COMP09024 Unix System Administration

Laboratory 2: File Management

Managing Files and Permissions

Learning Outcomes (Commands)

- **Managing filesystems: mkfs, fsck, mount, umount**
- **Managing files: mv, cp, rm, mkdir, cmp, cat, more**
- **Redirection filters: >, <, >>**
- **Managing permissions: chmod, chown, chgrp**

In this Lab we will work with filesystems and processes within a Unix-like environment. We will introduce the basic commands that enable you to create, check and mount filesystems.

Thereafter we will examine the behaviour of some basic file handling commands and filters. After this we will complete some exercises to show how the permission of files and directories can be set and changed and how they relate to one another.

Managing Filesystems

The most important programs relating to the administration of filesystems within Unix are **`mkfs`** (make filesystem) and **`fsck`** (filesystem check). These programs allow one to create and check the integrity of filesystems, respectively. Depending on the configuration, **`fsck`** can be automatically executed at boot-time. The command **`mkfs`** allows the creation (formatting) of a new filesystem on any physical device.

Two of the most important tasks are the creation of the *inodes* that will contain the information about the files on the system and the creation of back-up reservoirs within the physical device for such crucial information. As one always has to account for things going wrong, the program **`fsck`** is able to correct the occurrence of errors to a limited degree on an existing filesystem. The following exercise will show you that the creation of a file system with **`mkfs`** is actually straightforward and the ability of **`fsck`** to correct for errors is very strong, though in the old days, before journaling filesystems were introduced, **`fsck`** could be very slow.

Type the command:

```
[root@UWS ~]# more /etc/mtab
```

The **`/etc/mtab`** file holds a summary of all currently mounted filesystems. This shoud not be confused with a closely related file called **`/etc/fstab`**. Type the command:

```
[root@UWS ~]# more /etc/fstab
```

The file **`/etc/fstab`** holds the filesystems to be mounted at boot time. In detail **`/etc/fstab`** lists available disks and disk partitions, and indicates how they are to be initialised or otherwise integrated into the overall filesystem. At boot, line-by-line all un-commented **`fstab`** entries are prefixed by the **`mount`** command, which mounts the partitions including some qualifiers that are fed to the system. On a typical Unix-like system the file **`/etc/fstab`** will look like:

(Please remember that comments start with a hash-sign #)

```
[root@UWS ~]# cat /etc/fstab
# <file system> <mount pt>      <type>  <options>       <dump>  <pass>
/dev/root       /               ext2    rw,noauto       0       1
proc            /proc           proc    defaults        0       0
devpts          /dev/pts        devpts  defaults,gid=5,mode=620,ptmxmode=0666   0     0
tmpfs           /dev/shm        tmpfs   mode=0777       0       0
tmpfs           /tmp            tmpfs   mode=1777       0       0
tmpfs           /run            tmpfs   mode=0755,nosuid,nodev  0       0
sysfs           /sys            sysfs   defaults        0       0
```

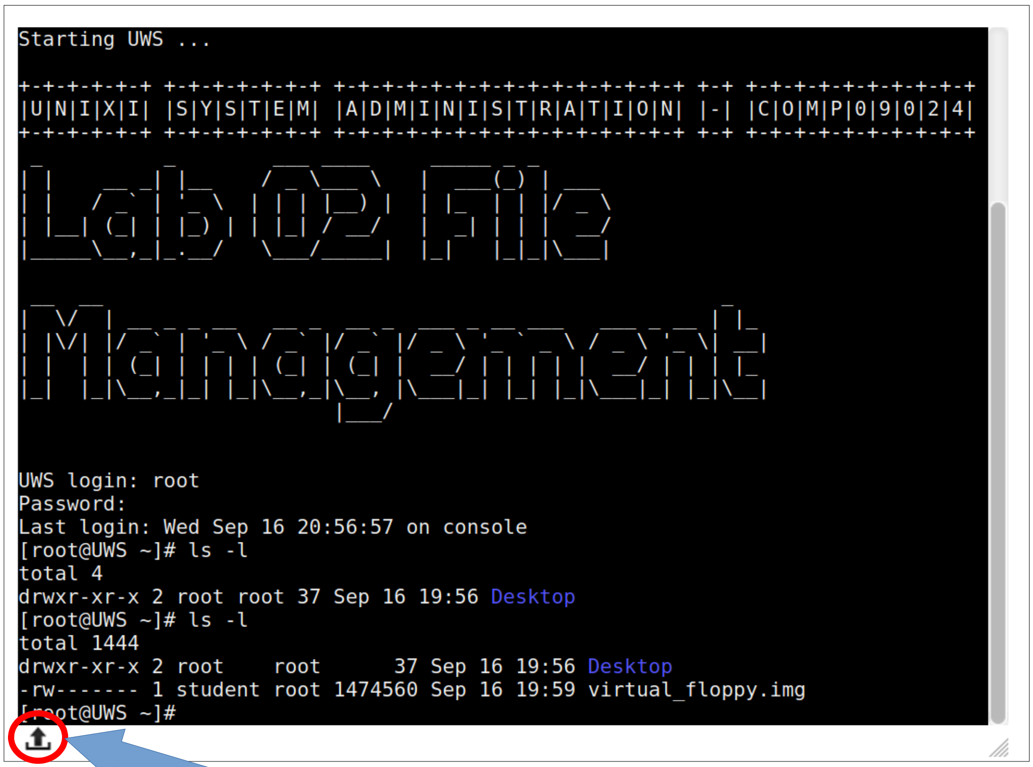
Traditionally, the **`fstab`** file was only read by programs, and not written. However, some administration tools can automatically build and edit **`fstab`**, which you might wish to do if you were required to install a new disk on a system.

The **`mount`** command itself, connects the device name (raw device name) with the mount-point. Please bear in mind that both entities exist independently of one another and that the mounting operation joins them together logically.

Now let us simulate inserting an hard disc with a ext2 filesystem data. Please follow the following steps:

1. Download the file **virtual_floppy.img** (a virtual floppy disk file) from MyUWS and place it anywhere on your system (e.g. Desktop).

2. Upload the **virtual_floppy.img** to the virtualized Linux running in your browser.



```
Starting UWS ...

+-+-+-+-+-+ +-+-+-+-+-+-+ +-+-+-+-+-+-+-+-+-+-+-+-+-+-+ +-+ +-+-+-+-+-+-+-+-+-+
|U|N|I|X|I| |S|Y|S|T|E|M| |A|D|M|I|N|I|S|T|R|A|T|I|O|N| |-| |C|O|M|P|0|9|0|2|4|
+-+-+-+-+-+ +-+-+-+-+-+-+ +-+-+-+-+-+-+-+-+-+-+-+-+-+-+ +-+ +-+-+-+-+-+-+-+-+-+
 _          _        ___ ____    _____ _ _
| |    __ _| |__    / _ \___ \  |  ___(_) | ___
| |   / _` | '_ \  | | | |__) | | |_  | | |/ _ \
| |__| (_| | |_) | | |_| / __/  |  _| | | |  __/
|_____\__,_|_.__/   \___/_____| |_|   |_|_|\___|

 __  __                                                   _
|  \/  | __ _ _ __   __ _  __ _  ___ _ __ ___   ___ _ __ | |_
| |\/| |/ _` | '_ \ / _` |/ _` |/ _ \ '_ ` _ \ / _ \ '_ \| __|
| |  | | (_| | | | | (_| | (_| |  __/ | | | | |  __/ | | | |_
|_|  |_|\__,_|_| |_|\__,_|\__, |\___|_| |_| |_|\___|_| |_|\__|
                          |___/

UWS login: root
Password:
Last login: Wed Sep 16 20:56:57 on console
[root@UWS ~]# ls -l
total 4
drwxr-xr-x 2 root root 37 Sep 16 19:56 Desktop
[root@UWS ~]# ls -l
total 1444
drwxr-xr-x 2 root    root      37 Sep 16 19:56 Desktop
-rw------- 1 student root 1474560 Sep 16 19:59 virtual_floppy.img
[root@UWS ~]#
```

3. The selected file will be uploaded to the "**/root**" folder.

4. Create the /media/floppy directory to mount the disk image:

```
[root@UWS ~]# mkdir /media/floppy
```

5. Mount the virtual drive to this location:

```
[root@UWS ~]# losetup /dev/loop0 /root/virtual_floppy.img
```

```
[root@UWS ~]# mount /dev/loop0 /media/floppy
```

6. Check **`/etc/mtab`** to confirm that the disk is mounted, just run the "**mount**" command.

```
[root@UWS ~]# mount
```

7. The virtual floppy contains the file "file.txt". Check its content:

```
[root@UWS ~]# cat /media/floppy/file.txt
```

Questions

- **Q2.1)** Identify and note the line in **`/etc/mtab`** that contains the mount-information associated with the disk device.
- **Q2.2)** 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.
- **Q2.3)** After mounting the virtual **`floppy`**, has **`/etc/fstab`** changed as well, or are all entries still the same?

Now it is time to put a real filesystem… a Linux one! onto the virtual floppy. We will create an ext4 filesystem on the floppy. Please note: **All data on the virtual floppy will be lost!**

Please bear in mind that although the **`mount`** and **`umount`** commands are referring to the actual mount points, the raw access to the floppy is given via its device entry **`/dev/loop0`**. I hope you understand why this is the case by now. Please set in your home directory, if you haven't already.

```
[root@UWS ~]# cd ~
```

and dismount the floppy:

```
[root@UWS ~]# umount /media/floppy
```

Now continue with:
(Before you type, make doubly sure that you only refer to **`/dev/loop0`** !)

```
[root@UWS ~]#  mkfs -t ext3 /dev/loop0
```

Now mount again the virtual floppy

```
[root@UWS ~]# mount /dev/loop0 /media/floppy
```

and check whether "file.txt" exists or not.

```
[root@UWS ~]# ls -l /media/floppy
```

<u>Questions</u>

- **Q2.4)** Why is it important not to have target media mounted while you try to create a file system? Think of e.g. the mayhem that could happen otherwise.
- **Q2.5)** How many inodes and blocks are created on the system?
- **Q2.6)** How many blocks are reserved for the super user (root)? Do you have any idea what they may be reserved for?

Now manually run the program `fsck` on the virtual floppy:

```
[root@UWS ~]# fsck /dev/loop0
```

<u>Question</u>

- **Q2.7)** What is displayed by `fsck`?

As you should have observed the output of `fsck` is quite pedestrian, since nothing is (should be) wrong with the just freshly created filesystem on `/dev/fd0`. Now it is time to re-mount and set in our newly created filesystem:

```
[root@UWS ~]# mount /dev/loop0 /media/floppy

[root@UWS ~]# cd /media/floppy
```

and create a file:

```
[root@UWS floppy]# echo 'Viva Linux' > /media/floppy/tst.dat
```

Now perform:

```
[root@UWS floppy]# ls -l /media/floppy
```

<u>Question</u>

- **Q2.8)** Note any files and directories that are on the floppy at this time.

You may notice that there is a `lost+found` directory on the floppy. Nearly every filesystem has a directory like this to hold any files that have lost their inode-to-content connection during a failed I/O operation. On demand these directories are used to place inodes that cannot be resolved by the `fsck` command. After the successful completion of `fsck` the user may have a look in this directory in order to resolve the 'orphans' of the filesystem and try to reconnect them into the filesystem. An empty or even non-existent `lost+found` directory is always a good indication of a healthy filesystem. A large one will indicate that something went terribly wrong!

Question

- **Q2.9)** Can you think of any apocalyptic scenario that might produce an entry in the **lost+found** directory?

Filesystems normally get corrupted during inappropriate shutdown or in cases of hardware failure. Again Unix-like systems would allow you to cope with the worst kind of scenarios, but this will be a task for a more advanced course.

Now let us try to dismount the floppy via:

```
[root@UWS ~]# umount /media/floppy
```

Questions

- **Q2.10)** Why would the kernel not allow you to dismount a filesystem that you are currently using?

- **Q2.11)** Use the command **cd /root**, before issuing the command line **umount /media/floppy** again. Does the un-mounting operation work now?

Now try to set into **/media/floppy** again via:

```
[root@UWS ~]# cd /media/floppy
```

and display the contents of the directory:

```
[root@UWS floppy]# ls -l
```

Questions

- **Q2.12)** Why can you still set on the floppy, even though you have just ejected the floppy disk?
- **Q2.13)** Explain why you no longer see the previous files **lost+found** and **tst.dat** on the mount-point?

Managing Files and Using Filters

Log in as root and make sure that you are in the home directory of the root user by:

```
[root@UWS ~]# cd  /root
[root@UWS ~]# pwd
```

First we will create a simple file with the echo command to illustrate the basic behaviour of the **echo** command within our working environment (the shell!). Try the following:

```
[root@UWS ~]# echo ‘1234567890’ test.dat
```

Now employ re-direction **>** ('greater-than' symbol) to re-direct a string into a file

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

<u>**Question**</u>

- **Q2.14)** Explain the difference between the two command lines given above. What does the re-direction operator **>** do?

Having created the file **`test.dat`**, we will rename it with the move command: **`mv`.**

```
[root@UWS ~]# mv test.dat test_01.dat
```

<u>**Question**</u>

- **Q2.15)** What happened to the original **`test.dat`** file?

Now let us continue with:

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

<u>**Question**</u>

- **Q2.16)** Examine the file **`test_01.dat`** with the **`cat`** command. What does the operator **>>** do?

Now we will copy the file **`test_01.dat`** to another file in our directory (could just as easily be any other directory within the file-system) with the copy command **`cp`**. Type the following lines:

```
[root@UWS ~]# cp test_01.dat test_a.dat
[root@UWS ~]# cp -i test_01.dat test_b.dat
[root@UWS ~]# cp -i test_01.dat test_b.dat
```

<u>**Question**</u>

- **Q2.17)** Based on the commands shown above: Explain the effect of the **`-i`** qualifier in the **`cp`** command.

It should be clear from the above that we have created two identical files called **`test_a.dat`** and **`test_b.dat`**. To check for identical files we can use the compare command **`cmp`**:

```
[root@UWS ~]# cmp test_a.dat test_b.dat
```

<u>**Question**</u>

- **Q2.18)** What is the output of the above command line, or is there no output at all?

Now extend **`test_a.dat`** by:

```
[root@UWS ~]# echo `ei caramba' >> test_a.dat
```

and use the **`cmp`** command again:

```
[root@UWS ~]# cmp test_a.dat test_b.dat
```

Question

- **Q2.19)** What is the output of the above **`cmp`** command now and why?

Now consider the 'concatenate' command **`cat:`**

```
[root@UWS ~]# cat test_a.dat
```

Question

- **Q2.20)** What is the output of the above command? Do you know another Unix command that does roughly the same?

Let us create a third test file by typing:

```
[root@UWS ~]# echo `Doh!' > test_c.dat
```

Now type:

```
[root@UWS ~]# cat test_a.dat  test_b.dat test_c.dat
```

after this try:

```
[root@UWS ~]# cat test_a.dat test_b.dat test_c.dat > test_abc.dat
```

Question

- **Q2.21)** What will **`cat`** do with the 3 files? Can you achieve the same result with the **`more`** command?

Now let us create a subdirectory with the command 'make directory' **`mkdir`**

```
[root@UWS ~]# mkdir temp10
```

after this type:

```
[root@UWS ~]# cp test_abc.dat ./temp10/.
```

Then examine the contents of `temp10`

Questions

- **Q2.22)** What does the `./` represent in the above command line and is it redundant here?
- **Q2.23)** 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/.` ?

These examples should have given you an insight in to how the basic file commands work. Of course this is only a glimpse of the things that you can do, but it helps a lot to know the basics. In order to get rid of our example files, we will use the delete command `rm` (short for ReMove).

```
[root@UWS ~]# rm –i test*
```

Questions

- **Q2.24)** Has the `-i` qualifier for `rm` the same meaning as for the `cp` command?
- **Q2.25)** What does the asterisk sign '`*`' represent in the above expression?

Managing Permissions

Every file has a set of associated permissions. These permissions determine what kind of rights users have in respect of the file. Access to all files (data files, programs, processes and devices) is organised around the concept of *users* and *groups*, and the idea of ownership and protection.

For any given file there is a user who owns that file. File access rights can be shared with different groups, each comprising any number of users. Linux provides three different types of file access:

read:	**r**
write:	**w**
execute:	**x**

and three basic classes of access:

user
group
world

As you may know, every user belongs to a particular group (which may consist of that user alone). World is a common name for all the users who don't belong to a given group. As you may have already noticed the `–l` option prints permissions information among other things:

The first character block (excluding the initial character) gives information about the permissions assigned to user, group, and world.

e.g. `-rwxr-xr--` means:

user: read, write, execute
group: read and execute
world: read (only)

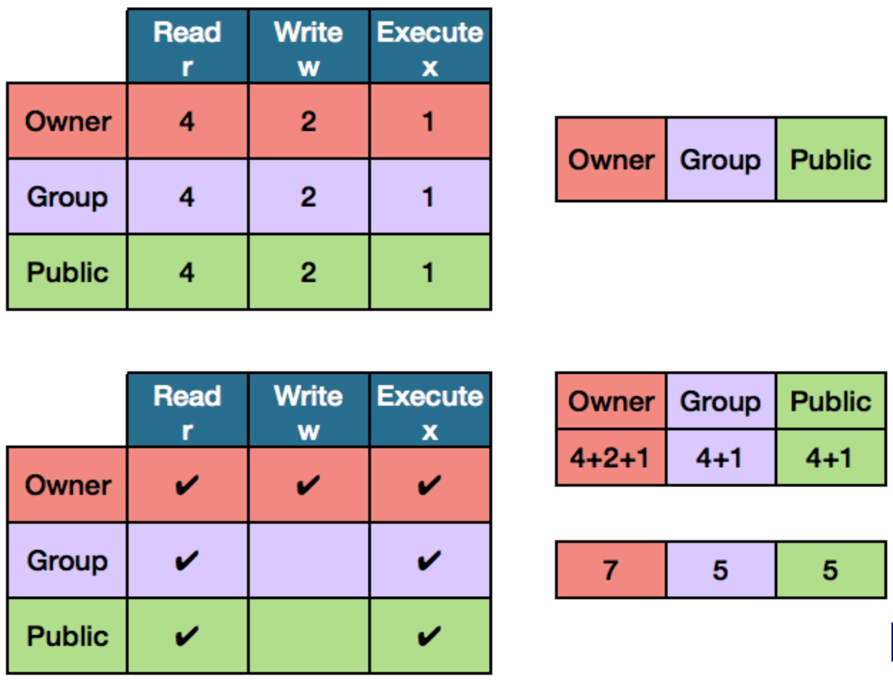
Withheld permissions are denoted by hyphens, e.g. two hyphens in a row appear as: `--`

<u>**Questions**</u>

- **Q2.26)** Write down the permissions (in words) of **`/etc/fstab`** and **`/etc/shadow`** in your logbook, and explain why the latter is so highly restricted.
- **Q2.27)** To which category, do you think the root-user belongs (user, group, world, or none of these)?
- **Q2.28)** Check the entries in **`/etc/group`** to find out which users in your system belong to the group 'audio'.

The actual coding of the permissions is done using a bit-wise notation. Permissions for user, group and world are each represented by a 3-bit binary number symbolising read, write and execute permissions, respectively. We actually use an octal representation (utilising eight symbols, 0-7) for each access class (2^3=8 combinations). When a binary bit is set to 1, this means that the corresponding permission is enabled. E.g. the write-only permission in bits would be **2** in octal and **010** in binary, which equals **$0*2^2 + 1*2^1 + 0*2^0 = 2$**. This bit-wise notation provides a convenient shorthand to change file permission using the **`chmod`** (change access mode) command.

Consult **`man chmod`** to get a first impression of the chmod command. The following graphic example shows the resulting permissions of **`chmod 755`**:



	Read r	Write w	Execute x
Owner	4	2	1
Group	4	2	1
Public	4	2	1

Owner	Group	Public

	Read r	Write w	Execute x
Owner	✔	✔	✔
Group	✔		✔
Public	✔		✔

Owner	Group	Public
4+2+1	4+1	4+1

7	5	5

Then continue in the **/root** directory with:

```
[root@UWS ~]# echo 'Donuts' > test_a.dat
```

Question

- **Q2.29)** Use the `ls -l` command to find out the default user, group and world permissions for this newly created file.

Continue with:

```
[root@UWS ~]# chmod 777 test_a.dat
[root@UWS ~]# chown student test_a.dat
[root@UWS ~]# chgrp www-data test_a.dat
```

Repeat `ls -l`:

Questions

- **Q2.30)** 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**?
- **Q2.31)** Is a permission of **7** (in octal notation) a good idea to enable for world-access?
- **Q2.32)** 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.
- **Q2.33)** What is the most permissive access that you can grant in octal representation?

Continue with:

```
[root@UWS ~]# chown root test_a.dat
[root@UWS ~]# chgrp root test_a.dat
```

Directory permissions operate a little differently, but the basic idea is the same. Make sure you are in the **/root** directory and create a second subdirectory called **temp11**:

Question

- **Q2.34)** What is the default permission given to the new directory **/root/temp11**?

The meaning of the read, write, and execute permissions when applied to a directory are:

read:	**list the files within the directory** (e.g. with a command like `ls`)	
write:	**alter directory contents**	(e.g. remove or add files to it)
execute:	**allows to enter the directory** (e.g. access files and directories inside)	

<u>**Questions**</u>

- **Q2.35)** Which directory permission will allow you to create a files inside of it ? (eg: executing **`touch`** and **`echo`** commands)
- **Q2.36)** Which directory permission is necessary to run a binary executable located inside of it?
- **Q2.37)** Which directory permission is necessary to run a shell-script located insede of it ? (a shell script is an ASCII code!)

An important point to note is that the permission to remove a file from a directory is independent (and effectively overrules) the permission relating to the file itself. If you grant write permission to group or world in respect of a directory you are asking for trouble, since group and world can then delete files in that directory. Contra-intuitively this is possible even if the file itself is protected!
Try:

```
[root@UWS ~]# chmod 755 /root
[root@UWS ~]# cd /root
[root@UWS ~]# cp test_a.dat temp11/.
[root@UWS ~]# chmod 733 temp11
```

the above command creates a fairly protected directory called **`temp11`**.

```
[root@UWS ~]# chmod  700  temp11/test_a.dat
```

this command creates a very protected file called **`test_a.dat`** within the protected directory.

```
[root@UWS ~]# cp temp11/test_a.dat temp11/test_b.dat
[root@UWS ~]# cd temp11
[root@UWS temp11]# chmod 777 test_b.dat
```

these three commands create an unprotected file **`test_b.dat`** within the protected directory.

Now try to log in as the (normal) **`student`**, who has no root privileges on the machine using the **`su`** command:

```
[root@UWS ~]# su -l student
```

then do the following:

```
[root@UWS ~]# cd /root/temp11
[root@UWS ~]# ls
```

<u>**Question**</u>

- **Q2.38)** 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?

Now continue by performing the following commands. For each command note the machine response in your logbook.

Question

- **Q2.39)** Write down the kernel response for each of the 3 '**`more`**'-command lines given below.

First, the highly protected file
```
[root@UWS ~]# more /root/temp11/test_a.dat
```

Then, the unprotected file
```
[root@UWS ~]# more /root/temp11/test_b.dat
```

Finally, a non-existent file
```
[root@UWS ~]# more /root/temp11/test_c.dat
```

Now do the same for the **`cp`** command (by trying to copy the files to the home directory of student)

First, the highly protected file
```
[root@UWS ~]# cp /root/temp11/test_a.dat /home/student
```

Then, the unprotected file
```
[root@UWS ~]# cp /root/temp11/test_b.dat /home/student
```

Finally, a non-existent file
```
[root@UWS ~]# cp /root/temp11/test_c.dat /home/student
```

Question

- **Q2.40)** Write down the response of the kernel to each of the 3 '**`cp`**'-command given above.

Please reflect on the "security" settings assigned to the directory and the files and their impact on commands like **`ls`**, **`cp`** and **`more`** for this directory and files by answering the following questions:

Questions

- **Q2.41)** Explain why you *can* copy a file, although you *can't* see it with the **`ls`** command.

- **Q2.42)** Can you create a file that can be copied by group and world users, residing in a directory with a permission setting of **`700`**? (Tip: change the directory permission for **`temp11`** accordingly and run the task again).

- **Q2.43)** 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.

- **Q2.44)** 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.

– END OF LAB –