**Chapter 4 Notes**

**4.1 ls Revisisted (-l): Listing File Attributes**

* File attributes are stored in the inode, a structure that is maintained in a separate area of the hard disk.
* ls –l command
  + Lists the listing of all the file attributes, in ASCII collating sequence
  + Seven fields with 9 records
    - All of these attributes are stored in the inode which is pointed to by the inode number
    - Types and permissions
      * first field shows file type
        + –a ordinary file
        + –d directory
        + –l symbolic link
      * Remaining 9 characters show the permission type which take r,w,x,-
    - Links
      * Indicates number of links with the file
      * Directories have a count of at least 2
      * Even though a 2 files may have the same file size that doesn’t confirm that there’s a single file out there
        + They may have different file sizes since one may contain the pointer to the real file
    - Ownership and group ownership
      * Shows the owner of the files as the third field, the fourth field shows the group owner of the files
      * The owner may tamper with the file is every possible way
    - Size
      * Fifth field shows the size in bytes
        + Represents the character count and not the disk space consumption of the file

Kernel allocates space in 1024 bytes or more

Since it is in memory it is of 2^10

So even though a file may be 163 bytes it may occupy 1024 bytes on the system.

* + - Last modification time
      * Displayed to the closest second, the year is displayed if it has been longer than a year (6 months in Linux)
    - Filename
      * Lists by filename
    - At the top it lists the total number of blocks that is taken up by these files on disk
      * Blocks are loaded in from there sector and buffered (this is a chunk of the main memory)
* ls –ld
  + Lists the attributes of a directory rather than the filenames it contains
  + No option to list only directories

**4.2 File Permissions**

* Each group represents a category when looking at the permissions
  + owner’s, group’s, other’s
  + r
    - indicates read permission, you may display the file
  + w
    - indicates write permission, which means you may edit the file
  + x
    - indicates execute permission, which means the file may be executed as a program
  + –
    - absence of a corresponding permission
* umask
  + determines the default permissions for file creation
  + must be set before the file is created
* chmod
  + The change mode command will allow you change the permissions of a file
* Note that if the owner renounces ownership of the file, the group’s permissions then apply to him

**4.3 chmod: changing File Permissions**

* owner is often referred to as the user
* chmod [-R] mode file
* mode can be represented in 2 ways
  + relative manner by specifying the changes to the current permissions
  + absolute manner by specifying the final permissions
* Relative permissions
  + only changes the permissions specified in mode and leaves the other permissions unchanged
  + mode as used in the syntax has 3 categories
    - user category
    - operation to be performed
    - the type of permissions
    - chmod (command) u(category) +(operation) x(permission) note(file)
      * chmod u+x note
    - categories
      * u – user
      * g – group
      * o – others
      * a – all
    - operation
      * + - assigns permission
      * - - removes permission
      * = assigns absolute permission
    - permission
      * r – read
      * w – write
      * x – execute
    - you can write chmod –ugo+x date.sh and it is the same as chmod –a+x date.sh
    - you can use 2 expression in one chmod command using a comma as the delimiter
      * chmod u-x,go+r date.sh
      * some versions don’t accept the comma
* Absolute permissions
  + Assigns only the specified permissions and removes the others limitedly
    - chmod =r date.sh
      * will make only read available to all, the same as chmod a=r date.sh or chmod ugo=r date.sh
  + has limitations as only octal numbers do the absolute assignment
    - You understand octal numbers but what is the point of having the (=) operator then. Is it just relative permission assignment
  + A file with 000 can be deleted and a file with 777 can have it’s deletion prevented
    - It’s the file permissions of the directory that permit/deny deletion of files
* Recursive Operation (-R)
  + descends a directory hierarchy and applies the expression to every file and subdirectory it finds in the tree walk

**4.4 The Directory**

* Stores the filename and the inode number
  + Size of a directory is the number of files housed by it and not the size of the files
* umask
  + works in reverse of chmod, it will prevent the octal number from occurring
  + each process has it’s own mask which is applied whenever a file is created
* Read Permission
  + means that the list of the filenames stored in that directory
  + if the read permission is removed ls won’t work
    - This doesn’t prevent you from reading the files separately if you know their names
* Write Permission
  + implies that you are able to create or removes files contained within that directory
    - that would make the kernel modify the entries
    - Security issues are normally related to a directories write permission