Lab 02 - Managing Files & Permissions

1. *Identify and note the line in `/etc/mtab` that contains the mount-information associated with the disk device.*

```
[root@UWS ~]# cat /etc/mtab | grep floppy
/dev/loop0 /media/floppy ext2 rw,relatime,errors=continue 0 0
```

2. *Note down the outputs of the commands `ls -l /dev/loop0` and `ls -l /media/floppy`. Compare the outputs and comment on what is actually achieved by the mount command that you have just performed.*

```
[root@UWS ~]# ls -l /dev/loop0
brw-------    1 root     root        7,   0 Sep 20 18:53 /dev/loop0
[root@UWS ~]# ls -l /media/floppy/
total 13
-rw-r--r--    1 root     root           61 Sep 13  2020 file.txt
drwx------    2 root     root        12288 Sep 13  2020 lost+found
```

The floppy disk image has been attached to `/dev/loop0` which, in turn, is mounted onto `/media/floppy`. The use of a loop device allows for loop mounting of images onto the overall filesystem as though they were physical devices.

The difference between the outputs stems from the fact that the disk image is mounted *via* the loop device onto `/media/floppy`, the logical point at which the contents of the image are mounted onto the overall filesystem.

3. *After mounting the virtual floppy, has `/etc/fstab` changed as well, or are all entries still the same?*

No changes. `fstab` defines filesystems to be mounted at boot, and *how* they're integrated as part of the overall filesystem.

4. *Why is it important not to have target media mounted while you try to create a file system?*

If mounted, the kernel may try to perform read or write operations on the target media. Unmounting the media ensures there is no risk of the operating system accessing the device and interfering with the creation of a new filesystem.

5. *How many `inodes` and `blocks` are created on the system?*

Assuming "the system" refers to the system as a whole:

```
[root@UWS ~]# df / -P
Filesystem           1024-blocks    Used Available Capacity Mounted on
/dev/root              1048576    138868    909708  13% /
[root@UWS ~]# find / -type f -print | wc -l
16203
```

- 1,048,576 blocks (~1.0G using `df / -h`)
- The number of inodes is contingent on the size of the filesystem, with a default configuration of one inode per 2048 bytes. Assuming this to be the case there are 524,288 inodes in `/dev/root`.

6. *How many blocks are reserved for the super user (root)?*

5% of the total capacity is reserved by the root user to prevent the system from failing if or when it runs out of available space. In this instance, 5% of 1,048,576 blocks is ~52,429 blocks.

7. *Manually run the program `fsck` on the virtual floppy. What is displayed by `fsck`?*

Given that the filesystem is mounted at present:

```
[root@UWS ~]# fsck /dev/loop0
fsck from util-linux 2.32.1
e2fsck 1.44.2 (14-May-2018)
/dev/loop0 is mounted.
e2fsck: Cannot continue, aborting.
```

Unmounting the filesystem results in the following output:

```
[root@UWS ~]# umount /dev/loop0
[root@UWS ~]# fsck /dev/loop0
fsck from util-linux 2.32.1
e2fsck 1.44.2 (14-May-2018)
/dev/loop0: clean, 12/184 files, 48/1440 blocks
```

8. *Note any files and directories that are on the floppy at this time.*

```
[root@UWS floppy]# ls -la /media/floppy
total 19
drwxr-xr-x    3 root     root          1024 Sep 21 10:37 .
drwxr-xr-x    3 root     root            60 Sep  9  2020 ..
-rw-r--r--    1 root     root            61 Sep 13  2020 file.txt
drwx------    2 root     root         12288 Sep 13  2020 lost+found
-rw-r--r--    1 root     root            11 Sep 21 10:37 tst.dat
```

9. *Can you think of any apocalyptic scenario that might produce an entry in the `lost+found` directory?*

An improper shutdown might result in orphaned inodes, or if a block device experiences some kind of hardware failure.

10. *Why would the kernel not allow you to dismount a filesystem that you are currently using?*

Pending IO operations should complete before a filesystem is unmounted. Linux, for example, buffers writing to disk in RAM until such a time as the kernel deems appropriate.

11. *Use the command cd /root, before issuing the command line umount /media/floppy again. Does the un-mounting operation work now?*

Yes!

```
[root@UWS ~]# cd /root
[root@UWS ~]# pwd
/root
[root@UWS ~]# umount /media/floppy
[root@UWS ~]# mount
/dev/root on / type 9p (rw,sync,dirsync,relatime,trans=virtio)
devtmpfs on /dev type devtmpfs (rw,relatime,size=255020k,nr_inodes=63755,mode=75
5)
```

```
proc on /proc type proc (rw,relatime)
devpts on /dev/pts type devpts (rw,relatime,gid=5,mode=620,ptmxmode=666)
tmpfs on /dev/shm type tmpfs (rw,relatime,mode=777)
tmpfs on /tmp type tmpfs (rw,relatime)
tmpfs on /run type tmpfs (rw,nosuid,nodev,relatime,mode=755)
sysfs on /sys type sysfs (rw,relatime)
nfsd on /proc/fs/nfsd type nfsd (rw,relatime)
```

12. *Why can you still set on the floppy, even though you have just ejected the floppy disk?*

The mount point we used to access the image in the overall filesystem is a directory we created. Devices, virtual or otherwise, are usually mounted onto empty directories by convention.

13. *Explain why you no longer see the previous files `lost+found` and `tst.dat` on the mount-point?*

Because we've unmounted the image and its contained filesystem from the overall filesystem.

14. *Explain the difference between the two command lines given. What does the re-direction operator `>` do?*

```
[root@UWS ~]# echo '1234567890' test.dat
1234567890 test.dat
[root@UWS ~]# echo '1234567890' > test.dat
```

- The first command is doing little more than supplying standard input arguments to the `echo` programme, and `echo` is doing what it does: re-direct standard input to standard output.
- The second command uses the `>` redirection character which redirects the standard input on the left to a designated file on the right. The file is created if it doesn't already exists, and if it does, the content of the file is overwritten.

15. *(After performing the `mv` command) What happened to the original `test.dat` file?*

```
[root@UWS ~]# mv test.dat test_01.dat
[root@UWS ~]# ls -l
total 1448
drwxr-xr-x    2 root     root            37 Sep 21 13:01 Desktop
-rw-r--r--    1 root     root            11 Sep 21 13:13 test_01.dat
-rw-------    1 student  root       1474560 Sep 21 13:07 virtual_floppy.img
```

When using `mv` without specifying an alternate directory `mv` acts as though the file has been renamed. The inode of the file has not changed, instead the name of the directory entry associated with that inode has been modified.

16. *Examine the file `test_01.dat` with the `cat` command. What does the operator `>>` do?*

```
[root@UWS ~]# echo 'abcdefghi' >> test_01.dat
[root@UWS ~]# cat test_01.dat
1234567890
abcdefghi
```

The `>>` operator appends standard output of `echo` to the designated file.

17. *Based on the commands shown above: Explain the effect of the `-i` qualifier in the `cp` command.*

```
[root@UWS ~]# cp -i test_01.dat test_b.dat
[root@UWS ~]# cp -i test_01.dat test_b.dat
cp: overwrite 'test_b.dat'?
```

`cp`'s `-i` flag prompts the user if they wish to go ahead with the change if an overwrite will occur. The `-i` stands for 'interactive'.

18. *(After performing `cmp test_a.dat test_b/dat`) What is the output of the above command line, or is there no output at all?*

There is no output, indicating that there are no differences between the two files specified.

19. *What is the output of the above `cmp` command now and why?*

```
[root@UWS ~]# echo 'ei caramba' >> test_a.dat
[root@UWS ~]# cmp test_a.dat test_b.dat
cmp: EOF on test_b.dat
```

`cmp` reached the end of `test_b.dat` before reaching the end of file on `test_a.dat`.

```
student@UWS ~]$ od -c test_b.dat
0000000   1   2   3   4   5   6   7   8   9   0  \n   a   b   c   d   e
0000020   f   g   h   i  \n
0000025
[student@UWS ~]$ od -c test_a.dat
0000000   1   2   3   4   5   6   7   8   9   0  \n   a   b   c   d   e
0000020   f   g   h   i  \n   e   i       c   a   r   a   m   b   a  \n
0000040
```

20. *What is the output of the above command? Do you know another Unix command that does roughly the same?*

```
[student@UWS ~]$ cat test_a.dat
1234567890
abcdefghi
ei caramba
```

In cases like these with only a single argument `cat` will send the contents of a given file to standard out.put `less` will page through a file's content, `more` will print a file's content to standard output with a single argument, and `echo` can also be used in novel ways to print to standard output.

```
[student@UWS ~]$ less test_a.dat ; more test_a.dat ; echo "$(less test_a.dat)"
1234567890
abcdefghi
ei caramba
1234567890
abcdefghi
ei caramba
1234567890
abcdefghi
ei caramba
```

21. *What will cat do with the 3 files? Can you achieve the same result with the more command?*

```
[student@UWS ~]$ cat test_a.dat test_b.dat test_c.dat > test_abc.dat
[student@UWS ~]$ cat test_a.dat test_b.dat test_c.dat
1234567890
abcdefghi
ei caramba
1234567890
abcdefghi
Doh!
```

`cat` will concatenate the content of any given files and print their combined content to standard out. When the `>` operator is used along with a designated file name, the standard output of cat will be redirected to become the content of the file.

`more` will print to standard output the content of any files it's given, and on certain systems, will page through the results, and will allow the user to search using `/` while paging.

```
[student@UWS ~]$ more test_a.dat test_b.dat test_c.dat > more_test
[student@UWS ~]$ cat more_test
1234567890
abcdefghi
ei caramba
1234567890
abcdefghi
Doh!
```

22. *What does the `./` represent in the above command line and is it redundant here?*

`./` represents the current directory. The same result could have been achieved with the following command:

```
[student@UWS ~]$ cp test_abc.dat temp10/
```

23. *Try `cp test_abc.dat ./temp10/*` . What will be the result and does it differ from the previous `cp` command-line `cp test_abc.dat ./temp10/.` ?*

```
[student@UWS ~]$ cp test_abc.dat ./temp10/. && cat temp10/test_abc.dat
1234567890
abcdefghi
ei caramba
1234567890
abcdefghi
Doh!
```

We can see here that the file `test_abc.dat` has been copied to `temp10/` and the file name has been preserved.

```
[student@UWS ~]$ cp test_a.dat ./temp10/* && ls -lah temp10/
total 12
drwxr-sr-x    2 student  student       66 Sep 21 19:23 .
drwxr-sr-x    4 student  student      246 Sep 21 18:46 ..
-rw-r--r--    1 student  student       32 Sep 21 19:53 test_abc.dat
[student@UWS ~]$ cat temp10/test_abc.dat
1234567890
abcdefghi
ei caramba
```

This second command, particularly `cat`, demonstrates that `test_a.dat` has been copied to `temp10/` though the original content has been overwritten by that of `test_a.dat` while the original name has been preserved.

```
[student@UWS ~]$ cp test_a.dat ./temp10/. && ls -lah temp10/
total 16
drwxr-sr-x    2 student  student       93 Sep 21 19:23 .
drwxr-sr-x    4 student  student      246 Sep 21 18:46 ..
-rw-r--r--    1 student  student       32 Sep 21 19:57 test_a.dat
-rw-r--r--    1 student  student       32 Sep 21 19:53 test_abc.dat
```

Performing this third command shows that using the `/.` notation in `cp`'s destination argument preserves the source filename in the destination directory as opposed to the behaviour of the wildcard: overwriting the content of the first file it encounters, or creating a new file if none exists. Using `cp -i` will prevent any files from being overwritten.

24. *Has the `-i` qualifier for `rm` the same meaning as for the `cp` command?*

`rm -i` will prompt before each removal. `cp -i` will also prompt before overwriting a file.

25. *What does the asterisk sign “* represent in the above expression?*

`*` represents a wildcard, which can represent anything in a given expression. In the following example we can use the wildcard as part of an argument for the `find` command to locate each of our `.dat` files:

```
[student@UWS ~]$ ls
Desktop       temp10        test_01.dat   test_a.dat    test_abc.dat  test_b.dat
   test_c.dat
[student@UWS ~]$ find . -name '*.dat'
./test_01.dat
./test_a.dat
./test_b.dat
./test_c.dat
./test_abc.dat
```

26. *Write down the permissions (in words) of `/etc/fstab` and `/etc/shadow` in your logbook, and explain why the latter is so highly restricted.*

```
[student@UWS ~]$ ls -la /etc/fstab
-rw-r--r--    1 root     root           334 Sep  9  2020 /etc/fstab
```

user	group	world
read, write	read	read

```
[student@UWS ~]$ ls -la /etc/shadow
-rw-------    1 root     root          360 Sep 22 09:09 /etc/shadow
```

user	group	world
read, write	n/a	n/a

`shadow` contains encrypted passwords for all users who are defined in the `passwd` file and is readable by none other than `root`, which is why I had to perform `su -` in the following code block before I could `cat` the content of `shadow`.

```
[student@UWS ~]$ cat /etc/shadow
cat: cant open '/etc/shadow': Permission denied
[student@UWS ~]$ su -
[root@UWS ~]# cat /etc/shadow
root:$5$nAjmGoNs$dTVSaUVPM3v/AzxeE7tt ... 99999:7::::::
-- snip --
student:$1$9JBdC6to$lVduSFgip ... 99999:7:::
```

27. *To which category, do you think the root-user belongs (user, group, world, or none of these)?*

Taking a look at the persmissions for a file such as `/etc/shadow`, and given that `root` was able to read its contents in the above example, we can surmise that `root` is a user and their persmissions are defined in the first three permissions columns (in this instance, read and write).

```
[student@UWS ~]$ ls -la /etc/shadow
-rw-------    1 root     root          360 Sep 22 09:09 /etc/shadow
```

28. *Check the entries in /etc/group to find out which users in your system belong to the group 'audio'.*

```
[student@UWS ~]$ grep audio /etc/group
audio:x:29:
```

`audio` has no members. We can surmise this by inspecting the properties of an alternate group:

```
[student@UWS ~]$ grep wheel /etc/group
wheel:x:10:root,student
```

29. *Use the ls –l command to find out the default user, group and world permissions for this newly created file.*

```
[root@UWS ~]# echo 'Donuts' > test_a.dat
[root@UWS ~]# ls -l test_a.dat
-rw-r--r--    1 root     root            7 Sep 22 10:18 test_a.dat
```

user	group	world
read, write	read	read

30. *How did the permission, user- and group-ownership change? Describe in your own words what user, group and world are allowed to do with the file. What is achieved by the commands `chgrp` and `chown`?*

```
[root@UWS ~]# ls -l test_a.dat
-rw-r--r--    1 root     root           7 Sep 22 10:18 test_a.dat
[root@UWS ~]# chmod 777 test_a.dat
[root@UWS ~]# chown student test_a.dat
[root@UWS ~]# chgrp www-data test_a.dat
[root@UWS ~]# ls -l test_a.dat
-rwxrwxrwx    1 student  www-data       7 Sep 22 10:18 test_a.dat
```

- `chmod 777` applied read, write, and execute privileges to the user, the the group, and world.
- The file's user has full read, write, and execute permissions. The same is true for any member of the file's designated group, as well as all other users.
- `chgrp` modified the file's designated group ownership, from `root` to `www-data`.
- `chown` modified the file's designated owner from `root` to `student`.

31. *Is a permission of `7` (in octal notation) a good idea to enable for world-access?*

No. Any user with filesystem access can execute a file with `world` permissions set to 7.

32. *Change the permissions in a way that the user can read and write, but the group and world can only read the file. Note the necessary command in your lab-book.*

```
[root@UWS ~]# chmod 644 test_a.dat && ls -l test_a.dat
-rw-r--r--    1 student  www-data       7 Sep 22 10:18 test_a.dat
```

33. *What is the most permissive access that you can grant in octal representation?*

`777`

34. *What is the default permission given to the new directory /root/temp11?*

```
[root@UWS ~]# ls -la | grep temp11
drwxr-xr-x    2 root     root          37 Sep 22 10:29 temp11
```

user	group	world
read, write, execute	read, execute	read, execute

35. - Which directory permission will allow you to create a files inside of it ? (eg: executing `touch` and `echo` commands)*

We can find out by cycling through the permissions, making a directory, assigning ownership to `root`, and setting full, read, read permissions:

```
[student@UWS Desktop]$ mkdir testdir
[student@UWS Desktop]$ sudo chown root testdir/
[student@UWS Desktop]$ sudo chmod 744 testdir/
```

```
[student@UWS Desktop]$ ls -l
total 4
drwxr--r--    2 root     student         37 Sep 22 10:50 testdir
[student@UWS Desktop]$ touch testdir/file1
touch: testdir/file1: Permission denied
[student@UWS Desktop]$ tree
.
-- testdir
1 directory, 0 files
[student@UWS Desktop]$ sudo touch testdir/file1
[student@UWS Desktop]$ sudo tree
.
-- testdir
    -- file1
1 directory, 1 file
[student@UWS Desktop]$ tree
.
-- testdir
1 directory, 0 files
```

At this point we can see that with `root` as the owner of the directory, `student` cannot view the contents of `testdir`, nor can it create files within the directory.

Adding `group` write permissions to `testdir` maintains `student`'s inability to view or create files in the directory:

```
[student@UWS Desktop]$ sudo chmod 764 testdir/
[student@UWS Desktop]$ ls -l
total 4
drwxrw-r--    2 root     student         59 Sep 22 10:50 testdir
[student@UWS Desktop]$ tree
.
-- testdir
1 directory, 0 files
[student@UWS Desktop]$ touch testdir/studentfile
touch: testdir/studentfile: Permission denied
```

Finally, adding execute permissions to the `group`, `student` is able to finally view and create files in the directory:

```
[student@UWS Desktop]$ sudo chmod 774 testdir/
[student@UWS Desktop]$ touch testdir/studentfile
[student@UWS Desktop]$ tree
.
-- testdir
    |-- file1
    |-- studentfile
1 directory, 2 files
```

36. *Which directory permission is necessary to run a binary executable located inside of it?*

Execute.

37. *Which directory permission is necessary to run a shell-script located inside of it ? (a shell script is an ASCII code!)*

It depends. `source myscript.sh` requires read permissions. In the instance of executing using `./myscript.sh` the kernel will abort execution after parsing a shebang `#!` if the user does not have execute permissions on that script.

38. *Interpret the machine response to this command based on your knowledge about the permissions of the directory. Does the setting of `733` allow `ls` to be performed?*

The permissions granted to the directory means that user `student` cannot read the contents of `temp11`.

```
[student@UWS root]$ pwd
/root
[student@UWS root]$ ls -la | grep temp11
drwx-wx-wx    2 root     root            91 Sep 22 10:29 temp11
[student@UWS root]$ cd temp11/
[student@UWS temp11]$ ls -la
ls: cant open '.': Permission denied
total 0
[student@UWS temp11]$ sudo ls -la
total 16
drwx-wx-wx    2 root     root            91 Sep 22 10:29 .
-- snip --
```

39. *Write down the kernel response for each of the three `more`-command lines given below.*

```
[root@UWS ~]# more /root/temp11/test_a.dat
Donuts
[root@UWS ~]# more /root/temp11/test_b.dat
Donuts
[root@UWS ~]# more /root/temp11/test_c.dat
more: /root/temp11/test_c.dat: No such file or directory
```

40. *Write down the response of the kernel to each of the three `cp`-command given above.*

```
[root@UWS ~]# cp /root/temp11/test_a.dat /home/student
[root@UWS ~]# cp /root/temp11/test_b.dat /home/student
[root@UWS ~]# cp /root/temp11/test_c.dat /home/student
cp: can't stat '/root/temp11/test_c.dat': No such file or directory
```

41. *Explain why you can copy a file, although you can't see it with the ls command.*

```
[student@UWS ~]$ ls -lah /root/ | grep temp11
drwx-wx-wx    2 root     root          91 Sep 22 10:29 temp11
```

Student has execute permissions which allow for the use of any file inside a directory but the user must specifiy an exact file name.

42. *Can you create a file that can be copied by group and world users, residing in a directory with a permission setting of `700`?*

Given the permissions of the directory, no.

```
[root@UWS ~]# touch temp11/q42.file
[root@UWS ~]# chmod 700 temp11/
[root@UWS ~]# ls -lah /root/temp11 | grep q42.file
-rwxrwxrwx    1 root     root           0 Sep 22 13:05 q42.file
[root@UWS ~]# exit
logout
[student@UWS ~]$ cp /root/temp11/q42.file /home/student
cp: cant stat '/root/temp11/q42.file': Permission denied
[student@UWS ~]$ sudo chmod 777 /root/temp11/q42.file
[student@UWS ~]$ cp /root/temp11/q42.file /home/student
cp: cant stat '/root/temp11/q42.file': Permission denied
```

43. *Anonymous ftp-servers only allow users to copy files when they know the file's name. Based on your observations above, which permission would you grant to an anonymous ftp server directory called `download_here` and a file that you create for public download called `download_me`? Which permission would you grant to a file within this directory that should not be downloaded by group or world, called `no_download`? Comment on the advantages of running an anonymous server.*

- `download_here`: 711 allowing full permissions for the owner, execute only for group and world.
- `download_me`: 751 allowing full access for the owner, read and execute for the group, and execute only for world.
- `no_download`: 700 allowing full permissions for the owner and no permissions for group or world.
- Anonymous servers allow for the sharing of files without having to create a user acocunt every time someone new wants to downlload a file. Permissions can be carefully configured to ensure only certain actions can be performed.

44. *Based on your fresh experience: Which of the following permission settings in octal representation are potentially dangerous when assigned to a directory: `700, 033, 633, 644, 755`? Comment on each one individually.*

- `700` creates a directory which is essenaitlly non-existent for anyone except the owner.
- `033` defines a directory with zero permissions for the owner, and write and execute permissions for groups and the public. This allows for the creation, renaming, deletion of files, potentially disastrous, but without read permissions someone would need to know the names of the files they wanted to interact with ahead of time. Unwise for the owner to have zero permissions.
- `633` allows the owner to read and write but not execute. The same security concerns exist as outlined for `033`.
- `644` allows read and write permissions for the owner, and read permissions for all others. Groups and world would not be able to read the contents of the directory.
- `755` allows full permissions for the owner, and read and execute permissions for group and world. Group and world would be able to read the contents of a directory and execute its content.