to maintain the same time complexity of $O(1)$.
- The Load Factor decides “when to increase the size of the hash Table.”

Keywords

Rehashing

Load factor

Hash map

Open addressing

Double hashing

Prime number

Clustering

Self Assessment

1. Which statement is correct about open addressing?
 - A. It requires a hash table with fixed and known size.
 - B. All elements are stored in the hash table itself
 - C. The size of the table must be greater than or equal to the total number of keys.
 - D. All of above

2. In double hashing prime value is __
 - A. Less than table size
 - B. Greater than table size
 - C. Equal to table size
 - D. None of above

3. In double hashing how many hash functions used.
 - A. 4
 - B. 2
 - C. 1
 - D. 3

4. Which statement is correct about double hashing?
 - A. The second hash function is used to provide an offset value in case the first function causes a collision.
 - B. In double hashing, there are two hash functions.
 - C. Second hash function used to remove the collision when you encountered the collision.
 - D. All of above

5. Which hash function used in double hashing?
 - A. $(h1(key) + h2(key)) \text{ mod Table size}$
 - B. $(h1(key) + i * (key)) \text{ mod Table size}$
 - C. $(h1(key) + i * h2(key)) \text{ mod Table size}$
 - D. None of above

6. In which situation second hash function is used in double hashing?
 - A. In case table size is smaller
 - B. In case the first function causes a collision
 - C. In case first hash function provide zero value
 - D. None of above

7. In statement- $h2(key) = 7 - (\text{key mod } 7)$, 7 represent __
 - A. Table size
 - B. Key value
 - C. Prime number

- D. None of above
8. Which statement is correct for double hashing technique?
- A. No primary clustering
 - B. No secondary clustering
 - C. Double hashing can find the next free slot faster than the linear probing approach
 - D. All of above
9. Which statement is correct about rehashing?
- A. In which the table is resized
 - B. It is process of re-calculating the hash code of already stored entries
 - C. It is a collision resolution technique
 - D. All of above
10. In Load factor = n/m , m represents _
- A. Number of element
 - B. Number of key values
 - C. Number of bucket
 - D. None of above
11. In which situations rehashing is required.
- A. When table is completely full.
 - B. With quadratic probing when the table is filled half.
 - C. When insertions fail due to overflow.
 - D. All of above
12. The value of Load factor in rehashing should be__
- a) Less than 1
 - b) Greater than 1
 - c) Equal to 0
 - d) All of above
13. In Load factor = n/m , n represents _
- A. Number of bucket
 - B. Number of element
 - C. Number of key values
 - D. None of above
14. Table size is 3 and elements are 12, 13, and 14. Load factor is__
- A. 3
 - B. 2
 - C. 1

D. 0

15. What is notation for load factor?

- A. λ
- B. ∞
- C. μ
- D. Ω

Answers for Assessment

- | | | | | |
|-------|-------|-------|-------|-------|
| 1. D | 2. A | 3. B | 4. D | 5. C |
| 6. B | 7. C | 8. D | 9. D | 10. C |
| 11. D | 12. A | 13. B | 14. C | 15. A |

Review Questions

1. What are the conditions for double hashing?
2. Discuss double hashing technique with example.
3. What are the two hash functions used in double hashing?
4. What is significance of load factor in hashing?
5. What are the advantages of double hashing?
6. Discuss steps for rehashing.
7. What is time and space complexity of rehashing?



Further Readings

Burkhard Monien, Data Structures and Efficient Algorithms, Thomas Ottmann, Springer.

Kruse, Data Structure & Program Design, Prentice Hall of India, New Delhi.

Mark Allen Weles, Data Structure & Algorithm Analysis in C, Second Ed., Addison-Wesley Publishing.

RG Dromey, How to Solve it by Computer, Cambridge University Press.

Lipschutz. S. (2011). Data Structures with C. Delhi: Tata McGraw hill

Reddy. P. (1999). Data Structures Using C. Bangalore: Sri Nandi Publications

Samantha. D (2009). Classic Data Structures. New Delhi: PHI Learning Private Limited



Web Links

www.en.wikipedia.org

<https://learningsolo.com/what-is-rehashing-and-load-factor-in-hashmap/>

<https://www.scaler.com/topics/data-structures/load-factor-and-rehashing/>

<https://www.javatpoint.com/double-hashing-in-java>

<https://www.educative.io/edpresso/what-is-double-hashing>

LOVELY PROFESSIONAL UNIVERSITY

Jalandhar-Delhi G.T. Road (NH-1)

Phagwara, Punjab (India)-144411

For Enquiry: +91-1824-521360

Fax.: +91-1824-506111

Email: odl@lpu.co.in