TPCT's College of Engineering, Osmanabad

Second Year Engineering
Semester-I
CSE: Linux Operating System

List of Experiments with lab Manuals

1. **Installation of Linux operating system using CD/DVD/USB drive or PXE boot.**
2. **Execution of various file & directory handling commands.**
3. **Execution of simple C and C++ programs using CC and GCC compiler.**
4. **Create, mount & resize partition on disk.**
5. **Create user, group and assign various permissions to access a directory.**
6. **Share a directory in LAN using SMB.**
7. **Write a shell script program input-output statements and loops.**
8. **Write a shell script program using array & case statement.**
9. **Use of various text processing tools: grep & sed.**
10. **Write a program in AWK using loops.**

Experiment No. 01

Aim: Installation of Linux operating system using CD/DVD/USB drive or PXE boot.

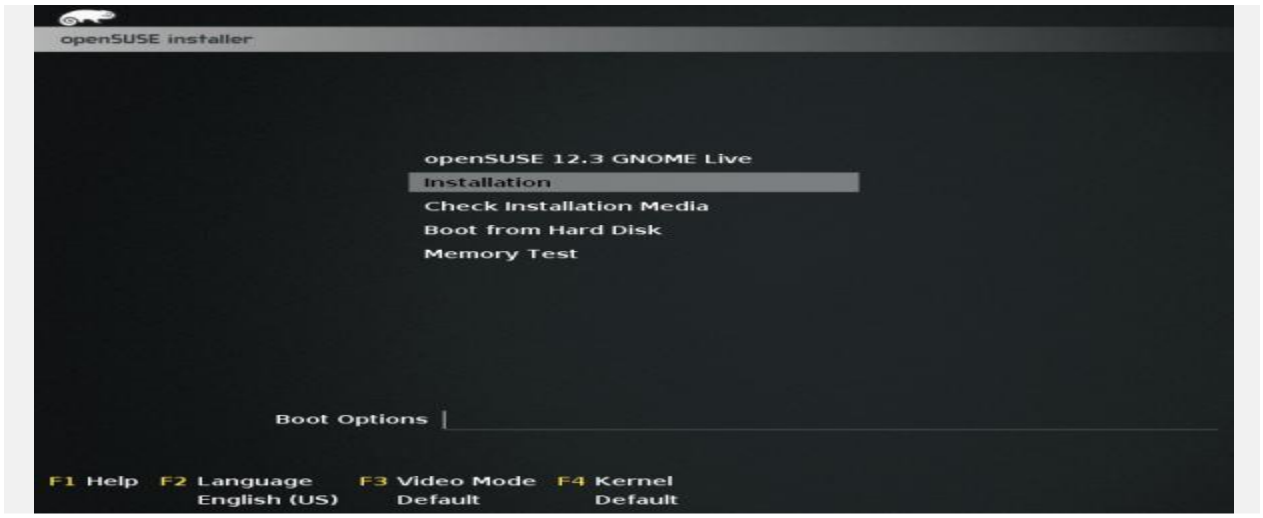
Tools Required: Linux operating system(any flavor) CD/DVD/USB.

Objective: To learn installation of Linux operating system.

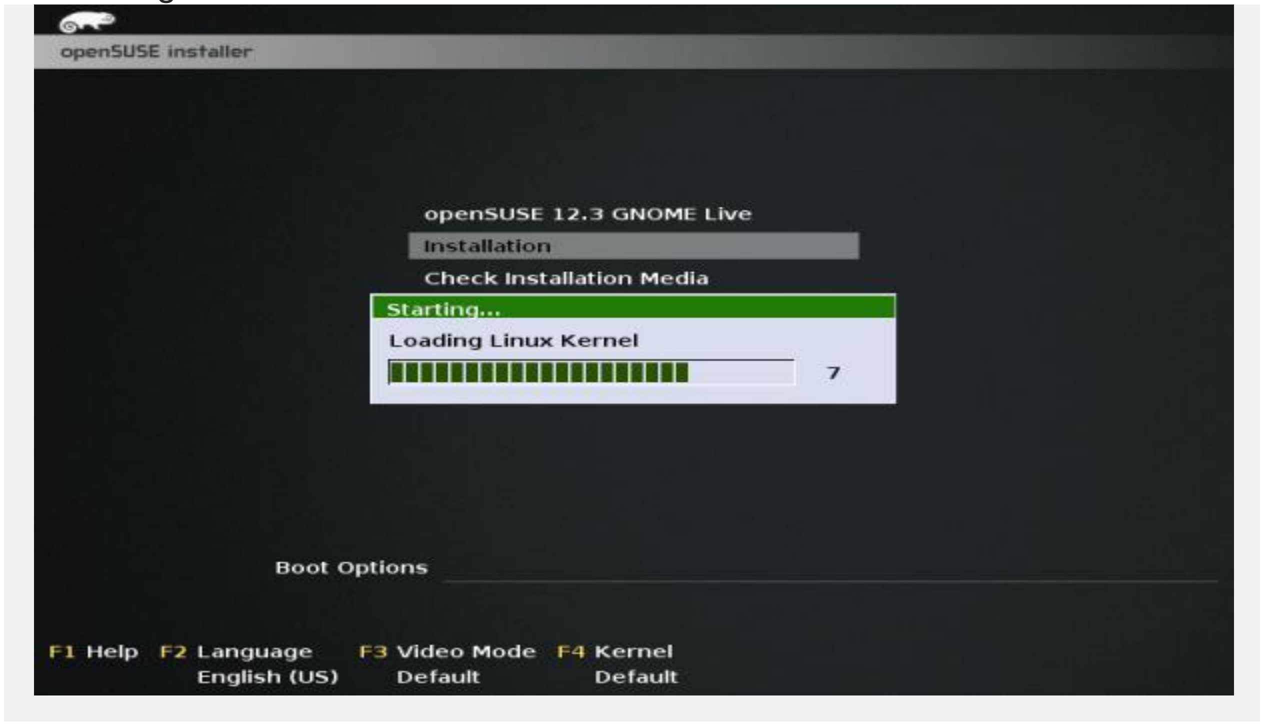
Thoery:

1. Boot your system with OpenSUSE 12.3 installation media i.e CD/DVD or ISO image.

2. Choose installation options to install openSUSE 12.3 on you system. Please select openSUSE 12.3 GNOME Live options to test it, before installation.

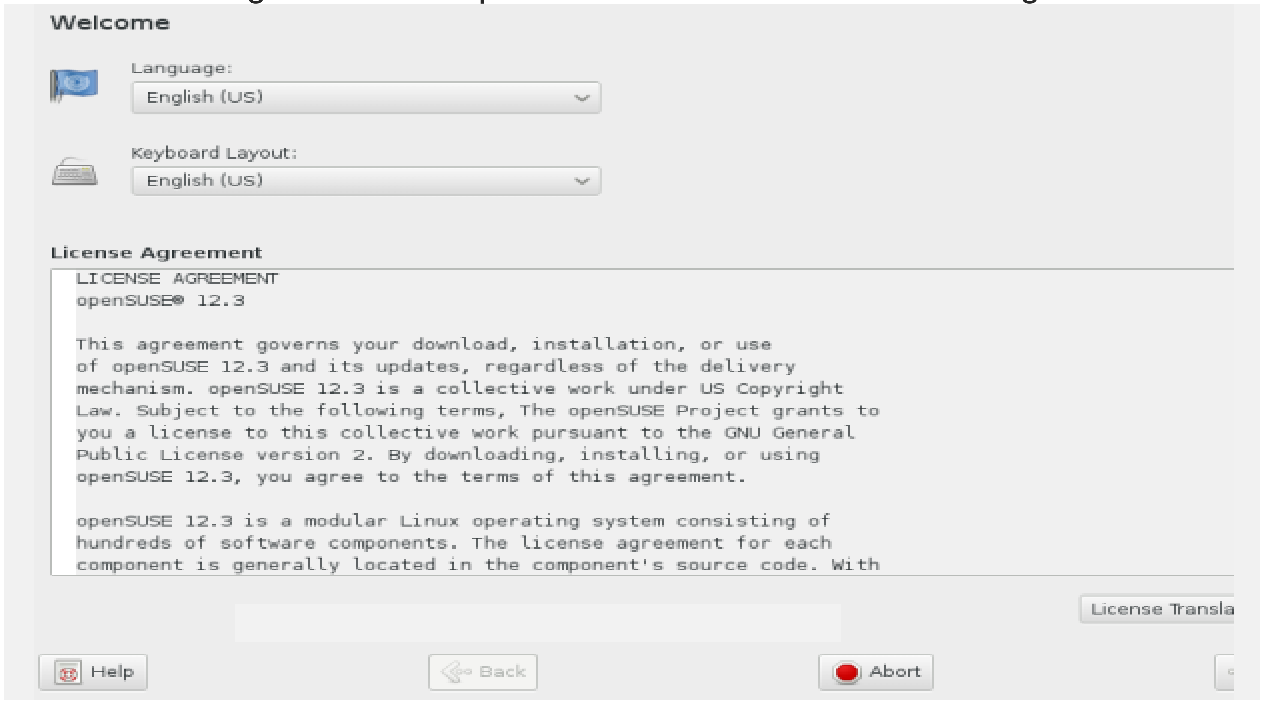


3. Loading Linux kernel.



Starting...
Loading Linux Kernel
7

4. Welcome screen, From where we can select Language and keyboard layout. Read license agreement and proceed further installation once agreed.


Welcome

Language: English (US)

Keyboard Layout: English (US)

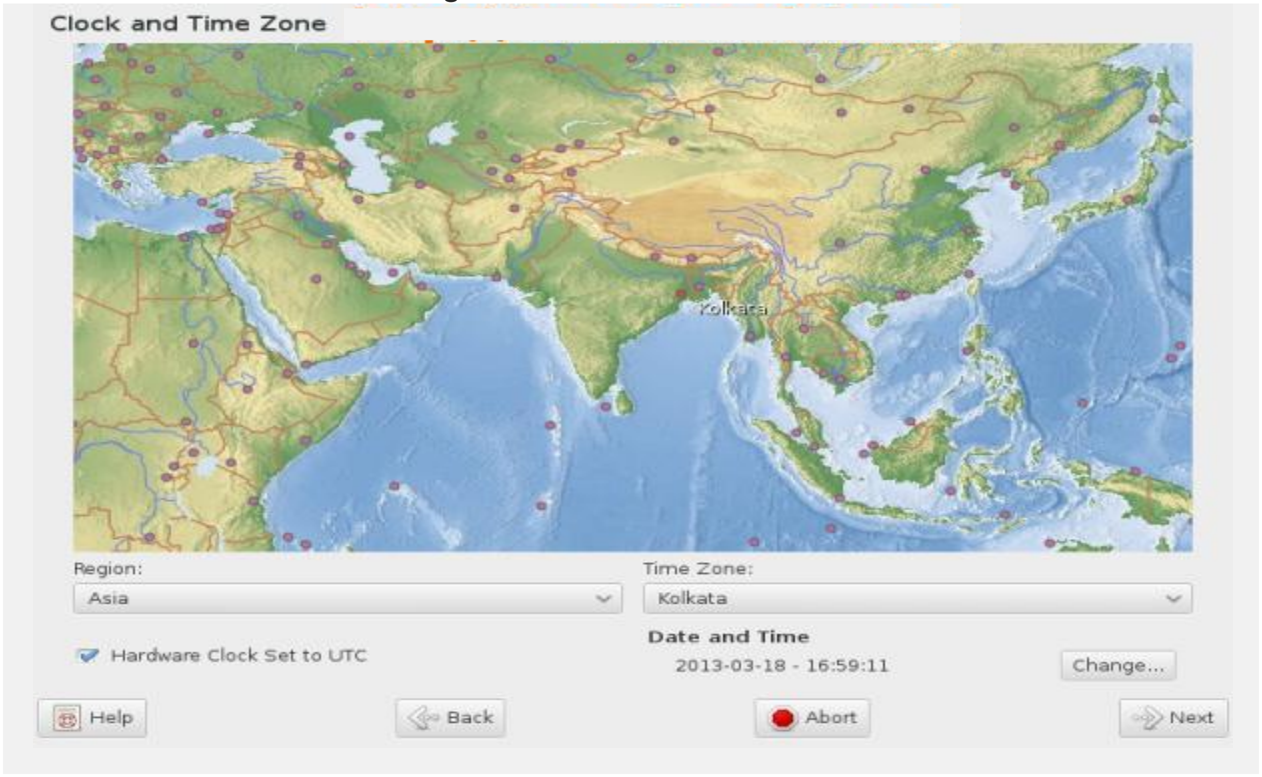
License Agreement

```
LICENSE AGREEMENT
openSUSE® 12.3

This agreement governs your download, installation, or use
of openSUSE 12.3 and its updates, regardless of the delivery
mechanism. openSUSE 12.3 is a collective work under US Copyright
Law. Subject to the following terms, The openSUSE Project grants to
you a license to this collective work pursuant to the GNU General
Public License version 2. By downloading, installing, or using
openSUSE 12.3, you agree to the terms of this agreement.

openSUSE 12.3 is a modular Linux operating system consisting of
hundreds of software components. The license agreement for each
component is generally located in the component's source code. With
```

5. Clock and timezone settings.



Region: Asia

Time Zone: Kolkata

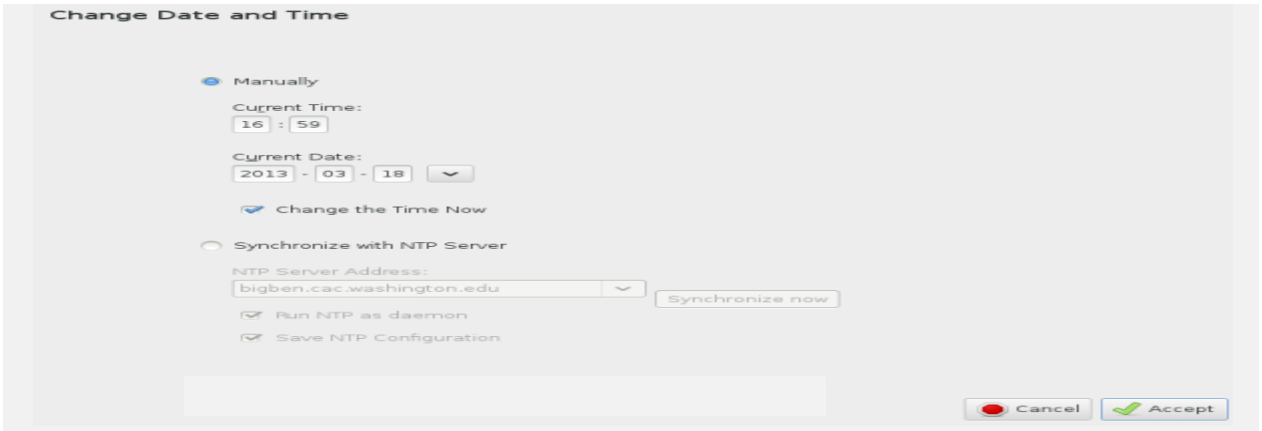
✓ Hardware Clock Set to UTC

Date and Time

2013-03-18 - 16:59:11

Change...

6. Please click on change if you want custom setting of date and time. You can change it manually or sync with NTP Server as show below. Click Accept once done.



● Manually

Current Time:

16 : 59

Current Date:

2013 - 03 - 18

✓ Change the Time Now

○ Synchronize with NTP Server

NTP Server Address:

bigben.cac.washington.edu

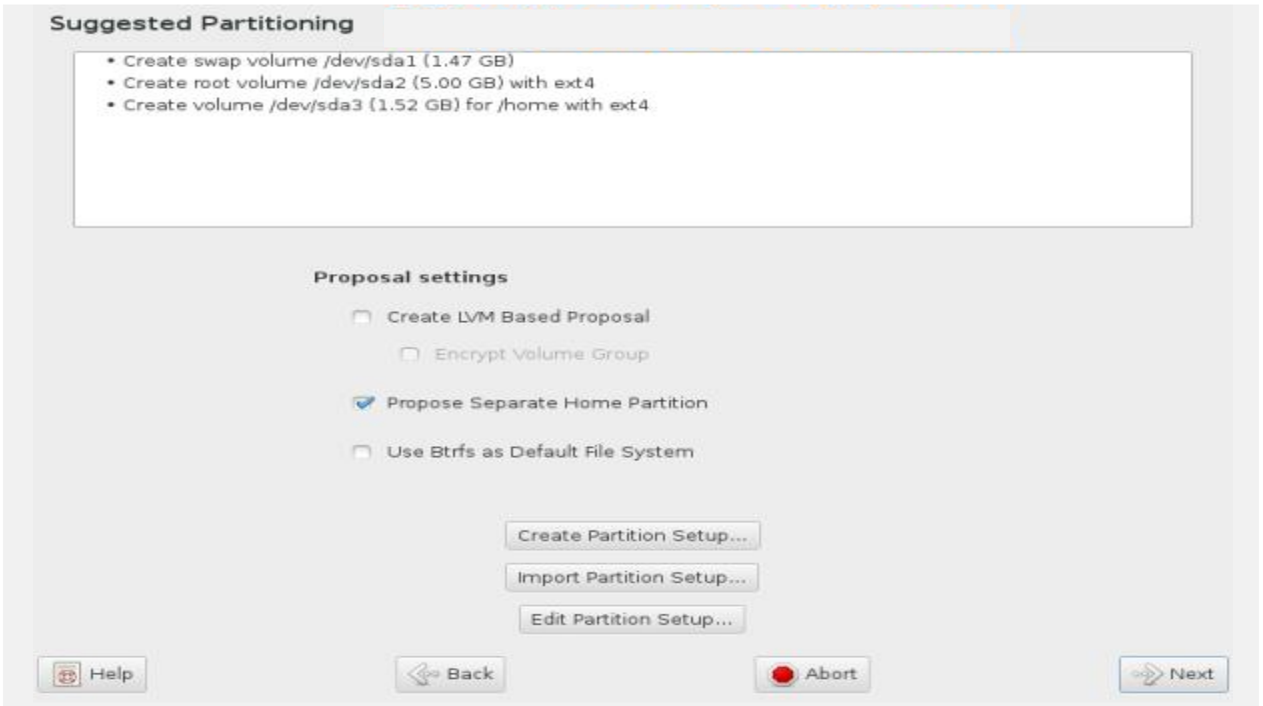
Synchronize now

✓ Run NTP as daemon

✓ Save NTP Configuration

Set Date and Time

7. File system partitioning. We opted default filesystem partition. You may choose manual filesystem partitioning as options provided.


Suggested Partitioning

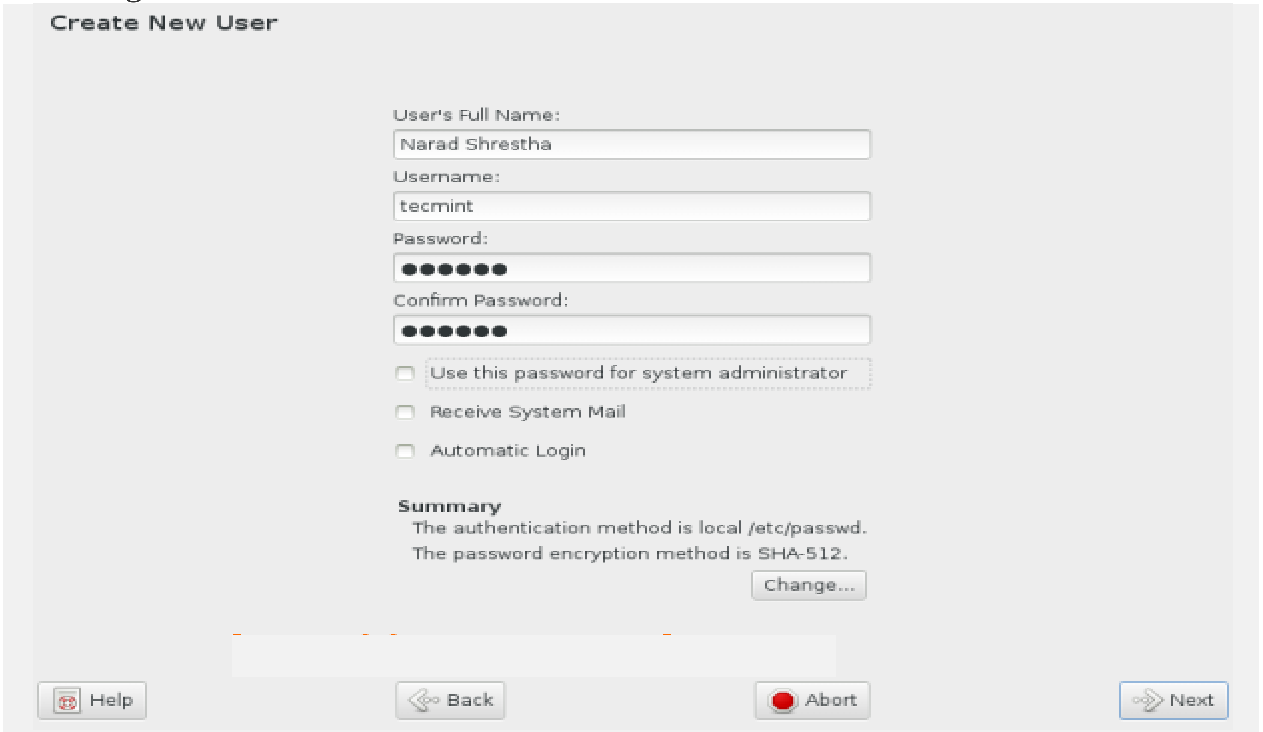
- Create swap volume /dev/sda1 (1.47 GB)
- Create root volume /dev/sda2 (5.00 GB) with ext4
- Create volume /dev/sda3 (1.52 GB) for /home with ext4

Proposal settings

- [ ] Create LVM Based Proposal
  - [ ] Encrypt Volume Group
- [x] Propose Separate Home Partition
- [ ] Use Btrfs as Default File System

OpenSuse File System Partitioning

8. Create new user and it's password. Uncheck all three options. Click on change to select authentication method.


Create New User

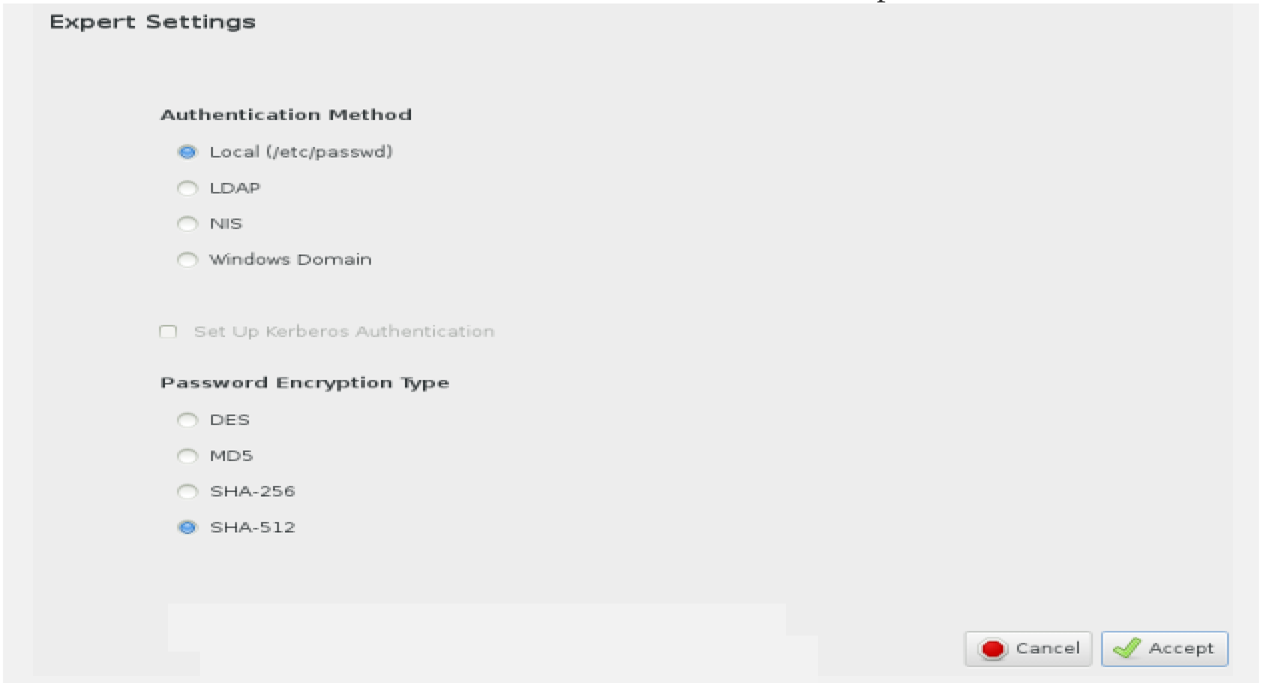
User's Full Name: Narad Shrestha
Username: tecmint
Password: ●●●●●●
Confirm Password: ●●●●●●

- [ ] Use this password for system administrator
- [ ] Receive System Mail
- [ ] Automatic Login

Summary
The authentication method is local /etc/passwd.
The password encryption method is SHA-512.

Create User and Password

9. Please select authentication method and click on Accept.



Expert Settings

Authentication Method

- [✓ Local (/etc/passwd)]
- LDAP
- NIS
- Windows Domain

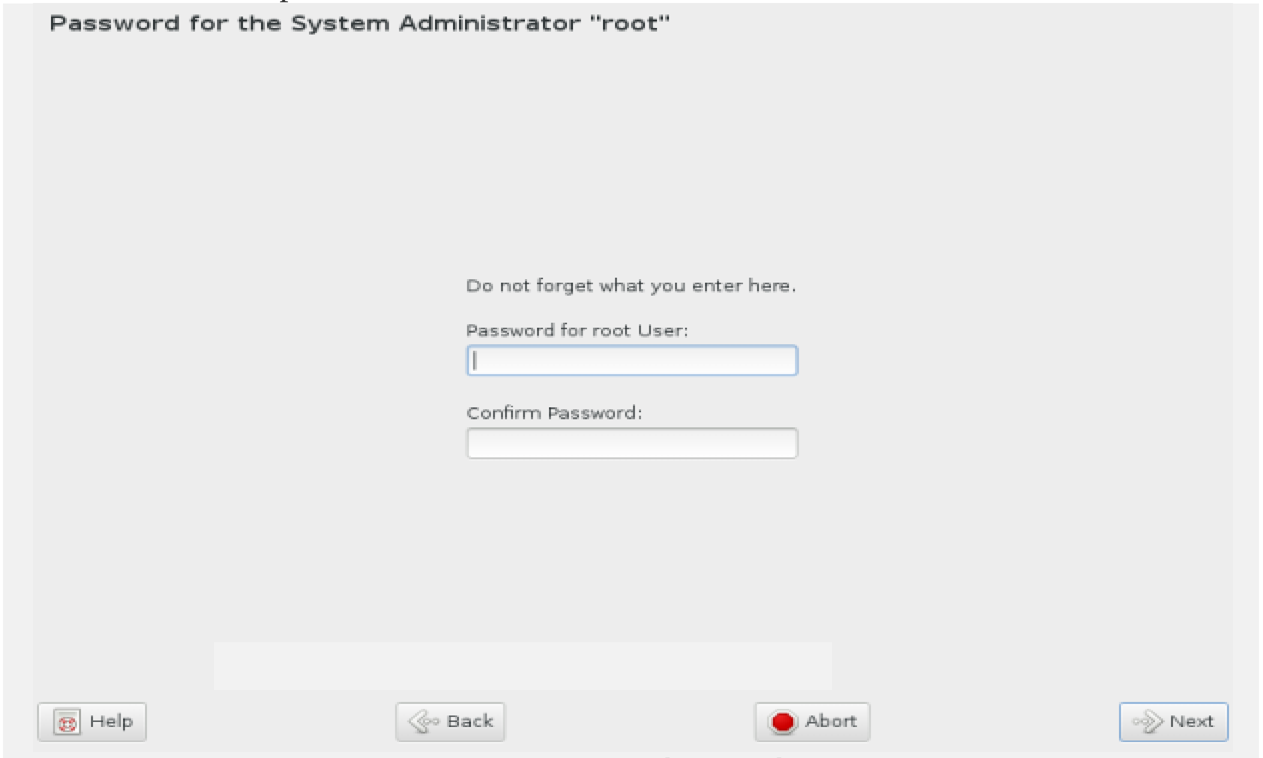
☐ Set Up Kerberos Authentication

Password Encryption Type

- DES
- MD5
- SHA-256
- [✓ SHA-512]

Select Authentication Method

10. Set root user password and click on Next.



Password for the System Administrator "root"

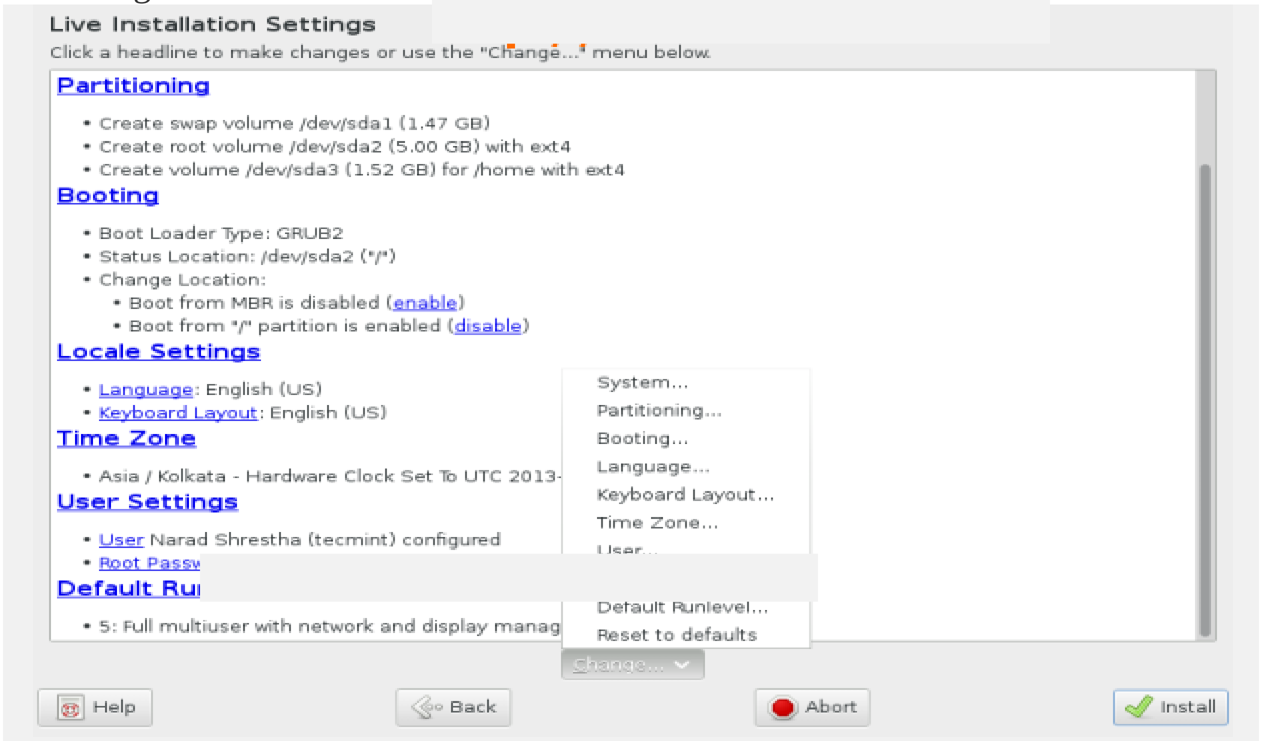
Do not forget what you enter here.

Password for root User:

Confirm Password:

Set root User and Password

11. Verify settings, you may change settings after clicking on headlines or click on Change button. Once done click on Install.


Live Installation Settings

Click a headline to make changes or use the "Change..." menu below.

Partitioning

- Create swap volume /dev/sda1 (1.47 GB)
- Create root volume /dev/sda2 (5.00 GB) with ext4
- Create volume /dev/sda3 (1.52 GB) for /home with ext4

Booting

- Boot Loader Type: GRUB2
- Status Location: /dev/sda2 ("/")
- Change Location:
  - Boot from MBR is disabled (enable)
  - Boot from "/" partition is enabled (disable)

Locale Settings

- Language: English (US)
- Keyboard Layout: English (US)

Time Zone

- Asia / Kolkata - Hardware Clock Set To UTC 2013-

User Settings

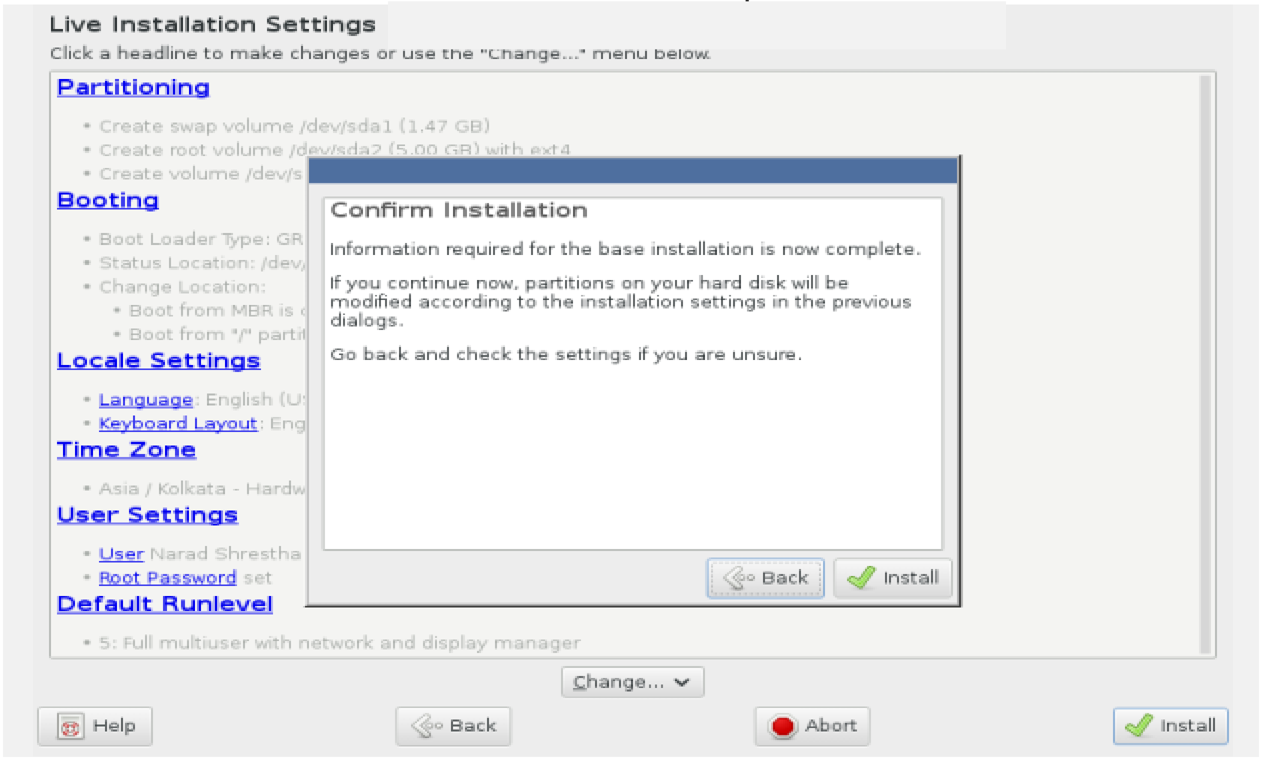
- User Narad Shrestha (tecmint) configured
- Root Passw

Default Ru

- 5: Full multiuser with network and display manag

Verify OpenSuse Settings

12. Installation confirmation. Click on Install to proceed.


Live Installation Settings

Click a headline to make changes or use the "Change..." menu below.

Partitioning

- Create swap volume /dev/sda1 (1.47 GB)
- Create root volume /dev/sda2 (5.00 GB) with ext4
- Create volume /dev/s

Booting

- Boot Loader Type: GR
- Status Location: /dev/
- Change Location:
  - Boot from MBR is
  - Boot from "/" parti

Locale Settings

- Language: English (U
- Keyboard Layout: Eng

Time Zone

- Asia / Kolkata - Hardw

User Settings

- User Narad Shrestha
- Root Password set

Default Runlevel

- 5: Full multiuser with network and display manager

Confirm Installation

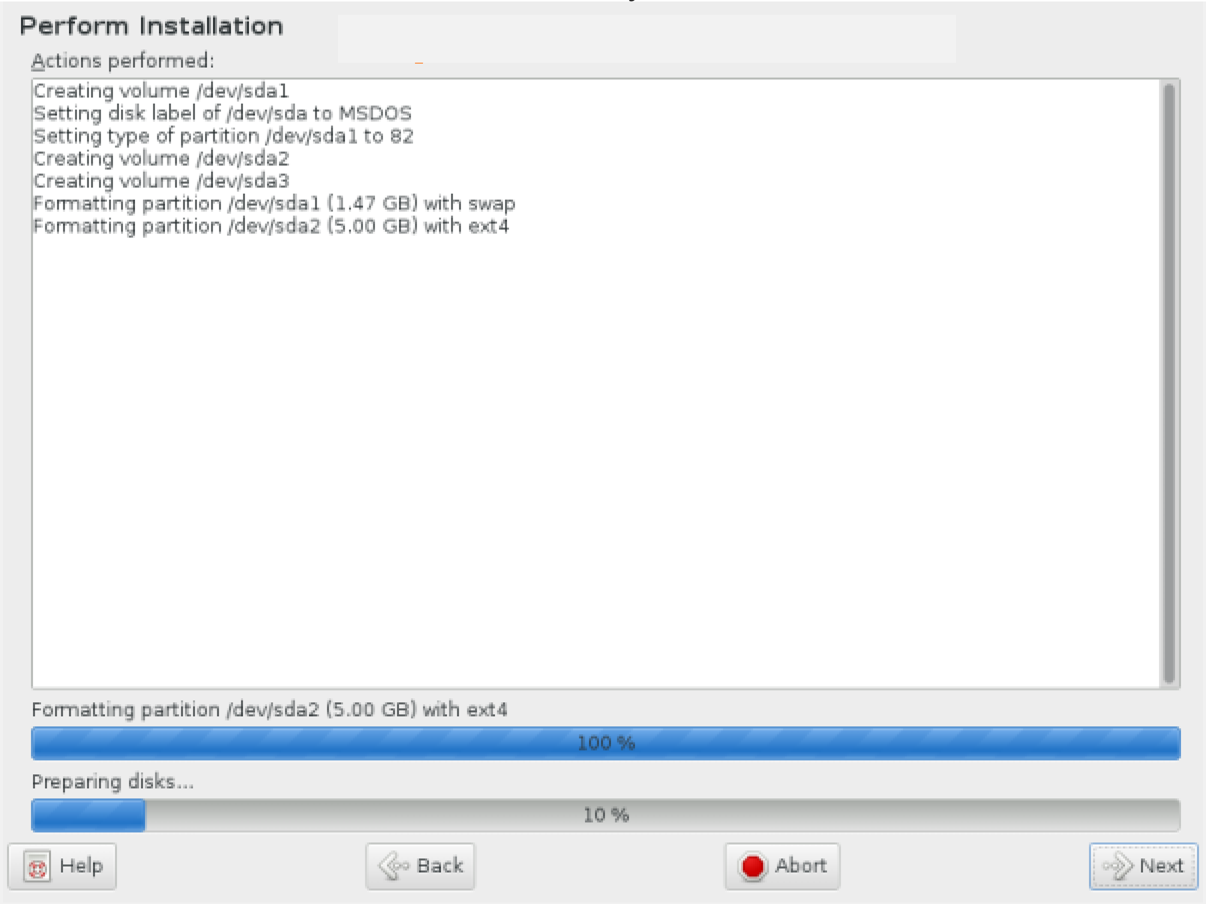
Information required for the base installation is now complete.

If you continue now, partitions on your hard disk will be modified according to the installation settings in the previous dialogs.

Go back and check the settings if you are unsure.

Start OpenSuse Installation

13. Performing installation. Creating volume and formatting filesystem for installation. Sit back and relax… This may take several time.



Perform Installation

Actions performed:

```
Creating volume /dev/sda1
Setting disk label of /dev/sda to MSDOS
Setting type of partition /dev/sda1 to 82
Creating volume /dev/sda2
Creating volume /dev/sda3
Formatting partition /dev/sda1 (1.47 GB) with swap
Formatting partition /dev/sda2 (5.00 GB) with ext4
```

Formatting partition /dev/sda2 (5.00 GB) with ext4

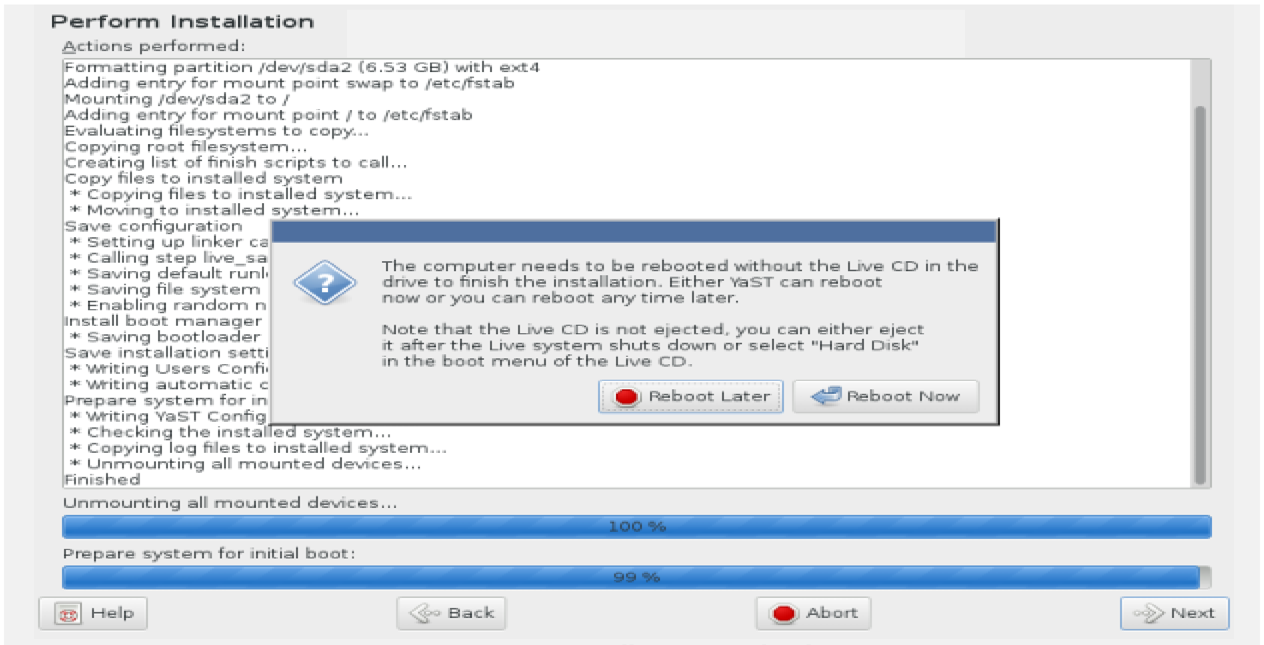
100 %

Preparing disks...

10 %

Performing OpenSuse Installation

14. Installation completed, remove installation media and click on Reboot Now.



Perform Installation

Actions performed:

```
Formatting partition /dev/sda2 (6.53 GB) with ext4
Adding entry for mount point swap to /etc/fstab
Mounting /dev/sda2 to /
Adding entry for mount point / to /etc/fstab
Evaluating filesystems to copy...
Copying root filesystem...
Creating list of finish scripts to call...
Copy files to installed system
 * Copying files to installed system...
 * Moving to installed system...
Save configuration
 * Setting up linker ca
 * Calling step live_sa
 * Saving default runl
 * Saving file system
 * Enabling random n
Install boot manager
 * Saving bootloader
Save installation setti
 * Writing Users Confi
 * Writing automatic c
Prepare system for in
 * Writing YaST Config
 * Checking the installed system...
 * Copying log files to installed system...
 * Unmounting all mounted devices...
Finished
```

The computer needs to be rebooted without the Live CD in the drive to finish the installation. Either YaST can reboot now or you can reboot any time later.

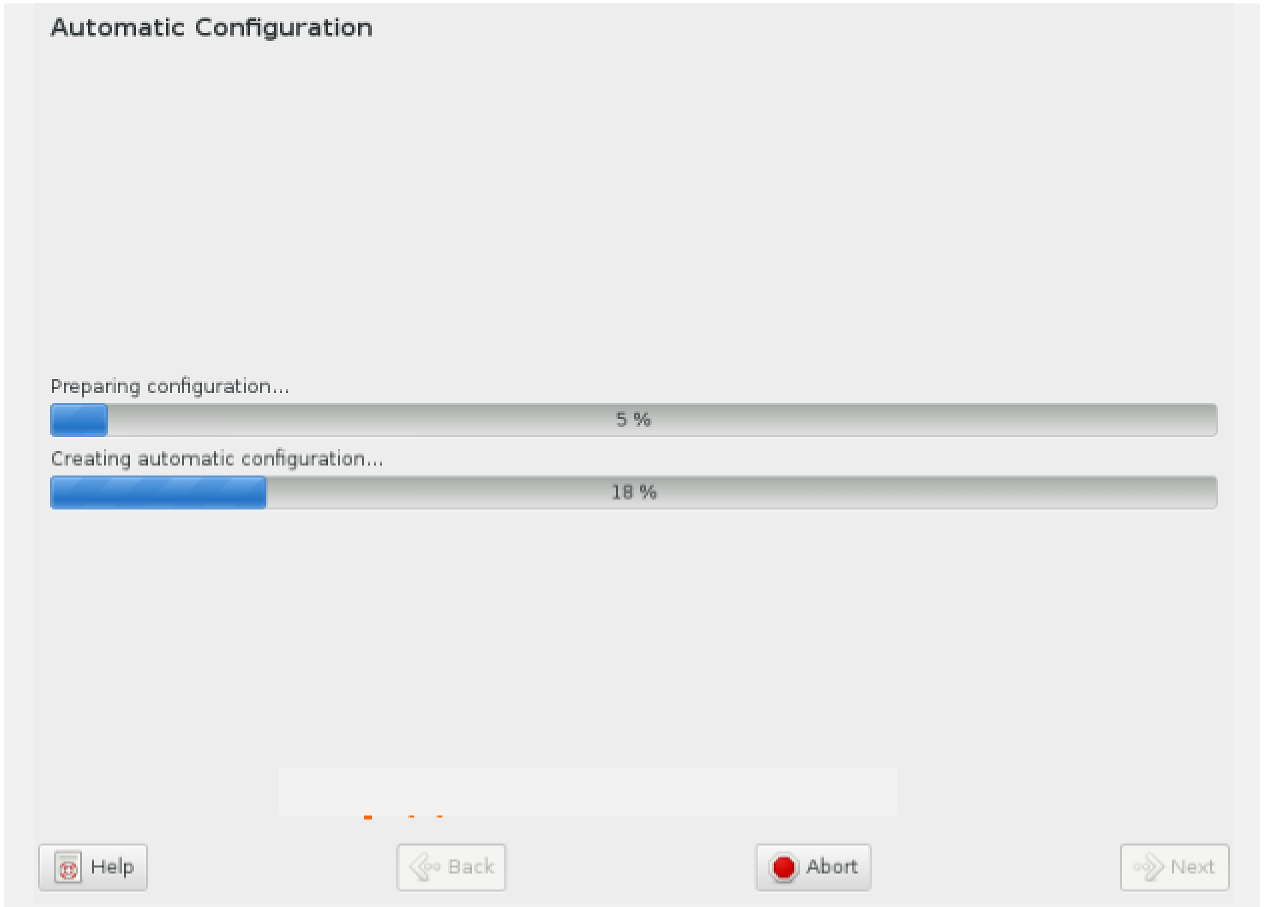
Note that the Live CD is not ejected, you can either eject it after the Live system shuts down or select "Hard Disk" in the boot menu of the Live CD.

Unmounting all mounted devices... 100 %

Prepare system for initial boot: 99 %

OpenSuse Installation Completed

15. Post installation.



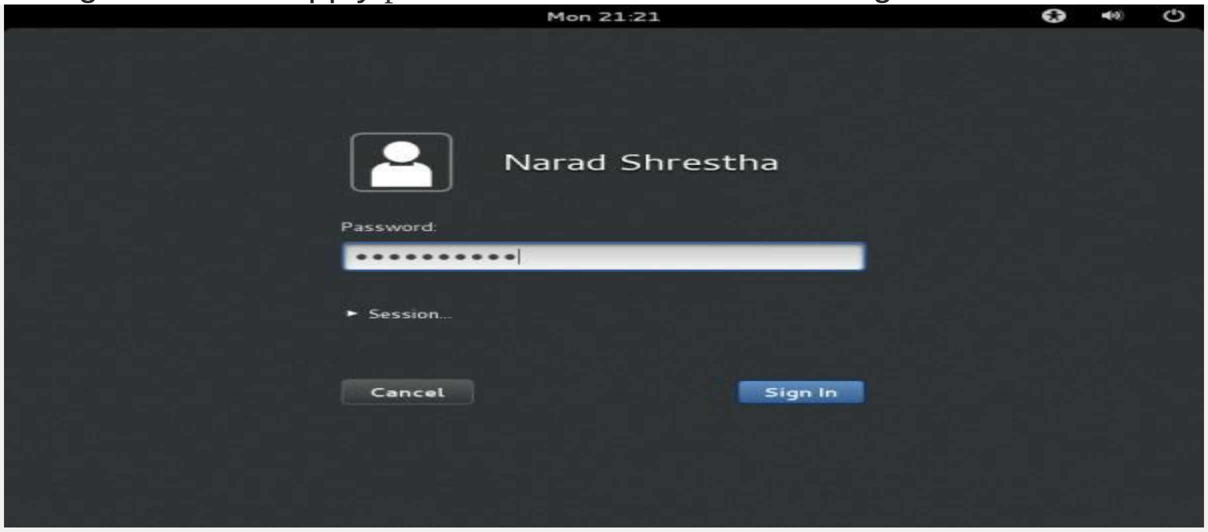
Automatic Configuration

Preparing configuration... 5 %

Creating automatic configuration... 18 %

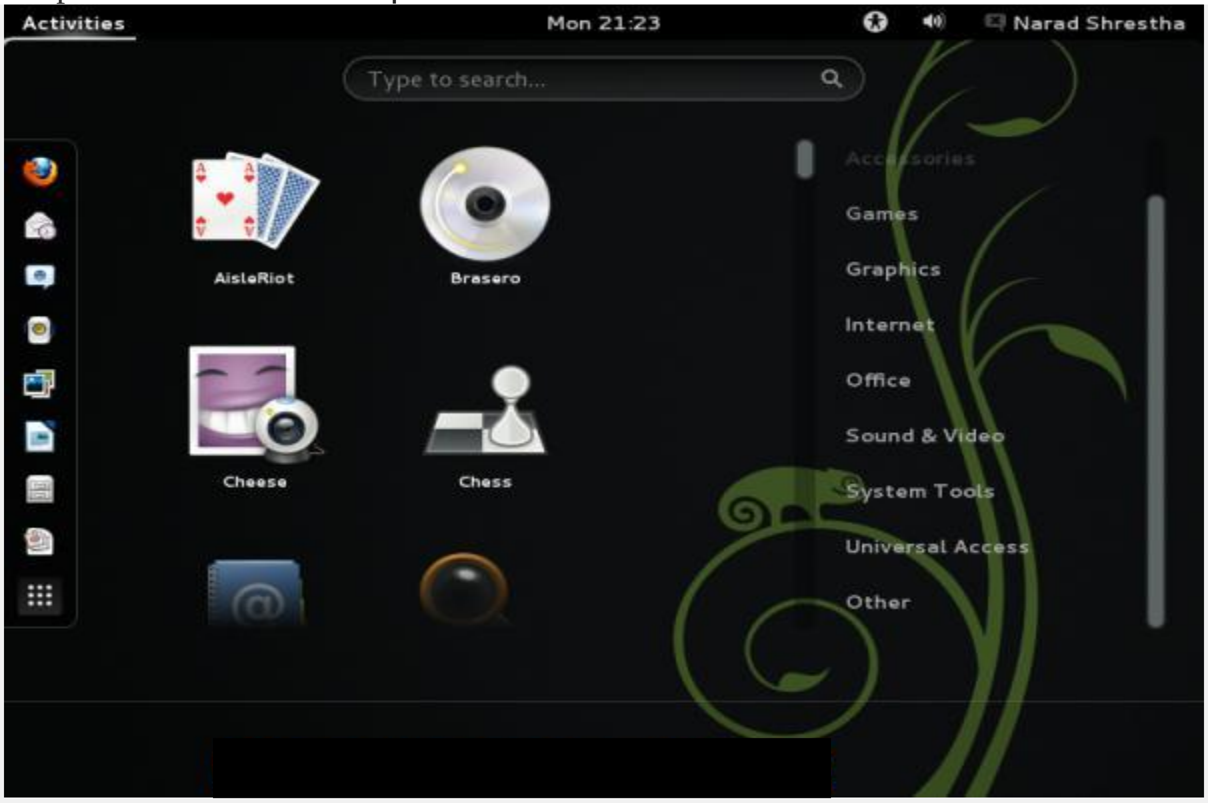
OpenSuse Post Installation

16. Login screen. Supply password for user created during installation.



OpenSuse Login Screen

17. openSUSE 12.3 Desktop.



OpenSuse 12.3 Desktop

Conclusion: Hence we have implemented installation of Linux operating system.

Experiment No. 02

1. **Aim: Execution of various file & directory handling commands.**

Tools Required: Linux operating system.

Objective: To learn how to implement and usage of file and directory handling commands.

Theory:

mkdir - make directories

Usage

mkdir [OPTION] DIRECTORY

Options

Create the DIRECTORY(ies), if they do not already exist.

Mandatory arguments to long options are mandatory for short options too.

-m, mode=MODE set permission mode (as in chmod), not rwxrwxrwx - umask

-p, parents no error if existing, make parent directories as needed

-v, verbose print a message for each created directory

-help display this help and exit

-version output version information and exit

cd - change directories

Use cd to change directories. Type cd followed by the name of a directory to access that directory.Keep in mind that you are always in a directory and can navigate to directories hierarchically above or below.

mv- change the name of a directory

Type mv followed by the current name of a directory and the new name of the directory.

Ex: mv testdir newnamedir

pwd - print working directory

will show you the full path to the directory you are currently in. This is very handy to use, especially when performing some of the other commands on this page

rmdir - Remove an existing directory

rm -r

Removes directories and files within the directories recursively.

chown - change file owner and group

Usage

chown [OPTION] OWNER[:[GROUP]] FILE

chown [OPTION] :GROUP FILE

chown [OPTION] --reference=RFILE FILE

Options

Change the owner and/or group of each FILE to OWNER and/or GROUP. With --reference, change the owner and group of each FILE to those of RFILE.

-c, changes like verbose but report only when a change is made

-dereference affect the referent of each symbolic link, rather than the symbolic link itself

-h, no-dereference affect each symbolic link instead of any referenced file (useful only on systems that can change the ownership of a symlink)

-from=CURRENT_OWNER:CURRENT_GROUP

change the owner and/or group of each file only if its current owner and/or group match those specified here. Either may be omitted, in which case a match is not required for the omitted attribute.

-no-preserve-root do not treat `/' specially (the default)

-preserve-root fail to operate recursively on `/'

-f, -silent, -quiet suppress most error messages

-reference=RFILE use RFILE's owner and group rather than the specifying OWNER:GROUP values

-R, -recursive operate on files and directories recursively

-v, -verbose output a diagnostic for every file processed

The following options modify how a hierarchy is traversed when the -R option is also specified. If more than one is specified, only the final one takes effect.

-H if a command line argument is a symbolic link to a directory, traverse it

-L traverse every symbolic link to a directory encountered

-P do not traverse any symbolic links (default)

chmod - change file access permissions

Usage

chmod [-r] permissions filenames

r Change the permission on files that are in the subdirectories of the directory that you are currently in. permission Specifies the rights that are being granted. Below is the different rights that you can grant in an alpha numeric format.filenames File or directory that you are associating the rights with Permissions

u - User who owns the file.

g - Group that owns the file.

o - Other.

a - All.

r - Read the file.

w - Write or edit the file.

x - Execute or run the file as a program.

Numeric Permissions:

CHMOD can also to attributed by using Numeric Permissions:

400 read by owner

040 read by group

004 read by anybody (other)

200 write by owner

020 write by group

002 write by anybody

100 execute by owner

010 execute by group

001 execute by anybody

ls - Short listing of directory contents

-a	list hidden files
-d	list the name of the current directory
-F	show directories with a trailing '/'
	executable files with a trailing '*'
-g	show group ownership of file in long listing

-i	print the inode number of each file
-l	long listing giving details about files and directories
-R	list all subdirectories encountered
-t	sort by time modified instead of name

cp - Copy files

cp myfile yourfile

Copy the files "myfile" to the file "yourfile" in the current working directory. This command will create the file "yourfile" if it doesn't exist. It will normally overwrite it without warning if it exists.

cp -i myfile yourfile

With the "-i" option, if the file "yourfile" exists, you will be prompted before it is overwritten.

cp -i /data/myfile

Copy the file "/data/myfile" to the current working directory and name it "myfile". Prompt before overwriting the file.

cp -dpr srcdir destdir

Copy all files from the directory "srcdir" to the directory "destdir" preserving links (-poption), file attributes (-p option), and copy recursively (-r option). With these options, a directory and all it contents can be copied to another dir

Conclusion: Hence we have implemented various file and directory handling commands.

Experiment No.03

Aim: Execution of simple C and C++ programs using CC and GCC compiler

Tools Required: Linux operating system, cc/gcc compiler.

Theory:

Step 1. Use a text editor to create the C source code.

Type the command gedit hello.c and enter the C source code below:

```
 #include main()
{
 printf("Hello World\n");
 }
```

Close the editor window.

Step 2. Compile the program.

Type the command gcc -o hello hello.c

This command will invoke the GNU C compiler to compile the file hello.c and output (-o) the result to an executable called hello.

Step34. Execute the program.

Type the command ./hello

This should result in the output

```
 Hello World
```

Conclusion: Hence we have implemented c/c++ program.

Experiment No.04

1. Aim: Create, mount & resize partition on disk.
Tools Required: Linux operating system.

Objective: To learn how to create partition, how to mount and un mount partition and how to resize a partition.

Theory:
Creating a New Partition in Linux
In most Linux systems, you can use the fdisk utility to create a new partition and to do other disk management operations.
Note: To be able to execute the commands necessary to create a new partition on Linux, you must have the root privileges.
As a tool with a text interface, fdisk requires typing the commands on the fdisk command line. The following fdisk commands may be helpful:

Options	Description
m	Displays the available commands.
p	Displays the list of existing partitions on your hda drive. Unpartitioned space is not listed.
n	Creates a new partition.
q	Exits fdisk without saving your changes.
l	Lists partition types.
w	Writes changes to the partition table.

To create a new partition on Linux
1. Start a terminal.
2. Start fdisk using the following command:
/sbin/fdisk /dev/hda
where /dev/hda stands for the hard drive that you want to partition.
3. In fdisk, to create a new partition, type the following command:
n
- When prompted to specify the Partition type, type p to create a primary partition or e to create an extended one. There may be up to four primary partitions. If you want to create more than four partitions, make the last partition extended, and it will be a container for other logical partitions.
- When prompted for the Number, in most cases, type 3 because a *typical* Linux virtual machine has two partitions by default.
- When prompted for the Start cylinder, type a starting cylinder number or press Return to use the first cylinder available.
- When prompted for the Last cylinder, press Return to allocate all the available space or specify the size of a new partition in cylinders if you do not want to use

all the available space.

By default, fdisk creates a partition with a System ID of 83. If you're unsure of the partition's System ID, use the
l
command to check it.

4. Use the
w
command to write the changes to the partition table.
5. Restart the virtual machine by entering the
reboot
command.
6. When restarted, create a file system on the new partition. We recommend that you use the same file system as on the other partitions. In most cases it will be either the Ext3 or ReiserFS file system. For example, to create the Ext3 file system, enter the following command:

```
/sbin/mkfs -t ext3 /dev/hda3
```

7. Create a directory that will be a mount point for the new partition. For example, to name it data, enter:

```
mkdir /data
```

8. Mount the new partition to the directory you have just created by using the following command:

```
mount /dev/hda3 /data
```

9. Make changes in your static file system information by editing the /etc/fstab file in any of the available text editors. For example, add the following string to this file:

```
/dev/hda3 /data ext3 defaults 0 0
```

In this string /dev/hda3 is the partition you have just created, /data is a mount point for the new partition, Ext3 is the file type of the new partition. For the exact meaning of other items in this string, consult the Linux documentation for the mount and fstab commands.

10. Save the /etc/fstab file.

Partitioning with fdisk

This section shows you how to actually partition your hard drive with the **fdisk** utility. Linux allows only 4 primary partitions. You can have a much larger number of logical partitions by sub-dividing one of the primary partitions. Only one of the primary partitions can be sub-divided.

Examples:

1. Four primary partitions
2. Mixed primary and logical partitions

5.1. fdisk usage

fdisk is started by typing (as root) **`fdisk`** ***`device`*** at the command prompt. *`device`* might be something like `/dev/hda` or `/dev/sda` .The basic **fdisk** commands you need are:

`p` print the partition table

`n` create a new partition

`d` delete a partition

`q` quit without saving changes

`w` write the new partition table and exit

Changes you make to the partition table do not take effect until you issue the write (w) command. Here is a sample partition table:

```
Disk /dev/hdb: 64 heads, 63 sectors, 621 cylinders
Units = cylinders of 4032 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hdb1   *         1       184    370912+  83  Linux
/dev/hdb2           185       368    370944   83  Linux
/dev/hdb3           369       552    370944   83  Linux
/dev/hdb4           553       621    139104   82  Linux swap
```

The first line shows the geometry of your hard drive. It may not be physically accurate, but you can accept it as though it were. The hard drive in this example is made of 32 double-sided platters with one head on each side (probably not true). Each platter has 621 concentric tracks. A 3-dimensional track (the same track on all disks) is called a cylinder. Each track is divided into 63 sectors. Each sector contains 512 bytes of data. Therefore the block size in the partition table is 64 heads * 63 sectors * 512 bytes er...divided by 1024. (See 4 for discussion on problems with this calculation.) The start and end values are cylinders.

Experiment No.05

Aim: Create user, group and assign various permissions to access a directory.

Tools Required: Linux operating system.

Objective: To learn creating a new user, group and assigning permissions to access a directory.

Theory:

To add a new user account, you can run either of the following two commands as root.

```
# adduser [new_account]

# useradd [new_account]
```

When a new user account is added to the system, the following operations are performed.

1. His/her home directory is created (/home/username by default).
2. The following hidden files are copied into the user's home directory, and will be used to provide environment variables for his/her user session.

```
.bash_logout

.bash_profile

.bashrc
```

3. A mail spool is created for the user at /var/spool/mail/username.
4. A group is created and given the same name as the new user account.

Understanding /etc/passwd

The full account information is stored in the /etc/passwd file. This file contains a record per system user account and has the following format (fields are delimited by a colon).

```
[username]:[x]:[UID]:[GID]:[Comment]:[Home directory]:[Default shell]
```

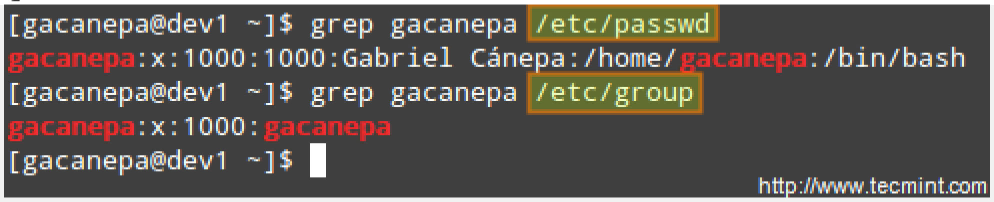
- Fields [username] and [Comment] are self explanatory.
- The x in the second field indicates that the account is protected by a shadowed password (in /etc/shadow), which is needed to logon as [username].
- The [UID] and [GID] fields are integers that represent the User IDentification and the primary Group IDentification to which [username] belongs, respectively.
- The [Home directory] indicates the absolute path to [username]'s home directory, and
- The [Default shell] is the shell that will be made available to this user when he or she logins the system.

Understanding /etc/group

Group information is stored in the /etc/group file. Each record has the following format.

```
[Group name]:[Group password]:[GID]:[Group members]
```

- [Group name] is the name of group.
- An x in [Group password] indicates group passwords are not being used.
- [GID]: same as in /etc/passwd.
- [Group members]: a comma separated list of users who are members of [Group name].



```
[gacanepa@dev1 ~]$ grep gacanepa /etc/passwd
gacanepa:x:1000:1000:Gabriel Cánepa:/home/gacanepa:/bin/bash
[gacanepa@dev1 ~]$ grep gacanepa /etc/group
gacanepa:x:1000:gacanepa
[gacanepa@dev1 ~]$
```

Add User Accounts

After adding an account, you can edit the following information (to name a few fields) using the usermod command, whose basic syntax of usermod is as follows.

```
# usermod [options] [username]
```

Setting the expiry date for an account

Use the –expiredate flag followed by a date in YYYY-MM-DD format.

```
# usermod --expiredate 2014-10-30 tecmint
```

Adding the user to supplementary groups

Use the combined -aG, or –append –groups options, followed by a comma separated list of groups.

```
# usermod --append --groups root,users tecmint
```

Changing the default location of the user’s home directory

Use the -d, or –home options, followed by the absolute path to the new home directory.

```
# usermod --home /tmp tecmint
```

Changing the shell the user will use by default

Use –shell, followed by the path to the new shell.

```
# usermod --shell /bin/sh tecmint
```

Displaying the groups an user is a member of

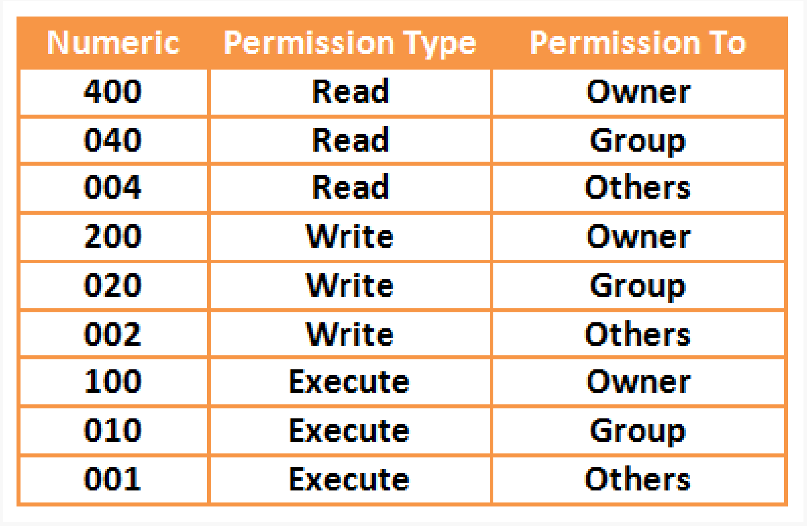
```
# groups tecmint
# id tecmint
```

there are Three basic file/directory operations that a user/group/other users can perform on the files and directories.

- **Read (r):** Permission to read the contents of the file/directory. In case of directories, a person can view all the files and sub-directories belonging to the directory.
- **Write (w):** Permission to modify the contents of the file/directory. In case of directories, a person can create a file or sub-directory in that directory.
- **Execute (x):** Permission to execute a file as a script/program. Executing a directory! Well, it does not make any sense. In case of directories, a person can enter that directory. In order to use `ls` and cd commands in /bin directory, a user should have Execute permissions.

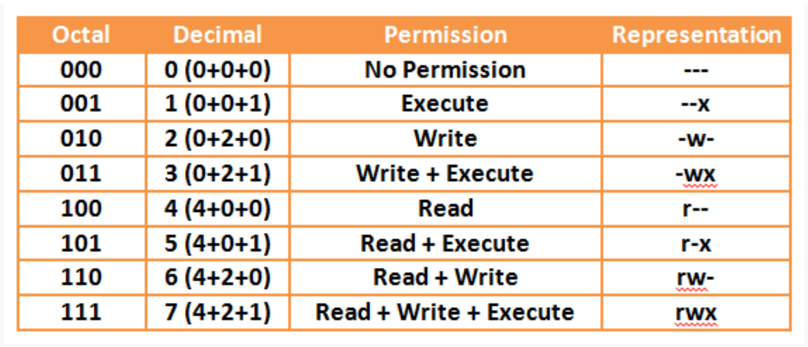
CHMOD assigns numeric values to the Read, Write and Execute permissions which are as follows:

- Read : 4
- Write : 2
- Execute : 1



Numeric	Permission Type	Permission To
400	Read	Owner
040	Read	Group
004	Read	Others
200	Write	Owner
020	Write	Group
002	Write	Others
100	Execute	Owner
010	Execute	Group
001	Execute	Others

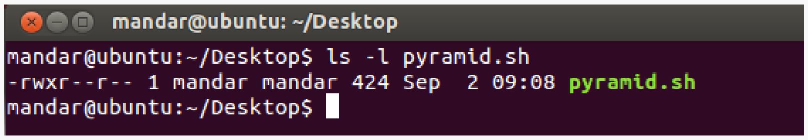
So, the permissions associated with any file/directory in Linux have a **3x3** format i.e. Three types of permissions (Read, Write and Execute) that are available for three types of users (Owner, Group and Other).



Octal	Decimal	Permission	Representation
000	0 (0+0+0)	No Permission	---
001	1 (0+0+1)	Execute	--x
010	2 (0+2+0)	Write	-w-
011	3 (0+2+1)	Write + Execute	-wx
100	4 (4+0+0)	Read	r--
101	5 (4+0+1)	Read + Execute	r-x
110	6 (4+2+0)	Read + Write	rw-
111	7 (4+2+1)	Read + Write + Execute	rwx

To observe this, just enter **`ls -l`** command that displays 9 characters for every file/directory representing the permissions for all the three types of users.

For Example:



```
mandar@ubuntu:~/Desktop$ ls -l pyramid.sh
-rwxr--r-- 1 mandar mandar 424 Sep  2 09:08 pyramid.sh
mandar@ubuntu:~/Desktop$
```

In above output,

- Owner (mandar) has Read + Write + Execute permissions.
- Group has Read permissions.
- Others have Read permissions.

Directory Permissions with CHMOD

To change the permissions associated with files and directories, you may either use *Octal Representation* (using numeric) or *Symbolic Representation* (using alphabets). We will restrict this part of our discussion up to the use of octal representation for changing files and directories permissions.

So, in octal representation of the permissions:

- First digit is for Owner
- Second digit is for Group
- Third digit is for Others

As an example, we have seen in our one of the previous articles- Getting Started with Linux Shell Scripting Language, we have used a command as **`chmod 744 helloworld.sh`**. Indirectly, we have given Read + Write + Execute (4+2+1) permissions to the Owner, Read (4) permission to the group and Read (4) permission to the others.

Now, if we wish to give Read + Write (4+2) permissions to the owner, Read (4) permissions to the group and others, then we need to enter following command:

```
chmod 644 <file_name>
```

Another example, to give Read + Execute permission (4 + 1 = 5) to user and no permission (0) to group, and Write (2) permission to others, enter following command:

```
chmod 502 <file_name>
```

UMASK, along with default permission of file/directory, is responsible for determining the final value of the default permission of a file/directory. The default permission for a file is **777** and for a directory, it is **666.** From these default permissions, the umask value is subtracted to get the final default permission for newly created files or directory. The default value of umask is **022**.

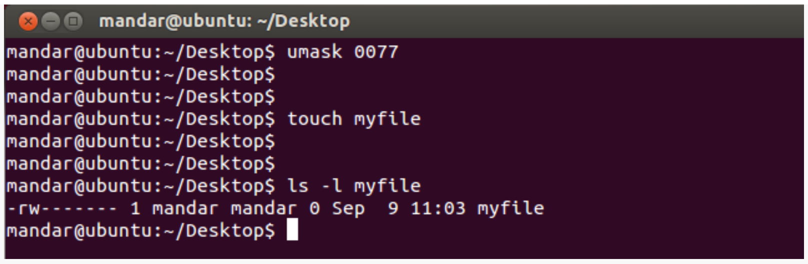
Final default permissions for file and directories are determined as follows:

- Default file permission: 666
- Default directory permission: 777
- Default umask : 022
- Final default file permission: 644
- Final default directory permission: 755

You may change the umask to an appropriate value based on your purpose. For example, if you wish no one but the owner can do anything with the file/directory then you can set umask as 0077.

```
umask 0077
```

After this action, when you make a new file/directory, the permissions associated with them will be as shown below:



```
mandar@ubuntu:~/Desktop$ umask 0077
mandar@ubuntu:~/Desktop$
mandar@ubuntu:~/Desktop$
mandar@ubuntu:~/Desktop$ touch myfile
mandar@ubuntu:~/Desktop$
mandar@ubuntu:~/Desktop$
mandar@ubuntu:~/Desktop$ ls -l myfile
-rw------- 1 mandar mandar 0 Sep  9 11:03 myfile
mandar@ubuntu:~/Desktop$
```

resentation

The symbolic representation used for three different types of users is as follows:

- **u** is used for user/owner
- **g** us used for group
- **o** is used for others

Usage of Symbolic Representation

1. Adding Single Permission

To change single permission of a specific set of users (owner, group or others), we can use **'+'** symbol to add a permission.

Syntax: **`chmod <user>+<permission> <file_name>`**

Example: `chmod u+x my_file`

Using above command, we can add Execute permission to the owner of the file.

2. Adding Multiple Permissions

This is similar to command explained above, you just need to separate those multiple permissions with a comma **(,)**.

Syntax: **`chmod <user>+<permission>,<user>+<permission> <file_name>`**

Example: `chmod g+x,o+x my_file`

Using above command, we can add Execute permissions to the group and other users of the file.

3. Removing a Permission

Removing a permission is as easy as adding a permission, just remember to use **'-'** symbol instead of **'+'**.

Syntax: **`chmod <user>-<permission> <file_name>`**

Example: `chmod o-x my_file`

Above command removes Execute permission from the other users of the file.

4. Making the Changes for All

In case we add or remove some permissions for all the users (owner, group and others), we can use a notation 'a' which denotes "All users".

Syntax: **`chmod a<+ or -><permission> <file_name>`**

Example: `chmod a+x my_file`

Above command will add Execute permission to all the users.

5. Copying the Permissions

If we wish to make permissions of two files/directories same, we can do it using `reference` option. Consider that, we want to apply permissions of myfile1 to some other file called myfile2, then use following command:

Example: `chmod --reference=myfile1 myfile2`

6. Applying Changes to All the Content of a Directory

If we want to apply some specific changes to all the files inside a directory, we can make use of option -R denoting that the operation is recursive.

Syntax: **`chmod -R <directory_name>/`**

That's enough for this article. In this article, I tried to cover the basics of files and directory permissions and the fundamental use of CHMOD command that helps us change those permissions associated with files and directories.

Conclusion: Hence we have implemented creating user, group and assigning permissions to access a directory.

Experiment No.06

Aim: Share a directory in LAN using SMB.

Tools Required: Linux operating system, SMB server.

Objective: To learn how to share a directory in LAN using SMB.

Theory:

All commands must be done as root (precede each command with 'sudo' or use 'sudo su').

1. Install Samba

```
1. sudo apt-get update
2. sudo apt-get install samba
```

2. Set a password for your user in Samba

```
1. sudo smbpasswd -a <user_name>
```

```
1. Note: Samba uses a separate set of passwords than the standard
   Linux system accounts (stored in /etc/samba/smbpasswd), so
   you'll need to create a Samba password for yourself. This
   tutorial implies that you will use your own user and it does
   not cover situations involving other users passwords, groups,
   etc...
```

```
Tip1: Use the password for your own user to facilitate.
Tip2: Remember that your user must have permission to write and
edit the folder you want to share.
Eg.:
sudo chown <user_name> /var/opt/blah/blahblah
sudo chown :<user_name> /var/opt/blah/blahblah
Tip3: If you're using another user than your own, it needs to
exist in your system beforehand, you can create it without a
shell access using the following command :
sudo useradd USERNAME --shell /bin/false

You can also hide the user on the login screen by adjusting
lightdm's configuration, in /etc/lightdm/users.conf add the
newly created user to the line :
hidden-users=
```

3. Create a directory to be shared

```
mkdir /home/<user_name>/<folder_name>
```

4. Make a safe backup copy of the original smb.conf file to your home folder, in case you make an error

```
sudo cp /etc/samba/smb.conf ~
```

5. Edit the file "/etc/samba/smb.conf"

```
sudo nano /etc/samba/smb.conf
```

```
1. Once "smb.conf" has loaded, add this to the very end of the file:
```

```

[<folder_name>]
path = /home/<user_name>/<folder_name>
valid users = <user_name>
read only = no
```

```
Tip: There Should be in the spaces between the lines, and note que
also there should be a single space both before and after each of
the equal signs.
```

6. Restart the samba:

```
sudo service smbd restart
```

7. Once Samba has restarted, use this command to check your smb.conf for any syntax errors

```
testparm
```

8. To access your network share

```
9.    sudo apt-get install smbclient
10.   # List all shares:
11.   smbclient -L //<HOST_IP_OR_NAME>/<folder_name> -U <user>
12.   # connect:
```

```
      smbclient //<HOST_IP_OR_NAME>/<folder_name> -U <user>
```

To access your network share use your username (<user_name>) and password through the path "smb://<HOST_IP_OR_NAME>/<folder_name>/" (Linux users) or "\\<HOST_IP_OR_NAME>\<folder_name>\" (Windows users). Note that "<folder_name>" value is passed in "[<folder_name>]", in other words, the share name you entered in "/etc/samba/smb.conf".

```
Note: The default user group of samba is "WORKGROUP".
```

Conclusion: Hence we have implemented how to share a directory using SMB.

Experiment No.07

Aim: Write a shell script program input-output statements and loops.

Tools required: Linux operating system.

Objective: To learn usage of input-output statements and loops in shell script.

```
# !/bin/bash
echo "enter a number"
read num
fact=1
while [ $num -ge 1 ]
do
fact=`expr  $fact\* $num`
num=’expr $num – 1’
done
echo "factorial of $n is $fact"
```

Output:

```
enter a number
4
factorial 0f 4 is 24

 total
characters :    63
words:          12
lines : 3
```

Conclusion: Hence we have implemented shell script for input-output statements and loops.

Experiment No.08

Aim: Write a shell script program using array & case statement.

Tools Required: Linux operating system.

Objective: To learn using arrays and case statement in shell script.

Theory:

```
#!/bin/sh

option="${1}"
case ${option} in
  -f) FILE="${2}"
     echo "File name is $FILE"
     ;;
  -d) DIR="${2}"
     echo "Dir name is $DIR"
     ;;
  *)
     echo "`basename ${0}`:usage: [-f file] | [-d directory]"
     exit 1 # Command to come out of the program with status 1
     ;;
esac
```

Output:

```
$./test.sh
test.sh: usage: [ -f filename ] | [ -d directory ]
$ ./test.sh -f index.htm
$ vi test.sh
$ ./test.sh -f index.htm
File name is index.htm
$ ./test.sh -d unix
Dir name is unix
```

Conclusion: Hence we have implemented array and case statement in shell script.

Experiment No.09

Aim: Use of various text processing tools: grep & sed.

Tools Required: Linux operating system.

Objective: To learn text processing using grep and sed tools.

Theory:

About grep

grep, which stands for "global regular expression print," processes text line by line and prints any lines which match a specified pattern.

grep syntax

```
grep [OPTIONS] PATTERN [FILE...]
```

Overview

Grep is a powerful tool for matching a regular expression against text in a file, multiple files, or a stream of input. It searches for the *PATTERN* of text that you specify on the command line, and outputs the results for you.

General Options

--help	Print a help message briefly summarizing command-line options, and exit.
-V, **--version**	Print the version number of **grep**, and exit.

Match Selection Options

-E, --extended-regexp	Interpret *PATTERN* as an extended regular expression (see Basic vs. Extended Regu

-F, **--fixed-strings**	Interpret *PATTERN* as a list of fixed strings, separated by newlines, any of which is to be matched.
-G, **--basic-regexp**	Interpret *PATTERN* as a basic regular expression (see Basic vs. Extended Regular Expressions). This when running **grep**.
-P, **--perl-regexp**	Interpret *PATTERN* as a Perl regular expression. This functionality is still experimental, and may pro messages.

Matching Control Options

-e *PATTERN*, **--regexp**=*PATTERN*	Use *PATTERN* as the pattern to match. This can be used to specify multiple searc beginning with a dash (**-**).
-f *FILE*, **--file**=*FILE*	Obtain patterns from *FILE*, one per line.
-i, **--ignore-case**	Ignore case distinctions in both the *PATTERN* and the input files.
-v, **--invert-match**	Invert the sense of matching, to select non-matching lines.
-w, **--word-regexp**	Select only those lines containing matches that form whole words. The test is that at the beginning of the line, or preceded by a non-word constituent character. Or, or followed by a non-word constituent character. Word-constituent characters are
-x, **--line-regexp**	Select only matches that exactly match the whole line.
-y	The same as **-i**.

About sed

sed, short for "stream editor", allows you to filter and transform text.

Description

sed is a *stream editor*. A stream editor is used to perform basic text transformations on an input stream (a file, or input from a pipeline). While in some ways similar to an editor which permits scripted edits (such as ed), **sed** works by making only one pass over the input(s), and is consequently more efficient. But it is **sed**'s ability to filter text in a pipeline which particularly distinguishes it from other types of editors.

sed syntax

```
sed OPTIONS... [SCRIPT] [INPUTFILE...]
```

If you do not specify *INPUTFILE*, or if *INPUTFILE* is "**-**", **sed** filters the contents of the standard input. The script is actually the first non-option parameter, which **sed** specially considers a script and not an input file if and only if none of the other options specifies a script to be executed (that is, if neither of the **-e** and **-f** options is specified).

Options

-n, **--quiet**, **--silent**	Suppress automatic printing of pattern space.
-e *script*, **--expression**=*script*	Add the script *script* to the commands to be executed.
-f *script-file*, **--file**=*script-file*	Add the contents of *script-file* to the commands to be executed.
--follow-symlinks	Follow symlinks when processing in place.
-i[*SUFFIX*], **--in-place**[=*SUFFIX*]	Edit files in place (this makes a backup with file extension *SUFFI*
-l *N*, **--line-length**=*N*	Specify the desired line-wrap length, *N*, for the "**l**" command.
--POSIX	Disable all GNU extensions.
-r, **--regexp-extended**	Use extended regular expressions in the script.
-s, **--separate**	Consider files as separate rather than as a single continuous long s
-u, **--unbuffered**	Load minimal amounts of data from the input files and flush the o
--help	Display a help message, and exit.
--version	Output version information, and exit.

About sed Programs

A **sed** program consists of one or more **sed** commands, passed in by one or more of the **-e**, **-f**, **--expression**, and **--file** options, or the first non-option argument if none of these options are used. This documentation frequently refers to "the" **sed** script; this should be understood to mean the in-order catenation of all of the scripts and script-files passed in.
Commands within a script or script-file can be separated by semicolons ("**;**") or newlines(ASCII code 10). Some commands, due to their syntax, cannot be followed by semicolons working as command separators and thus should be terminated with newlines or be placed at the end of a script or script-file. Commands can also be preceded with optional non-significant whitespace characters.
Each **sed** command consists of an optional address or address range (for instance, line numbers specifying what part of the file to operate on; see Selecting Lines for details), followed by a one-character command name and any additional command-specific code.

How sed Works

sed maintains two data buffers: the active *pattern space*, and the auxiliary *hold space*. Both are initially empty.
sed operates by performing the following cycle on each line of input: first, **sed** reads one line from the input stream, removes any trailing newline, and places it in the pattern space. Then commands are executed; each command can have an address associated to it: addresses are a kind of condition code, and a command is only executed if the condition is verified before the command is to be executed.
When the end of the script is reached, unless the **-n** option is in use, the contents of pattern space are printed out to the output stream, adding back the trailing newline if it was removed. Then the next cycle starts for the next input line.
Unless special commands (like '**D**') are used, the pattern space is deleted

between two cycles. The hold space, on the other hand, keeps its data between cycles (see commands **'h'**, **'H'**, **'x'**, **'g'**, **'G'** to move data between both buffers).

Selecting Lines With sed

Addresses in a **sed** script can be in any of the following forms:

number	Specifying a line number will match only that line in the input. (Note that sed counts lines c unless **-i** or **-s** options are specified.)
first~step	This GNU extension of **sed** matches every *step* lines starting with line *first*. In particular, lin non-negative *n* such that the current line-number equals *first* + (*n* * *step*). Thus, to select the use **1~2**; to pick every third line starting with the second, '**2~3**' would be used; to pick every '**10~5**'; and '**50~0**' is just another way of saying **50**.
$	This address matches the last line of the last file of input, or the last line of each file when th
/regexp/	This will select any line which matches the regular expression *regexp*. If *regexp* itself incluc be escaped by a backslash ("\"). The empty regular expression '//' repeats the last regular expression match (the same holds passed to the s command). Note that modifiers to regular expressions are evaluated when th it is invalid to specify them together with the empty regular expression.
\%regexp%	(The % may be replaced by any other single character.) This also matches the regular expression *regexp*, but allows one to use a different delimiter useful if the *regexp* itself contains a lot of slashes, since it avoids the tedious escaping of eve delimiter characters, each must be escaped by a backslash ("\").
*/regexp/***I** *\%regexp%***I**	The **I** modifier to regular-expression matching is a GNU extension which causes the *regexp* opposed to case-sensitive) manner.
*/regexp/***M**	The **M** modifier to regular-expression matching is a GNU **sed** extension which causes ^ anc

*\%regexp%***M**	the normal behavior) the empty string after a newline, and the empty string before a newline. There are speci ("\`" and "\'") which always match the beginning or the end of the buffer. **M** stands for multi-line.

If no addresses are given, then all lines are matched; if one address is given, then only lines matching that address are matched.
An address range can be specified by specifying two addresses separated by a comma ("**,**"). An address range matches lines starting from where the first address matches, and continues until the second address matches (inclusively).
If the second address is a *regexp*, then checking for the ending match will start with the line following the line which matched the first address: a range will always span at least two lines (except of course if the input stream ends).
If the second address is a number less than (or equal to) the line matching the first address, then only the one line is matched.
GNU **sed** also supports some special two-address forms; all these are GNU extensions:

0,*/regexp/*	A line number of **0** can be used in an address specification like **0,***/regexp/* so that **sed**will try to i In other words, **0,***/regexp/* is similar to **1,***/regexp/*, except that if *addr2* matches the very first lin consider it to end the range, whereas the **1,***/regexp/* form will match the beginning of its range a second occurrence of the regular expression. Note that this is the only place where the **0** address makes sense; there is no "0th" line, and com any other way will give an error.
*addr1***,+***N*	Matches *addr1* and the *N* lines following *addr1*.
*addr1***,~***N*	Matches *addr1* and the lines following *addr1* until the next line whose input line number is a m

Appending the **!** character to the end of an address specification negates the sense of the match. That is, if the **!** character follows an address range, then only lines which do not match the address range will be selected. This also works for singleton addresses, and, perhaps perversely, for the null address.

Overview Of Regular Expression Syntax

To know how to use **sed**, you should understand regular expressions ("regexp" for short). A regular expression is a pattern that is matched against a subject string from left to right. Most characters are ordinary: they stand for themselves in a pattern, and match the corresponding characters in the subject. As a simple example, the pattern

```
The quick brown fox
```

...matches a portion of a subject string that is identical to itself. The power of regular expressions comes from the ability to include alternatives and repetitions in the pattern. These are encoded in the pattern by the use of special characters, which do not stand for themselves but instead are interpreted in some special way. Here is a brief description of regular expression syntax as used in **sed**:

char	A single ordinary character matches itself.
*****	Matches a sequence of zero or more instances of matches for the preceding regular expre character, a special character preceded by "\", a "**.**", a grouped regexp (see below), or a b extension, a postfixed regular expression can also be followed by "*****"; for example, **a**** 2001 says that ***** stands for itself when it appears at the start of a regular expression or su implementations do not support this, and portable scripts should instead use "\*" in these
\\+	Like *****, but matches one or more. It is a GNU extension.
?	Like *****, but only matches zero or one. It is a GNU extension.
{*i*\**}**	Like *****, but matches exactly *i* sequences (*i* is a decimal integer; for compatibility, you sh inclusive).
{*i***,***j*\**}**	Matches between *i* and *j*, inclusive, sequences.
{*i*,**}**	Matches more than or equal to *i* sequences.

\\(*regexp*\\)	Groups the inner regexp as a whole; this is used to: • Apply postfix operators, like **\\(abcd\\)***: this will search for zero or more whole sequences of '**ab** while **abcd***would search for '**abc**' followed by zero or more occurrences of '**d**'. Note that supp required by POSIX 1003.1-2001, but many non-GNU implementations do not support it and hen portable. • Use back references (see below).
.	Matches any character, including a newline.
^	Matches the null string at beginning of the pattern space, i.e. what appears after the ^ must appear at the pattern space. In most scripts, pattern space is initialized to the content of each line. So, it is a useful simplification to t matching only lines where '**#include**' is the first thing on line—if there are spaces before, for example, t simplification is valid as long as the original content of pattern space is not modified, for example with a ^ acts as a special character only at the beginning of the regular expression or subexpression (that is, afte scripts should avoid ^ at the beginning of a subexpression, though, as POSIX allows implementations th ordinary character in that context.
$	It is the same as ^, but refers to end of pattern space. **$** also acts as a special character only at the end of t or subexpression (that is, before \**)** or \**|**), and its use at the end of a subexpression is not portable.
[*list***]** **[**^*list***]**	Matches any single character in *list*: for example, **[aeiou]** matches all vowels. A *list* may include sequen which matches any character between *char1* and *char2*. For example, **[b-e]** matches any of the character A leading ^ reverses the meaning of *list*, so that it matches any single character not in *list*. To include **]** i first character (after the ^ if needed); to include **-** in the list, make it the first or last; to include ^ put it af The characters **$**, *****, **.**, **[**, and \\ are normally not special within *list*. For example, **[\*]** matches either '\\' or special here. However, strings like **[.ch.]**, **[=a=]**, and **[:space:]** are special within *list* and represent collat equivalence classes, and character classes, respectively, and **[** is therefore special within list when it is fo Also, when not in **POSIXLY_CORRECT** mode, special escapes like \n and \tare recognized within *list* information.

regexp1|*regexp2*	Matches either *regexp1* or *regexp2*. Use parentheses to use complex alternative regular expressions. The each alternative in turn, from left to right, and the first one that succeeds is used. This option is a GNU ex
regexp1regexp2	Matches the concatenation of *regexp1* and *regexp2*. Concatenation binds more tightly than **|**, ^, and **$**, bu other regular expression operators.
\digit	Matches the *digit*-th **\\(**...**\\)** parenthesized subexpression in the regular expression. This option is called a b Subexpressions are implicitly numbered by counting occurrences of **\\(** left-to-right.
\\n	Matches the newline character.
\char	Matches *char*, where *char* is one of **$**, *****, **.**, **[**, \\, or ^. Note that the only C-like backslash sequences that y assume to be interpreted are **\\n** and \\\\; in particular **\\t** is not portable, and matches a '**t**' under most implem rather than a tab character.

Note that the regular expression matcher is greedy, i.e., matches are attempted from left to right and, if two or more matches are possible starting at the same character, it selects the longest.
For example:

abcdef	Matches "**abcdef**".
a*b	Matches zero or more "**a**" characters, followed by a single "**b**". For example, "**b**" or "**aaaaaaa**
a\\?b	Matches "**b**" or "**ab**".
a\\+**b**\\+	Matches one or more "**a**" characters followed by one or more "**b**"s. "**ab**" is the shortest possib "**aaaaab**", "**abbbbbb**", or "**aaaaaabbbbbbb**".
.*or **.**\\+	Either of these expressions will match all of the characters in a non-empty string, but only **.***
^**main.*(.*)**	This matches a string starting with "**main**", followed by an opening and closing parenthesis. T adjacent.
^#	This matches a string beginning with "#".
\\\\$	This matches a string ending with a single backslash. The regexp contains two backslashes fo

\$	This matches a string consisting of a single dollar sign.
[a-zA-Z0-9]	In the C locale, this matches any ASCII letters or digits.
[^ *tab*]\+	(Here *tab* stands for a single tab character.) This matches a string of one or more characters, none of which is Usually this means a word.
^\(.*\)\n\1$	This matches a string consisting of two equal substrings separated by a newline.
.\{9\}A$	This matches nine characters followed by an '**A**'.
^.\{15\}A	This matches the start of a string that contains 16 characters, the last of which is an '**A**'.

Conclusion: Hence we have implemented text tools- grep and sed.

Experiment No.10

Aim: Write a program in AWK using loops.

Tools Required: Linux operating system.
Objective: To learn how to create AWK program and awk program using loop.

Theory:

```
$ awk 'BEGIN {
   sum = 0; for (i = 0; i < 20; ++i) {
      sum += i; if (sum > 50) exit(10); else print "Sum =", sum
   }
}'
Output:
```

```
Sum = 0
Sum = 1
Sum = 3
Sum = 6
Sum = 10
Sum = 15
Sum = 21
Sum = 28
Sum = 36
Sum = 45
```

Conclusion: Hence we have implemented awk program using loop.