Unix Files and Attributes

Unix File Attributes

- We can use **ls** to obtain a long listing of a file. For example:

```
sepc92:> ls -l heart.final
-rw-r--r--.  1  glass  cs  213  Jan 31 00:12  heart.final
```

Field #	Field value	File Attribute Meaning
1	-rw-r--r--	the type and permission mode of the file, which indicates who can read, write, and execute the file
2	1	the hard link count
3	glass	the username of the owner of the file
4	cs	the group name of the file
5	213	the size of the file, in bytes
6	Jan 31 00:12	the time that the file was last modified
7	heart.final	the name of the file

Unix File Attributes - Filenames

- The name of the file is shown in field 7.
- A UNIX filename may be up to *255* characters in length. You may use any printable characters you want in a filename except the slash (/)
  - You avoid the use of any character that is special to a shell (like <,>, *, ?, or the tab), as these can confuse both the user and the shell.
- There is no requirement that a file end in an extension such as ".c" and ".h," although many UNIX utilities (e.g., the C compiler) will accept only files that end with a particular suffix.
  - Thus, the filenames "heart" and "heart.final" are valid
- The only filenames that you definitely *can't* choose are (..) and (.) as these are predefined filenames that correspond to your current working directory and its parent directory, respectively.

Unix File Attributes – File Modification Time

- Field 6 shows the time that the file was last modified and is used by several utilities.
- The **find** utility can be used to search for file names matching with a certain pattern.

  sepc92:> *find –name "*.c"*

  The above command is to find from the current directory and subdirectories those file names that has .c as the file name extension.
- The **find** utility supports an option based on the last modification time.

  sepc92:> *find –name "*.c" -mtime -1*

  The above command adds a condition that the modification time is less than 1 day ago.

Unix File Attributes – File Owner

- Field 3 tells you the owner of the file.
- Every UNIX process has an owner, which is typically the same as the username of the person who started it.
  - For example, my login shell is owned by “glass,” which is my username. Whenever a process creates a file, the file’s owner is set to the process’ owner.
  - This means that every file that I create from my shell is owned by “glass,” the owner of the shell itself.

Unix File Attributes – File Owner

- Note that while the text string known as the username is typically how we refer to a user, internally UNIX represents this as an integer known as the *user* ***ID.***
- The username is easier for humans to understand than a numeric ID.
- We will refer to the textual name as *username* and use *user* ***ID*** to refer to the numeric value itself.

Unix File Attributes – File Group

- Field 4 shows the file’s group.
- Every UNIX user is a member of a group.
- This membership is initially assigned by the system administrator and is used as part of the UNIX security mechanism.
- For example, my group name is “cs” Every UNIX process also belongs to a specific group, usually the same as that of the user which started the process. My login shell belongs to the group name “cs”.
  - Because each file created by a process is assigned to the same group as that of the process that created the file, every file that I create from my shell has the group name “cs.”

Unix File Attributes – File Group

- As with the user ID, the group is usually referenced by the text string name, but is represented internally as an integer value called the *group* ID.
- We will refer to the textual name as *group name* and use *group **ID*** to refer to the numeric value itself.

Unix File Attributes – File Type

- Field 1 describes the file’s type and permission settings. For convenience, here’s the output from the previous *ls* example:

  -rw-r--r--. 1 glass cs 213 Jan 31 00:12 heart.final

- The first character of Field 1 indicates the type of the file, which is encoded as shown below. In the example, the type of “heart.final” is indicated as a regular file.

Character	File type
-	regular file
d	directory file
b	buffered special file (such as a disk drive)
c	unbuffered special file (such as a terminal)
l	symbolic link
p	pipe
s	socket

Unix File Attributes – File Type

- A file’s type can often be determined by using the file utility. For example, when we ran **file** on “heart.final”, we can see this:

```
sepc92:> file heart.final            ... determine the file type
 heart.final: ascii text
sepc92:> _
```

Utility: **file** { *fileName* }+
The **file** utility attempts to describe the contents of the *fileName* arguments, including the language that any text is written in. When using **file** on a symbolic link file, **file** reports on the file that the link is pointing to, rather than the link itself.

Unix File Attributes – File Permissions

- The next nine characters of Field 1 indicate the file’s permission settings.
- In the current example, the permission settings are “rw-r--r--”:

```
-rw-r--r--.  1   glass   cs   213   Jan 31 00:12   heart.final
```

Unix File Attributes – File Permissions (con't)

- These nine characters should be thought of as being arranged in three groups of three characters, as shown below, where each cluster of three letters has the same format.

Read permission	Write permission	Execute permission
r	w	x

- If a dash occurs instead of a letter, then permission is denied.

User (owner)	Group	Others
rw-	r--	r--

(Note: While file permission is quite useful, it is not 100% accurate and can be fooled by some file formats.)

Unix File Attributes – File Permissions (con’t)

- The meaning of the read, write, and execute permissions depends on the type of the file, as shown below.

	Regular file	Directory file	Special file
Read	The process may read the contents.	The process may read the directory (i.e., list the names of the files that it contains).	The process may read from the file using the read() system call.
Write	The process may change the contents.	The process may add files to or remove files from the directory.	The process may write to the file, using the write() system call.
Execute	The process may execute the file (which makes sense only if the file is a program)	The process may access files in the directory or any of its subdirectories.	No meaning.

Changing A File’s Permissions : chmod

- The **chmod** utility is used to change the permission of files. For example, to remove read permission from others:

```
sepc92:> ls –l  heart.final               ...before
-rw-r--r--.   1 glass  music 213 Jan 31 00:12 heart.final
sepc92:> chmod o-r heart.final      ...remove read for others
sepc92:> ls –l  heart.final               ...after
-rw-r-----.   1  glass music 213 Jan 31 00:12 heart.final
sepc92:> _
```

Changing A File's Permissions : chmod (con't)

- The table below shows some other examples of the use of **chmod**.

Requirement	Change parameters
Add group write permission.	g+w
Remove user read and write permission.	u-rw
Add execute permission for user, group, and others.	a+x
Give others read permission.	o+r
Add write permission for user, and remove read permission from group.	u+w, g-r

Changing A File's Permissions : chmod (con't)

- Another example of chmod:

```
sepc92:> cd                              ...change to home directory
sepc92:> ls –ld .                        ...list attributes of home dir
drwxr-xr-x.   45   glass  music  4096   Apr 29 14:35   .
sepc92:> chmod o-rx .                     ...update permissions
sepc92:> ls –ld .                         ...confirm
drwxr-x---.   45   glass   music  4096   Apr 29 14:35   .
sepc92:> _
```

- Note that the -d option of **ls** is used to ensure that the attributes of the directory, rather than the attributes of its files, were displayed.

Changing A File's Permissions : chmod (con't)

- The **chmod** utility allows you to specify the new permission setting of a file as an octal number. Each octal digit represents a permission triplet. For example, if you wanted a file to have the permission settings

  rwxr-x---

  then the octal permission setting would be 750, calculated as shown below.

	User	Group	Others
setting	rwx	r-x	---
binary	111	101	000
octal	7	5	0

Changing A File’s Permissions : chmod (con’t)

- The octal permission setting would be supplied to **chmod** as follows:

```
sepc92:> chmod  750  .                ... update permissions
sepc92:> ls –ld .                     ... confirm
drwxr-x---.   45   glass  music  4096   Apr 29 14:35    .
sepc92:> _
```