Use cases:symbolic link

- When you want to move the physical file location without breaking the existing or common "well-known" references, you may want to create a symbolic link at the old location.

  - Create another copy of file somewhere and modify it causing different inode to be assigned to new copy.
  - Hard links now point to old obsolete copy which is invalid.
  - Soft links are dangling only if path of the file is changed but it is fixable.

  Solution:
  - Create a soft link at old file location with old file name pointing to new file.
    This way all existing soft links stay valid while hard links stay broken.

- This is useful when you try to upgrade a commonly used tool with a private version to serve several needs. In certain Linux distributions, such as OpenWRT, the /bin/bash, /bin/ls, /bin/mv, etc. are all links to /bin/busybox. This binary is capable to behave differently based on the argv[0] used.
  - Suppose hard links are used to point to /bin/busybox as they are faster.
  - If new implementation of busybox is installed causing inode change, hard links will become invalid and point to older implementation.

  - This problem ishandled by creating a soft link at old location pointing to new implementation.

Use cases:symbolic link

- Making applications easily accessible via the command line.

```
$ echo $PATH |cut -d ":" -f 7
/usr/bin
$ sudo ln -s /home/rekha/workingDir/shell/array.sh /usr/bin/test-link
$ ls -l /usr/bin/test-link
lrwxrwxrwx 1 root root 37 Aug  8 20:45 /usr/bin/test-link ->
/home/rekha/workingDir/shell/array.sh
$ cd ~
$ test-link
one two three four array describe array
```

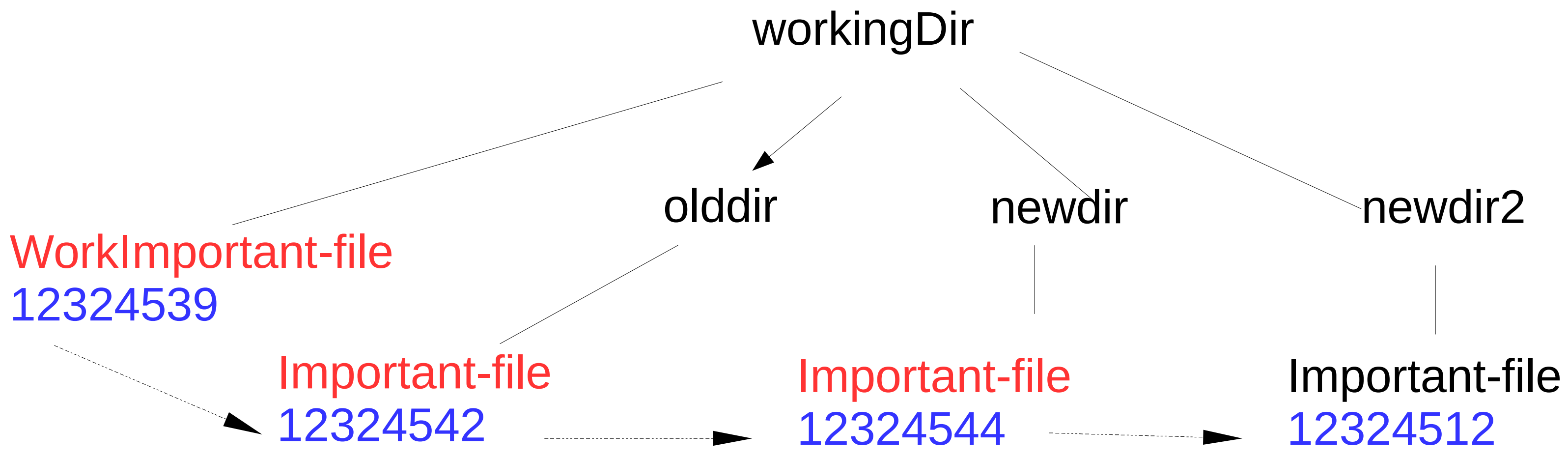
- Hard links must be on the same device and don't require additional disk space. Useful when dealing with photos, videos and audio files. To find all hardlinks to an inode, use following command.

```
        $ find ~/ -xdev -inum 12324494   2>/dev/null
```

- Symbolic links are used heavily in /etc. It enables files located in other places in the hierarchy to be used without duplication. Symbolic links can cross device boundaries, but require additional disk space for the pointer.

Use cases:chain of soft links

```
rekha@rekha-ThinkPad-P50:~/workingDir$ ls -il workImportant-file
12324539 lrwxrwxrwx 1 rekha rekha 21 Aug  8 17:41 workImportant-file -> olddir/important-file
rekha@rekha-ThinkPad-P50:~/workingDir$ ls -il olddir/important-file
12324542 lrwxrwxrwx 1 rekha rekha 24 Aug  8 17:42 olddir/important-file -> ../newdir/important-file
rekha@rekha-ThinkPad-P50:~/workingDir$ ls -il newdir/important-file
12324544 lrwxrwxrwx 1 rekha rekha 25 Aug  8 18:23 newdir/important-file -> ../newdir2/important-file
rekha@rekha-ThinkPad-P50:~/workingDir$ ls -il newdir2/important-file
12324512 -rw-rw-r-- 1 rekha rekha 47 Aug  8 17:31 newdir2/important-file
rekha@rekha-ThinkPad-P50:~/workingDir$ cat workImportant-file
this is an important file
.
adding more stuff.
```



Remote login/network (secure shell)

- SSH, or Secure Shell, is a protocol used to securely log onto remote systems.
- Secure Shell (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network.
- ssh daemon (sshd) must be listening on some port (often port 22).
- Remote machine be configured to accept incoming SSH connections.
- Can be used to execute remote commands.
- Common options: -X/-Y (imports X11 - graphical window), -f (puts ssh into the background before executing the remote command).
- There are ssh clients available for most operating systems.

```
ssh [options] [username@]<remote-machine-name/IP address>
ssh [options] [username@]<remote-machine-name/IP address> <command>
```

```
ssh user@10.3.1.168
ssh -X user@10.3.1.168 firefox  (run Firefox remotely)
```

Remote Secure File Transfer

sftp (Secure File Transfer Protocol):

```
sftp username@remote machine
sftp user@10.3.1.168
```

- Transfers files between local and remote machines securely.
- Uses an interactive console.
- Same connection settings as ssh.
- Common commands include:

  help

  get - download from remote machine

  get remote-path-to-file local-path-to-file

  put - upload to remote machine

  put local-path-to-file remote-path-to-file

  cd, pwd, ls - (on remote machine)

  lcd, lpwd, lls - (on local machine)

  exit – to exit from sftp program

Remote Secure File Copy

EXPORT:

- copies <file> to the remote machine over an encrypted channel.
- Notice the colon (:) It is necessary.
- Below command reads as “copy local_file” into “remote_file” at “remote_host” using “user account”.

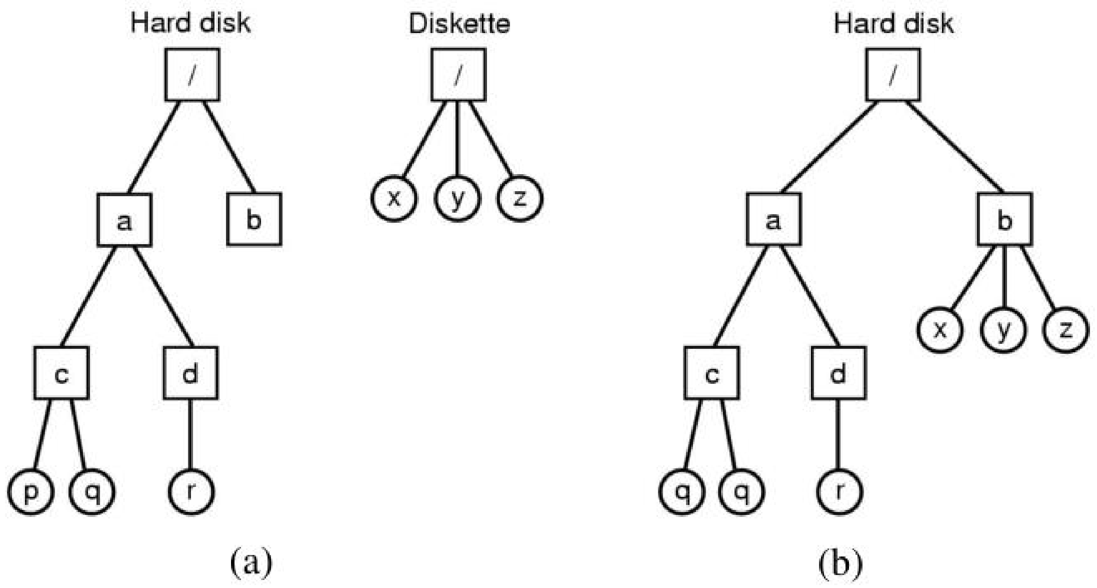
```
#Upload: local ->remote
scp [-r] local_file user@remote_host:remote_file
scp list user@10.3.1.168:
```

IMPORT:

```
# download: remote -> local
scp user@remote_host:remote_file local_file
scp user@10.3.1.168:\* .
```

File Systems - Mounting

- File systems are “mounted” at an existing directory on the current tree, if there are any files in that directory they are inaccessible until the file system is un-mounted
- To access the data, user must have to attach the partition to some mount point directory.(by default /media or /mnt Directory exists for this purpose)
- Both local and remote file systems can be mounted in the same way, this makes access to networked file systems transparent
- Unix can serve as both a server and client for network file system access



(a) Before mounting. (b) After mounting

File Systems – Mount/unmount

- mount: To make the filesystem visible to the users and the operating system

  ```
  mount  -t   type <Device to be mount>    <mount point directory>
  mount /dev/sda1 /media/rekha
  mount | grep /media/rekha
  ```

- umount:
  - To safely detach the filesystem from its mount point directory tree.
  - usually when we need to shrink a filesystem size or running the filesystem check(fsck) to check the integrity test on the partition it should be on inactive status.

```
umount  <mount point directory>
umount  /dev/sda1
mount | grep /dev/sda1
```

File Systems – Mount/unmount

- To check the mounted partitions, you can use mount command. The kernel always track the mounted filesystem looking at /etc/mtab file.
- Try df -hT and fdisk -l

```
# /dev/sda1 does not show up in the mount table
rekha@rekha-ThinkPad-P50:~$ mount |grep /dev/sda1
```

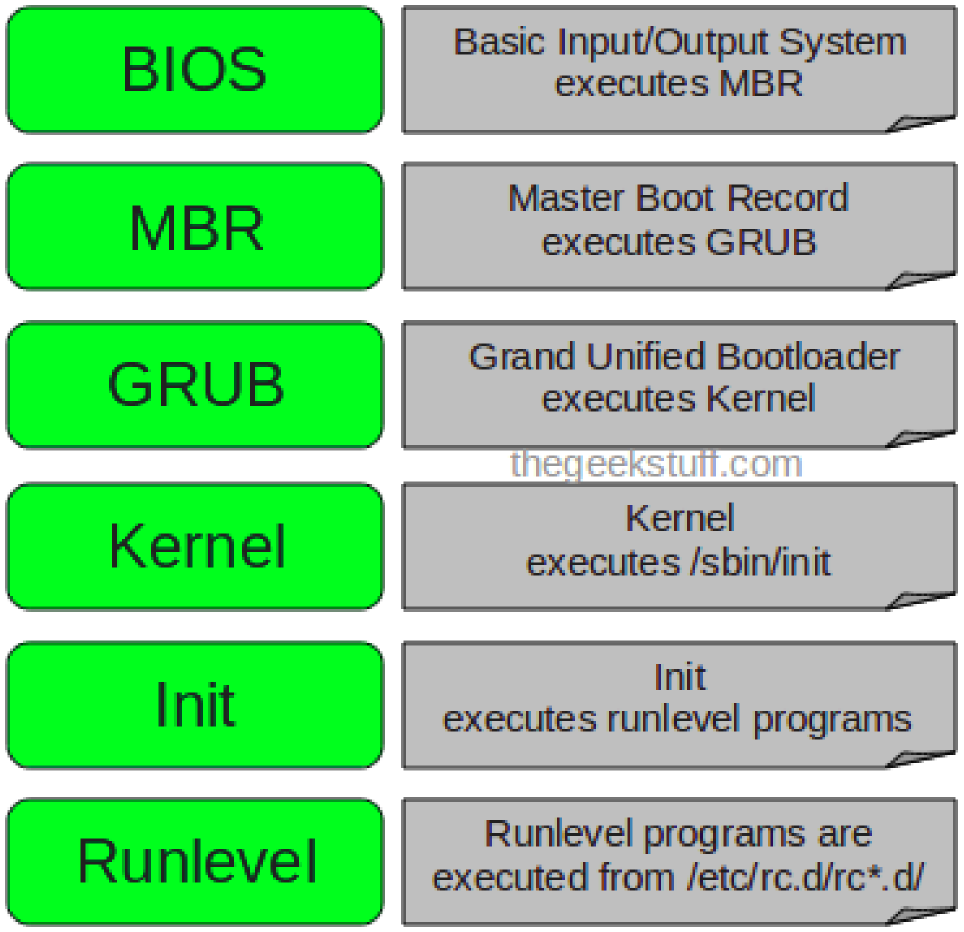
- To check the unmounted partitions, you can use blkid
  $blkid -o list | grep "not mounted"

```
#sda1 shows in "not mounted" list.
rekha@rekha-ThinkPad-P50:~$ blkid -o list|grep "not mounted"
          ntfs    Windows7_OS (not mounted) 64A2ABE6A2ABBAC6
          ntfs    Lenovo_Recovery (not mounted) D470B80C70B7F2FA
          ntfs    Windows RE (not mounted) 94681920681902A2
/dev/sda1  ntfs    Data2    (not mounted)  E666BF8966BF58CF
                   (not mounted)
                   (not mounted)
```

File Systems(demo)

```
# mount /dev/sda1  on /media/rekha
rekha@rekha-ThinkPad-P50:~$ sudo mount /dev/sda1 /media/rekha
[sudo] password for rekha:
#files on /dev/sda1 are accessible from file system at /media/rekha
rekha@rekha-ThinkPad-P50:~$ ls -ld /media/rekha/*
drwxrwxrwx 1 root root    0 Jul  4  2017 /media/rekha/MININT
drwxrwxrwx 1 root root 4096 Jul  4  2017 /media/rekha/$RECYCLE.BIN
drwxrwxrwx 1 root root    0 Jul 17  2017 /media/rekha/System Volume Information
-rwxrwxrwx 1 root root   55 Aug  9 12:01 /media/rekha/test
rekha@rekha-ThinkPad-P50:~$ more /media/rekha/test
can you see contents of this test file after mounting?
#/dev/sda1 shows up in the mount table
rekha@rekha-ThinkPad-P50:~$ mount |grep /dev/sda1
/dev/sda1 on /media/rekha type fuseblk
(rw,relatime,user_id=0,group_id=0,allow_other,blksize=4096)
#/dev/sda1 does not show up in the list of unmounted partitions
rekha@rekha-ThinkPad-P50:~$ blkid -o list|grep "not mounted"
        ntfs    Windows7_OS (not mounted) 64A2ABE6A2ABBAC6
        ntfs    Lenovo_Recovery (not mounted) D470B80C70B7F2FA
        ntfs    Windows RE (not mounted) 94681920681902A2
                    (not mounted)
                    (not mounted)
rekha@rekha-ThinkPad-P50:~$ sudo umount /dev/sda1
```

Boot Process


BIOS
Basic Input/Output System executes MBR
MBR
Master Boot Record executes GRUB
GRUB
Grand Unified Bootloader executes Kernel
thegeekstuff.com
Kernel
Kernel executes /sbin/init
Init
Init executes runlevel programs
Runlevel
Runlevel programs are executed from /etc/rc.d/rc*.d/

BIOS:
PowerOnSelfTest, checks for system integrity, located in the motherboard in a small, rewritable memory chip
Invokes the initial bootstrap program from MBR

MBR:
Ist sector of bootable disk typically /dev/hda, or /dev/sda
512 bytes
Stores partition table and info on GRUB
Looks for bootable partition and invokes GRUB code there.

GRUB:
Let you choose which kernel to boot
/boot/grub/grub.cfg
Loads linux kernel and initrd images
Starts the kernel boot.

Boot Process: Initialization

Kernel:

initrd(initial RAM disk) is temporary filesystem which has compiled drivers
used to access hardware until kernel is fully booted.
Mounts root file system.
Runs /sbin/init to start the first process in the system

Init:

- The init is a daemon process which starts as soon as the computer starts and continue running till, it is shutdown.
- Init is the first process that starts when a computer boots, making it the parent of all other running processes directly or indirectly. Typically it is assigned “pid=1“.
- Init identifies the default initlevel and uses that to load all appropriate program.

Boot Process: Run Level

Depending on your default init level setting, the system will execute the programs from one of the following directories.

Halt:	Run level 0 – /etc/rc0.d/
Single User Mode:	Run level 1 – /etc/rc1.d/
MultiUser(without NFS):	Run level 2 – /etc/rc2.d/
Full MultiUser:	Run level 3 – /etc/rc3.d/
Unused:	Run level 4 – /etc/rc4.d/
X11:	Run level 5 – /etc/rc5.d/
Reboot:	Run level 6 – /etc/rc6.d/

Programs starts with S are used during startup. S for startup.

Programs starts with K are used during shutdown. K for kill.

Programming in Unix:Compiling

- Unix made by programmer for programming
- gcc compiler – for ‘c’, g++ for ‘c++’
- Various options, -O,-c,-g,-I

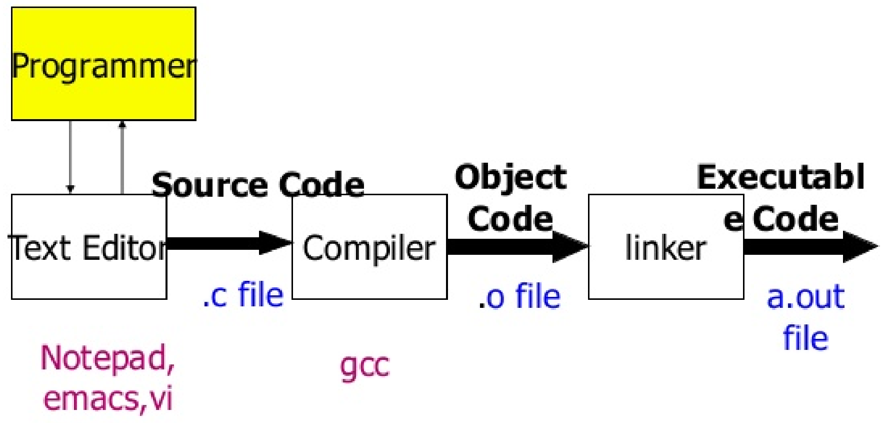
‘-O’ sets optimization level

‘-c’ only compile not link

‘-g’ for debug

‘-I’ for pre-processing only

Compiled Languages



Programming in Unix:Debugging

- The C language is the standard language of Unix
  - Most vendors have proprietary compilers
  - GNU gcc is available on almost all Unix-like systems and many other OS's
- There are numerous other compilers available for Unix systems; C++, Fortran, Java

- gdb Debugger

```
$gcc -g fork-demo.c

$gdb a.out
```

makefile

Makefile

```
all : fork-demo.obj
fork-demo.obj : fork-demo.c
	gcc -o fork-demo fork-demo.c
```

```
$make
```

1. call make to build the programs specified in makefile

2. first line is rule -> target on left of : and depandencies on right of :

3. second line is command -> command to be executed to build target

4. target needs to be built only if timestamp is changed on any of the depandencies

5. Command line should be tabbed

6. ‘make’ will execute target given by ‘all’ or first target,

   else specify your target in command line

Makefile with variables

```
makefile-vars
EXE = fork-demo

${EXE} : ${EXE}.c
	gcc -o ${EXE} ${EXE}.c

clean:
	rm  ${EXE}
```

```
$make -f makefile-vars

$make -f makefile-vars clean
```

1. Variables can be defined in makefile

2. Variables can be accessed using ${var} notation

3. Use -f option if makefile has name other than “makefile”

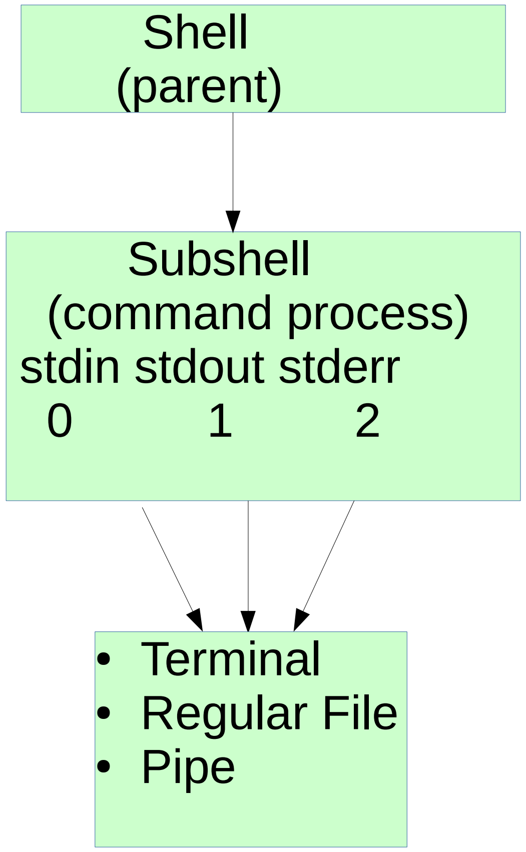
4. More than one target can be specified in makefile. If not explicitly specified, all or first target is considered to be built.

5. Target can be specified on command line.

Shells - Starting a Program

The steps a shell takes to run a program

- shell reads the command name
- shell executes a *fork* system call, clones itself,
- the parent shell waits for the child shell process to terminate
- Child shell sets stdin,stdout,stderr for redirection and piping as specified in the command
- Child shell locates the program file
- the child shell uses the *exec* system call to load the program over itself
- the loaded program starts executing
- when the program terminates the child process terminates
- the kernel informs the waiting parent shell
- the waiting shell program resumes execution and prints a prompt



Shell (parent)
Subshell (command process)
stdin stdout stderr
0 1 2
• Terminal
• Regular File
• Pipe