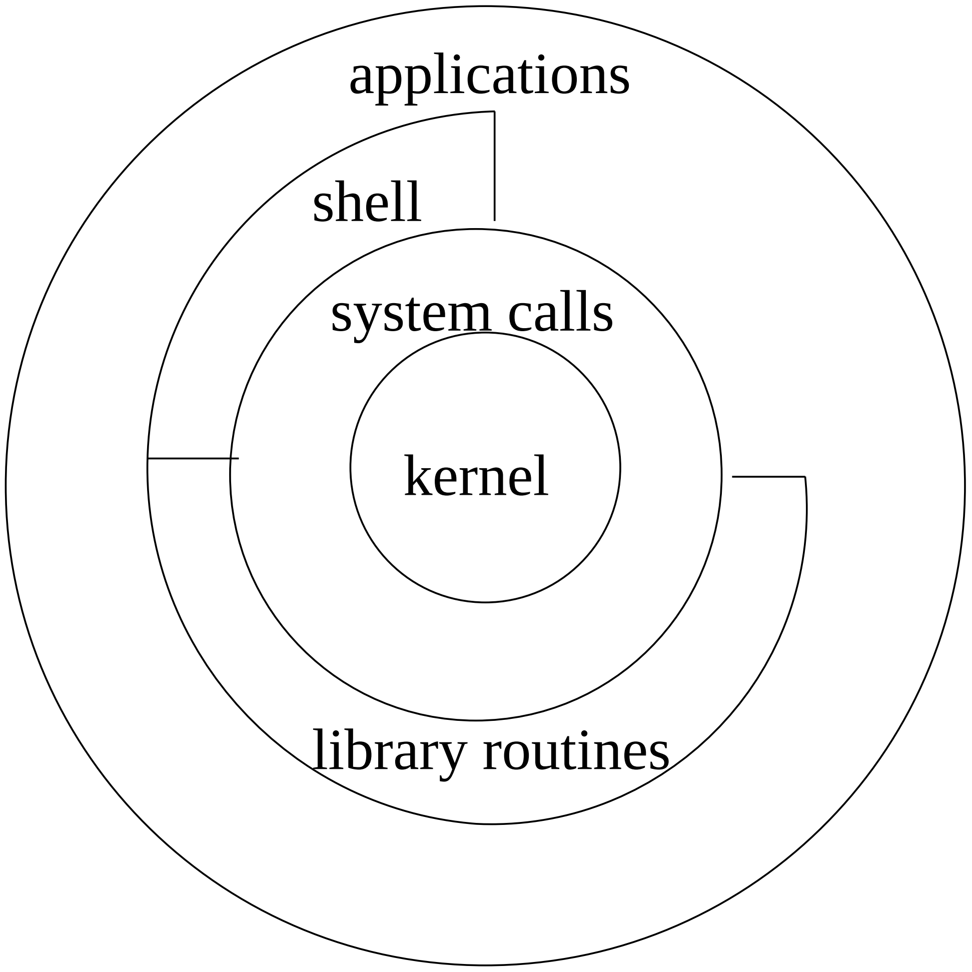
## Unix System Architecture



## Unix system implementations

- UNIX system V Release 4
- 4.4 BSD
- FreeBSD
- GNU Linux
- Mac OS X
- Solaris

## Unix standardization–C standard

ISO C standard not only defines the syntax and semantics of the language, but also a standard library. All comptemporary UNIX system provide the library routines that are specified in the C standard.

ISO C library can be divided into 24 areas based on the header files defined by the standard. The header files are listed in the following table.

| | |
|---|---|
| `<assert.h>` | Diagnostics |
| `<complex.h>` | Complex |
| `<ctype.h>` | Character handling |
| `<errno.h>` | Errors |
| `<fenv.h>` | Floating-point environment & |
| `<float.h>` | Characteristics of oating types |
| `<inttypes.h>` | Format conversion of integer types |
| `<iso646.h>` | Alternative spellings |
| `<limits.h>` | Sizes of integer types |
| `<locale.h>` | Localization |
| `<math.h>` | Mathematics |
| `<setjmp.h>` | Nonlocal jumps |
| `<signal.h>` | Signal handling |
| `<stdarg.h>` | Variable arguments |
| `<stdbool.h>` | Boolean type and values |
| `<stddef.h>` | Common denitions |
| `<stdint.h>` | Integer types |

| | |
|---|---|
| `<stdio.h>` | `Input/output` |
| `<stdlib.h>` | `General utilities` |
| `<string.h>` | `String handling` |
| `<tgmath.h>` | `Type-generic math` |
| `<time.h>` | `Date and time` |
| `<wctype.h>` | `Wide character classication and mapping utilities` |

**Note**: Linux contains all the header files, while other Unix systems may not.

## Unix standardization–POSIX

POSIX(Portable Operating System Interface) is a family of standards developed by the IEEE. Currently POSIX documentation is divided in three parts:

- POSIX Kernel APIs (which include extensions for POSIX.1, Real-time Services, Threads Interface, Real-time Extensions, Security Interface, Network File Access and Network Process-to-Process Communications)
- POSIX Commands and Utilities (with User Portability Extensions, Corrections and Extensions, Protection and Control Utilities and Batch System Utilities)
- POSIX Conformance Testing

Upgrades to POSIX standard

- POSIX.1, Core Services (incorporates Standard ANSI C)
- POSIX.1b, Real-time extensions
- POSIX.1c, Threads extensions

## Unix standardization–Single Unix Specification

Single UNIX Specification (SUS) is the collective name of a family of standards for computer operating systems to qualify for the name "Unix". The user and software interfaces to the OS are specified in four main sections:

- Base Definitions - a list of definitions and conventions used in the specifications and a list of C header files which must be provided by compliant systems.
- Shell and Utilities - a list of utilities and a description of the shell
- System Interfaces - a list of available C system calls which must be provided.
- Rationale - the explanation behind the standard.

The standard user command line and scripting interface is the Bourne Shell. Other user-level programs, services and utilities include awk, echo, ed, and numerous (hundreds) others. Required program-level services include basic I/O (file, terminal, and network) services.

website: http://www.unix.org/online.html

## Linux Standard Base

The Linux Standard Base, or LSB, is a joint project by several GNU/Linux distributions under the organizational structure of The Free Standards Group to standardize the internal structure of Linux-based operating systems. The LSB is based on the POSIX specification, the Single UNIX Specification, and several other open standards, but extends them in certain areas.

**Latest release**: 3.1: Released 2005-10-31. This version has been submitted as ISO/IEC 23360.

**Website**: http://www.linuxbase.org/

## Unix logging in

- **Password file** When we log in to a Unix system, we enter our usernames, password. The system then looks up our login name in its password file, usually the file /etc/passwd. A typical line will be like this:

  ```
  wch:x:500:100:Changhua Wu:/home/wch:/bin/bash
  ```

- **Shells** When we log in, a shell program will be launched for us to enter commands. Shell usually accetps commands from the terminal/console or from script files. The five five commonly-used shells are listed below:

| Name | Path | FreeBSD | Linux | Mac OS | Solaris |
|---|---|---|---|---|---|
| Bourne shell | /bin/sh | · | link to bash | link to bash | · |
| Bourne-again shell | /bin/bash | optional | · | · | · |
| C shell | /bin/csh | link to tcsh | link to tcsh | link to tcsh | · |
| Korn shell | /bin/ksh | | | | · |
| TENEX C shell | /bin/tcsh | · | · | · | · |

For more information, such as the history about the shells, visit

`http://en.wikipedia.org/wiki/Unix_shell`

## Linux/Unix Shell

The file /etc/shells gives an overview of known shells on a Linux system:

```
mia:~> cat /etc/shells

/bin/bash

/bin/sh

/bin/tcsh

/bin/csh
```

The default shell is set in the /etc/passwd file, like this line for user mia:

```
mia:*:Mia Maya:/home/mia:/bin/bash
```

To switch from one shell to another, just enter the name of the new shell in the active terminal.

```
mia:~> tcsh
```

## Unix file system

On a Unix/Linux system, everything is a file, if something is not a file, then it is a process.

- No difference between a file and a directory
- Programs, services, texts, images, and so forth, are all files
- Named pipes and sockets are special files
- Input and output devices, and generally all devices, are considered to be files

## Sorts of files

- **Regular files**: most files are regular files.
- **Directories**: files that are lists of other files.
- **Special files**: the mechanism used for input and output. Most special files are in /dev, we will discuss them later.
- **Links**: a system to make a file or directory visible in multiple parts of the system's file tree. We will talk about links in detail.
- **(Domain) sockets**: a special file type, similar to TCP/IP sockets, providing inter-process networking protected by the file system's access control.
- **Named pipes**: act more or less like sockets and form a way for processes to communicate with each other, without using network socket semantics.

## Information of file types from ls

The -l option to ls displays the file type, using the first character of each input line:

```
drwxr-xr-x 2 wch users 4096 2005-08-01 22:33 Desktop
drwxr-xr-x 9 wch users 4096 2005-08-07 14:44 Document
-rwxr-xr-x 1 wch users 329737 2005-02-03 22:39 dscf0033.jpg
lrwxr-xr-x 1 wch users 47 2005-02-28 13:51 mdiqttest ->
/home/wch/project/mdiqttest/mdiqttest/mdiqttest
```

Symbols used by ls for file types

- -: regular file, d: Directory
- l: link, c: special file
- s: Socket, p: Name pipe
- b: Block device

## Disk Partitioning

Why partition?

- to achieve higher data security in case of disaster.

Partition layout and types

- data partition: normal Linux system data, including the root partition containing all the data to start up and run the system. Systems in mixed environments may contain partitions for other system data, such as a partition with a FAT or VFAT file system for MS Windows data or relatively new Reiser file system.
- swap partition: expansion of the computer's physical memory, extra memory on hard disk. Swap space (indicated with swap) is only accessible for the system itself, and is hidden from view during normal operation.

## Disk Partitioning

Common data partitions

- a partition for user programs (/usr)
- a partition containing the users' personal data (/home)
- a partition to store temporary data like print- and mail-queues (/var)
- a partition for third party and extra software (/opt)

## Disk Partitioning

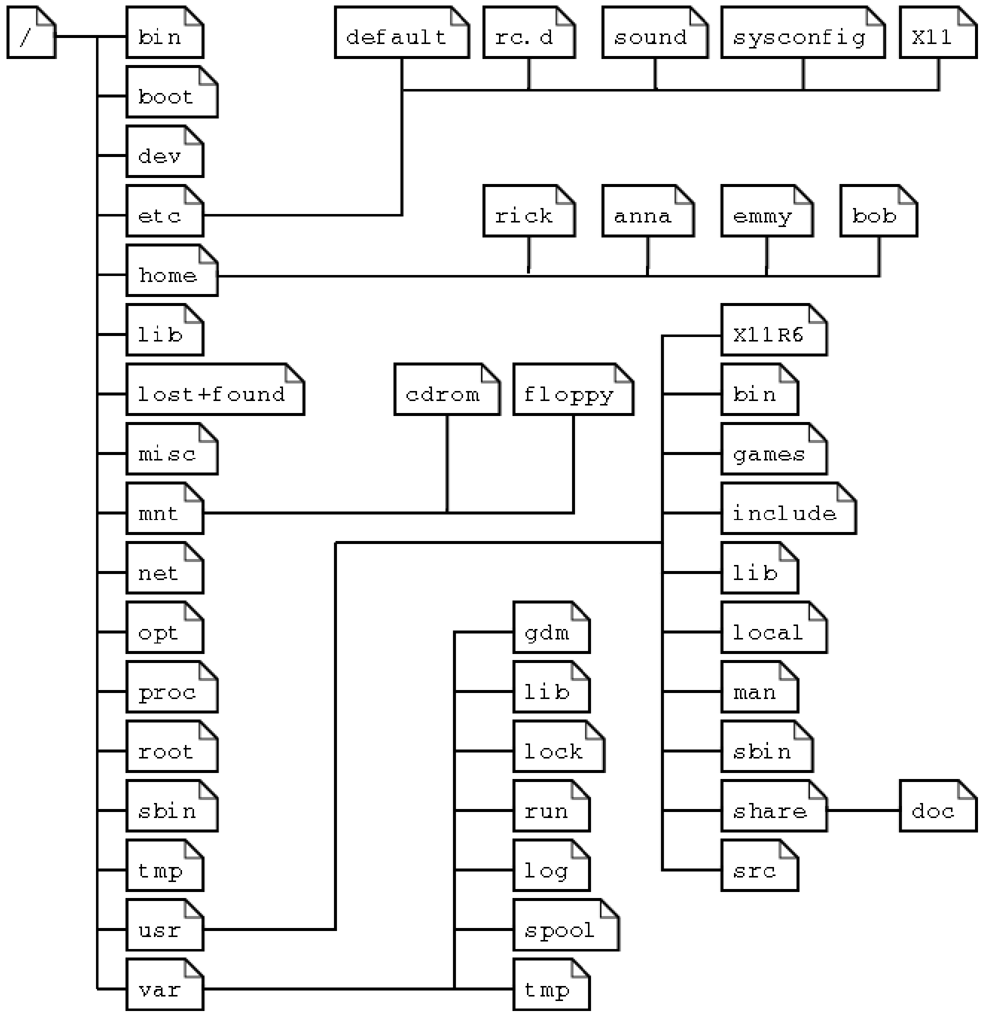
df: a tool to display information about the partitions

The df command only displays information about active non-swap partitions. These can include partitions from other networked systems

```
freddy: > df -h
  Filesystem     Size   Used   Avail   Use%   Mounted on
  /dev/hda8      496M   183M   288M     39%   /
  /dev/hda1      124M   8.4M   109M      8%   /boot
  /dev/hda5       19G    15G   2.7G     85%   /opt
  /dev/hda6      7.0G   5.4G   1.2G     81%   /usr
  /dev/hda7      3.7G   2.7G   867M     77%   /var
```

Partitions are mounted on a mount point, which can be almost any directory in the system.

## Linux file system layout on a Redhat system



Subdirectories of the root directory

- **/bin** Common programs, shared by the system, the system administrator and the users.
- **/boot** Kernel image
- **/etc** Most important system configuration files are in /etc, this directory contains data similar to those in the Control Panel in Windows
- **/home** Home directories of the common users.
- **/lib** Library files, includes files for all kinds of programs needed by the system and the users.
- **/lost+found** Every partition has a lost+found in its upper directory. Files that were saved during failures are here.
- **/misc** For miscellaneous purposes.
- **/mnt** Standard mount point for external file systems, e.g. a

CD-ROM or a digital camera.

- **/net** Standard mount point for entire remote file systems
- **/opt** Typically contains extra and third party software.
- **/proc** A virtual file system containing information about system resources. More information about the meaning of the files in proc is obtained by entering the command man proc in a terminal window.
- **/root** The administrative user's home directory. Mind the difference between /, the root directory and /root, the home directory of the root user.
- **/sbin** Programs for use by the system and the system administrator.
- **/tmp** Temporary space for use by the system, cleaned upon reboot, so don't use this for saving any work!

- **/usr** Programs, libraries, documentation etc. for all user-related programs.
- **/var** Storage for all variable files and temporary files created by users, such as log files, the mail queue, the print spooler area, space for temporary storage of files downloaded from the Internet, or to keep an image of a CD before burning it.

## inode

A file is represented by an inode: a kind of serial number containing information about the actual data that makes up the file. A inode contains the following information:

- Owner and group owner of the file.
- File type (regular, directory, ...)
- Permissions on the file
- Date and time of creation, last read and change.
- Date and time this information has been changed in the inode.
- Number of links to this file (see later in this chapter).
- File size
- An address defining the actual location of the file data.

## More about inode

- Every partition has its own set of inodes.
- Throughout a system with multiple partitions, files with the same inode number can exist.
- There is a maximum of the number of inodes that a partition can hold.
- Use `ls -i` to display inode numbers.

## PATH variable

This variable lists those directories in the system where executable files can be found, and thus saves the user a lot of typing and memorizing locations of commands.

To list the content of PATH variable, try `echo $PATH`. Typical output will be like:

```
rogier:> echo $PATH /usr/lo-
cal/bin:/usr/local/sbin:/sbin:/usr/sbin:/bin:/usr/bin:/usr/bin/X11:/usr/games
```

In this example, the directories /usr/local/bin, /usr/local/sbin, /sbin, /usr/sbin, /bin, /usr/bin, /usr/bin/X11, /usr/games are subsequently searched for the required program. As soon as a match is found, the search is stopped

## Absolute and relative paths

- Absolute path: The path starts with a slash and is called an absolute path. For example

  /home/wch/Document/cs202/slides_linux_filesystem.pdf

- Relative path: The paths that don't start with a slash are always relative. For example if the current directory is /home/wch/Document/cs202/, then

  ./slides_linux_filesystem.pdf refers to slides_linux_filesystem.pdf in the current directory.

  ../slides_linux_filesystem.pdf refers to slides_linux_filesystem.pdf in the parent directory.

## Access permission of files

- Each file has three types of users: Owner, Group users, and other users
- For each type of users, there are three rights: Read, Write, and Execution, represented by three bits.
- Totally, 9 bits are used to define the access permission. The first three bits for owner, the second three bits for group users, the third three bits for other users.
- Try `ls -l` , the output may be like:

  ```
  rw-r–r– 1 wch users 50688 2005-08-07 16:36 temp.txt
  ```

  It shows that the owner can read and write the file, and that the group users and other users can only read the file.

## ls: View directory contents and file properties

This command can given information about: file name, file type, file permissions, file size, inode number, creation date and time, owners and amount of links to the file. It has the following options:

- **-a**: show all the files including the hidden files, which have a name starting with a dot.
- **-l**: shows a long list of files and their properties as well as the destinations that any symbolic links point to
- **-t**: sort by modification time
- **-s**: print size of each file, in blocks
- **-r**: reverse order while sorting

The options can be combined, for example ls -latr displays the same files, only now in reversed order of the last change, so that the file changed most recently occurs at the bottom of the list.

## Color-ls default color scheme

On most Linux versions ls is aliased to color-ls by default. To achieve this, every file type has its own color. The standard scheme is in /etc/DIR_COLORS:

| Color | File type |
|---|---|
| blue | directories |
| red | compressed archives |
| white | text files |
| pink | images |
| cyan | links |
| yellow | devices |
| green | executables |
| flashing red | broken links |

If ls is not aliased to color-ls, use option –color[=WHEN] to control whether color is used to distinguish file types. WHEN may be never, always, or auto

## Mkdir: Creating directories

Creating directories and subdirectories. For example:

```
richard:~/archive > mkdir 1999 2000 2001
richard:~/archive > ls
1999/ 2000/ 2001/
```

-p: creating directories and subdirectories in one step. For example:

```
richard:~/archive > mkdir 2001/reports/Restaurants-Michelin/
mkdir: cannot create directory ‘2001/reports/Restaurants-Michelin/’:
No such file or directory
richard:~/archive > mkdir -p 2001/reports/Restaurants-Michelin/
richard:~/archive > ls 2001/reports/
Restaurants-Michelin/
```

## mv: Moving files

moving a file from one directory to another or change its name.

format: mv file1 file2

```
richard:~/archive > mv ../report[1-4].doc reports/Restaurants-Michelin/
```

This command is also applicable when renaming files:

```
richard:~> ls To_Do
-rw-rw-r-- 1 richard richard 2534 Jan 15 12:39 To_Do
richard:~> mv To_Do done
richard:~> ls -l done
-rw-rw-r-- 1 richard richard 2534 Jan 15 12:39 done
```

## Copying and deleting files

**cp**, format `cp fromfile tofile`

- **-r**: recursive copy

**rm**, format `rm filename`

- **-r**: remove the contents of directories recursively
- **-f**: ignore nonexistent files, never prompt
- **-i**: prompt before any removal

`rm -rf temp/` will remove the whole `temp/` directory without any prompt. Be careful!

## Searching for files

**Which**: looking in the directories listed in the user's search path for the required file, suitable for the executable binary files, for example

```
tina:∼> which acroread
/usr/bin/acroread
```

shows the binary file of acroread is in directory /usr/bin/

Using the which command also checks to see if a command is an alias for another command:

```
gerrit:∼> which -a ls
ls is aliased to 'ls -F –color=auto'
ls is /bin/ls
```

## Searching for files

**Find**: searching other paths beside those listed in the search path. The most common use is for finding file names:

```
find path -name searchstring
```

This command not only allows you to search file names, it can also accept file size, date of last change and other file properties as criteria for a search. To search files of a certain size, use -size option:

```
peter:~> find . -size +5000k
psychotic_chaos.mp3
```

## Searching for files

**locate**: Search for a file based on a file index database that is updated only once every day. It uses less resources than find and therefore shows the results nearly instantly.

**slocate**: same as locate except that it prevents users from getting output they have no right to read.

On most systems, **locate** is a symbolic link to **slocate**

```
wch@ubuntu:~$ ls /usr/bin/locate -l
lrwxrwxrwx 1 root root 7 2005-07-31 12:14 /usr/bin/locate -> slocate
```

## Browsing for files

**cat**: Output was streamed in an uncontrollable way.

format: cat file

**less**, **more**: Browse text files under control

format: less file

**pg**: Browse text files one page at a time

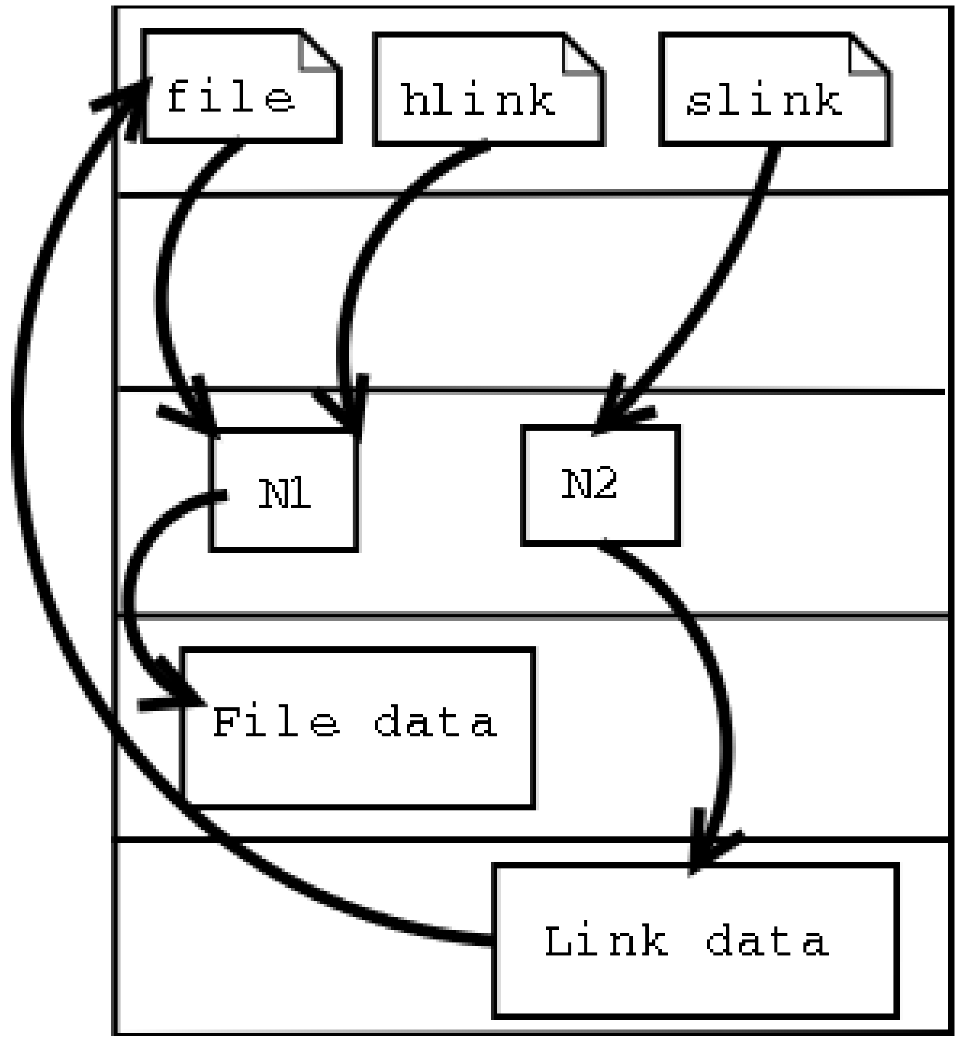
format: pg file

## Creating link between files

A link is nothing more than a way of matching two or more file names to the same set of file data. There are two ways to achieve this:

- **Hard link**: Associate two or more file names with the same inode.
  - Hard links share the same data blocks on the hard disk, while they continue to behave as independent files.
  - Hard links can't span partitions, because inode numbers are only unique within a given partition.
- **Soft link**: a small file that is a pointer to another file.
  - A soft link contains the path to the target file instead of a physical location on the hard disk.
  - Soft links can span across partitions.

## Hard and soft link mechanism



## More about hard link and soft link

Each regular file is in principle a hardlink.

To create hardlink, use `ln`, format `ln targetfile linkname`

To create soft link. use `ln -s` , format: `ln -s targetfile linkname`

## Access permission of files

Access mode codes

| Code | Meaning |
|---|---|
| 0 or - | the access right is not granted. |
| 4 or r | read access is granted |
| 2 or w | write permission is granted |
| 1 or x | execute permission is granted |

User group codes

| Code | Meaning |
|---|---|
| u | user permissions |
| g | group permissions |
| o | permissions for others |

## Change access permission

**chmod**: Change access permission of files

- **o**: change the access permisson of users in other group
- **g**: change the group acess permission
- **u**: chane the owner access permission
- **+**: add the acess permission
- **-**: remove the access permission

For example: `chmod u+x config.txt` adds the execute permission to the owner `chmod u-w config.txt` removes the write permission of the owner

## Numeric arguments for chmod

When using **chmod** with numeric arguments, the values for each granted access right have to be counted together per group. Thus we get a 3-digit number.

The most common combinations:

| Command | Meaning |
|---|---|
| chmod 400 file | To protect a file against accidental overwriting. |
| chmod 500 directory | To protect yourself from accidentally removing, renaming or moving files from this directory. |
| chmod 600 file | A private file only changeable by the user who entered this command. |
| chmod 644 file | A publicly readable file that can only be changed by the issuing user. |
| chmod 660 file | Users belonging to your group can change this file, others don't have any access to it at all. |
| chmod 700 file | Protects a file against any access from other users, while the issuing user still has full access. |
| chmod 755 directory | For files that should be readable and executable by others, but only changeable by the issuing user. |
| chmod 775 file | Standard file sharing mode for a group. |
| chmod 777 file | Everybody can do everything to this file. |

## Sample development — more

The logic of more

- Step 1: show 24 lines from input
- Step 2: print [more?] message, input Enter, SPACE, or q
- Step 3: if Enter, advance one line
- Step 4: if SPACE, go back to Step 1.
- Step 5: if q, exit

## A draft version of more

```
/* more01.c  - version 0.1 of more
 *     read and print 24 lines then pause for a few special commands */
#include        <stdio.h>
#define PAGELEN 24
#define LINELEN 512
void do_more(FILE *);
int  see_more();
int main( int ac , char *av[] )
{
        FILE *fp;
        if ( ac == 1 ) do_more( stdin );
        else {
                while ( --ac ) {
                        if ( (fp = fopen( *++av , "r" )) != NULL ){
                                do_more( fp ) ;
                                fclose( fp );
                        }
                        else
                                exit(1);
                }
        }
        return 0;
}
```

## A draft version of more

```
void do_more( FILE *fp )
/*
 *  read PAGELEN lines, then call see_more() for further instructions
 */
{
        char    line[LINELEN];
        int     num_of_lines = 0;
        int     see_more(), reply;

        while ( fgets( line, LINELEN, fp ) ){           /* more input */
                if ( num_of_lines == PAGELEN ) {        /* full screen? */
                        reply = see_more();             /* y: ask user  */
                        if ( reply == 0 )               /*    n: done   */
                                break;
                        num_of_lines -= reply;          /* reset count */
                }
                if ( fputs( line, stdout )  == EOF )    /* show line */
                        exit(1);                        /* or die */
                num_of_lines++;                         /* count it */
        }
}
```

```
int see_more()
/*
 *     print message, wait for response, return # of lines to advance
 *     q means no, space means yes, CR means one line
 */
{
        int     c;

        printf("\033[7m more? \033[m");         /* reverse on a vt100 */
        while( (c=getchar()) != EOF )           /* get response */
        {
                if ( c == 'q' )                 /* q -> N               */
                        return 0;
                if ( c == ' ' )                 /* ' ' => next page     */
                        return PAGELEN;         /* how many to show     */
                if ( c == '\n' )                /* Enter key => 1 line  */
                        return 1;
        }
        return 0;
}
```

## The second version of more

The first version of the more get input from the standard input, which may become a problem.

```
%ls /bin/more01
```

more01 does not stop after 25 lines.

To solve, this problem, we can make more read from the keyboard.

There is a special file /dev/tty, which is actually a connection to the keyboard and screen. Therefore, more can communicate with the terminal via this file.

## The second version of more

```
/*  more02.c  - version 0.2 of more
 *      read and print 24 lines then pause for a few special commands
 *      feature of version 0.2: reads from /dev/tty for commands
 */
#include  <stdio.h>
#define PAGELEN 24
#define LINELEN 512
void do_more(FILE *);
int see_more(FILE *);
int main( int ac , char *av[] )
{
        FILE *fp;
        if ( ac == 1 )  do_more( stdin );
        else
                while ( --ac )
                        if ( (fp = fopen( *++av , "r" )) != NULL ) {
                                do_more( fp ) ;
                                fclose( fp );
                        }
                        else
                                exit(1);
        return 0;
}
```

## The second version of more

```
void do_more( FILE *fp )
/* read PAGELEN lines, then call see_more() for further instructions */
 {
        char    line[LINELEN];
        int     num_of_lines = 0;
        int     see_more(FILE *), reply;
        FILE    *fp_tty;
        fp_tty = fopen( "/dev/tty", "r" );        /* NEW: cmd stream   */
        if ( fp_tty == NULL )                     /* if open fails     */
                exit(1);                          /* no use in running */
        while ( fgets( line, LINELEN, fp ) ){     /* more input */
                if ( num_of_lines == PAGELEN ) {       /* full screen? */
                        reply = see_more(fp_tty);  /* NEW: pass FILE *  */
                        if ( reply == 0 )              /*    n: done   */
                                break;
                        num_of_lines -= reply;         /* reset count */
                }
                if ( fputs( line, stdout )  == EOF )   /* show line */
                        exit(1);                       /* or die */
                num_of_lines++;                        /* count it */
        }
}
```

## The second version of more

```
int see_more(FILE *cmd)       /* NEW: accepts arg  */
/*
 * print message, wait for response, return # of lines to advance
 * q means no, space means yes, CR means one line
 */
{
        int c;

        printf("\033[7m more? \033[m");          /* reverse on a vt100 */
        while( (c=getc(cmd)) != EOF )           /* NEW: reads from tty  */
        {
                if ( c == 'q' )                 /* q -> N               */
                        return 0;
                if ( c == ' ' )                 /* ' ' => next page */
                        return PAGELEN;         /* how many to show */
                if ( c == '\n' )                /* Enter key => 1 line */
                        return 1;
        }
        return 0;
}
```

## Problems to be solved in the second version

- How to respond immediately to a keystroke?
- How to know the percentage of a file shown?
- How to make it run on all kinds of terminals?