COMP110 Practical 3
Access Control, Synchronisation and Remote Access

1 Introduction

This practical is intended to familiarise you with the file access control mechanisms of the file system used by the Department and relates to the following two module learning outcomes:

- To *effectively use relevant software packages* and *appreciate different types of software*;

- To effectively use general IT facilities including organising your file store, *taking advantage of access control and security features of operating systems.*

By default, your own files can only be accessed by yourself. It is important that you understand why this is so, how you can check who can access your files or the files of others, how you restrict or open access to files, and how you change what can be done with a file.

In addition, we will consider how you might best synchronise the files on the Departmental filestore with the files you have at home on a PC or laptop and how you can remotely access Departmental computing facilities.

This document can be found at

http://intranet.cse.liv.ac.uk/publish/COMP110/notes/practical03.pdf

While you work through the tasks below compare your results with those of your fellow students and ask one of the demonstrat

2 File Permissions

We will use the Department's Linux systems for most of the exercise   rm to connect to one of the departmental Linux systems.
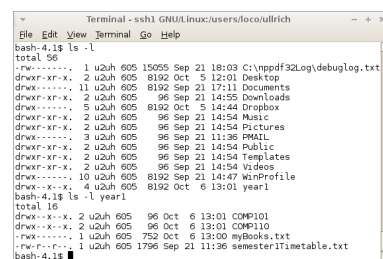
2.1 Overview

You have already seen that using `ls -l` you can get a '*long listing*' of the files in the current directory. If you have completed the exercises of Practical 1, then your home directory should contain a subdirectory called `year1` and files `semester1Timetable.txt` and `myBooks.txt` in that subdirectory. If you have used names different from these, then you need to adjust the exercises accordingly. Executing the commands

▶ `ls -l`
▶ `ls -l year1`

Figure 1: Permissions

should result in output resembling that shown in Figure 1 (remember: `$` is the command prompt, not a character that is part of the command that you need to enter; '`l`' is the letter ℓ, not the number '`1`').

As you can see, the output of `ls -l` consists of several columns with the right-most column obviously containing the names of files and directories. The third column from the left, the one containing `u2uh}` in Figure 1, indicates the *owner*. For your own files, the corresponding column

should show your used id. The fourth column from the left, the one containing 605 in Figure 1, indicates the *group*. Again, the group shown for your files will be different. The left-most column, a rather cryptic looking string of characters and dashes, shows the *permissions* (alternatively called *access rights*) for each of the files and directories.

All modern operating systems use *access control lists* to control who can do what with a particular file system object. To this end, each file system object is associated with such an access control list that contains *access control entries*, each of which gives an individual user or group the right to perform an operation such as reading, writing, or executing the file system object.

Linux, like any traditional UNIX operating system, recognises three classes of users with respect to operations on file system objects: *owner*, *group*, and *other*. Operations are categorised as *read* (r), *write* (w), and *execute* (x). Finally, the file system categorises file system objects into various kinds of objects, including *files* and *directories*. Having read (r), write (w), and execute (x) permission takes slightly different meaning for files versus directories:

Permission		For a file	For a directory
read	(r)	allowed to view file contents	allowed to view directory contents
write	(w)	allowed to write to file	allowed to remove or add new files to directory
execute	(x)	allowed to execute file	allowed to access files in the directory

Some clarification is in order regarding permissions for directories:

- To remove or add a file to a directory, or to otherwise modify an already existing file in a directory, you actually need both write (r) and execute/access (x) permission for the directory.

- Similarly, while read (r) p   es indeed allow you to see what files are i   you that you do not have ac   ee any additional information for these files, e.g. their owners or the p

So, what does the information shown in Figure 1 and repeated below mean for the files and directories involved?

```
drwx--x--x.  year1
-rw-r--r--.  semester1Timetable.txt
-rwx------.  myBooks.txt
```

- The first character indicates the *type* of the file: 'd' stands for 'directory', '-' for 'regular file', 'l' for link, etc.

  So, year1 is a directory, while semester1Timetable.txt and myBooks.txt are regular files.

- The next block of three characters indicates the permissions that the owner of the files has.

  So, 'rwx' for year1 and myBooks.txt means that the owner has read, write and execute permission. Remember, since year1 is a directory while myBooks.txt is a regular file, these permissions have a slightly different meaning for each of these.

  For semester1Timetable.txt, 'rw-' indicates that we have read and write permission, but not execute permission.

- The next block of three characters indicates the permissions for members of the group.

  For year1, the three characters are '--x', so the group 605 has execute permission for this directory and only execute permission. For semester1Timetable.txt, the three characters are 'r--', so the group 605 has read permission. Finally, for myBooks.txt, the three characters are '---', so the group 605 has no rights with respect to this file.

- The next block of three characters indicates the permissions for other users.

  As it happens, for the three file system objects considered here, as the third block of three characters is identical to the second block of three characters, other users have the same permissions as the group `605`.

2.2 Changing File Permissions: Symbolic Notation

To change the permissions for file system objects you use the `chmod` command. In is simplest form, `chmod` takes two arguments:

1. A description of the permissions that you want to set or a description of how you want to change the existing permissions. This description can be given in symbolic notation or numeric notation.

   (a) In symbolic notation, you need to specify for which user or group of users you want to change the permissions, how you want to change them, and which permissions you want to change:

Which user?		How to change the permission?		Which permission?	
`u`	user/owner	`+`	add this permission	`r`	read
`g`	group	`-`	remove this permission	`w`	write
`o`	other	`=`	set exactly this permission	`x`	execute
`a`	all (of t				

   For example, '   ' m the owner, i.e. from you   mission for all users, including yo

   Regarding the first and third group, you can pick more than o   , '`ug+rw`' means that for both user/owner and group you wa permission.

2. A list of file system objects for which you want to change the permissions, with a space separating the files system objects within the list.

Putting both together,

▶ `chmod u-x year1/myBooks.txt`

means that you want to remove execute permission from the owner of the file `year1/myBooks.txt`. Execute the command above and use

▶ `ls -l year1`

to see what the effect of the command has been. The output from `ls -l year1` should resemble that shown in Figure 2: The permissions for the owner of the file should have changed from '`rwx`' to '`rw-`'.

In Figure 1 we have seen that the permissions for the file `semester1Timetable.txt` give read permissions to all users (owner, group, and others). Such generous read permissions should be avoided if at all possible as you are required to keep your files secure from unauthorised access[1].

Figure 2: chmod (1)

---

[1]For example, if a file that is readable by all users would contain your work for a COMP101 assignment, then

To restrict the read permissions for `semester1Timetable.txt` use the command

```
$ chmod og-r year1/semester1Timetable.txt
```

and see what the effect is using `ls -l year1` (see Figure 3).

Let us see what the default permissions are for a newly created file. To create such a file you can use the `touch` command:

► `touch year1/newFile.txt`

will create a new file called `newFile.txt` in the directory `year1`. Execute this command and use

► `ls -l year1`

to see what the permissions for `newFile.txt` are (see Figure 4).

As you can see, the file is readable and writable by the owner, i.e. yourself, but by nobody else. If you were to create a new file under Windows, then by default the execute permission would also be set for the owner of the file. Give it a try.

So far we have not seen an example of an executable file. Let us create one.

Using your favourite editor, e.g. gedit, create a new file in your home directory, called `myFirstShellScript` with the following content (also see Figure 5):

```
#!/bin/sh

echo "Hello World!"
```

Here, the first line indicat
to execute the rest of the file, namely, the file `/bin/`
tem's default shell. That will be the GNU Bourne Again Shell or
bash for short.

Check with `ls -l` what the permissions are for `myFirstShellScript` once you have saved it. Not surprisingly, it is readable and writable by the owner, but nothing else.

Try to execute the file by using the command

► `./myFirstShellScript`

in the same directory where this file is stored. You should get an error message telling you that you do not have permission to execute the file. This is correct as so far nobody has execute permission for this file.
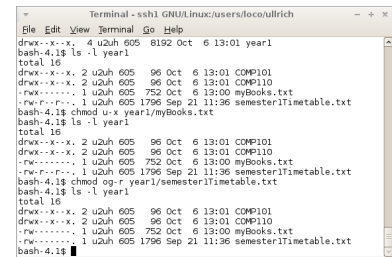
Let us change that using the command

► `chmod u+x myFirstShellScript`

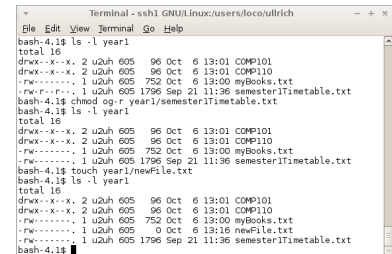Then try to execute `myFirstShellScript` again. This time you will succeed and the script will produce the output

`Hello World!`

See Figure 6.

Figure 3: chmod (2)

Figure 4: chmod (3)

ure 5: Shell script

Figure 6: Executing files

---

all students could potentially see what you have done and one of them could submit a copy as their own work. We would consider that to be collusion involving both parties, i.e. yourself and the other student, not plagiarism.

Now that you know how to change the permissions of a file system object, you can check whether what has been said on page 2 about permissions for directories is true. Do the following:

1. Change the permissions for directory `year1` to '`r--------`', i.e. read permission for the owner only, no other permissions for the owner, no permissions for group or other.

   Then use `ls -l year1` to see whether you can still obtain a long listing of the contents of the `year1` directory.

2. Now give yourself execute permission for `year1` in addition to read permission, i.e. set the permissions to '`r-x------`'.

   Use `ls -l year1` again to see what the long listing of the contents of the `year1` directory looks like.

3. Next, see whether write permission alone allows you to create a file in the the directory `year1`. To do so, execute the commands

   ▶ `chmod u=w year1`
   ▶ `touch year1/testFile`

   The system should deny you the permission to create `testFile`.

4. Adding execute permission to the directory should solve this problem.

   ▶ `chmod u+x year1`
   ▶ `touch year1/test`

   This time creating the file

2.3 Changing File Permissions: Octal Notation

`chmod` also allows to change permissions using a numeric notatio

▶ `chmod 640 year1/semester1Timetable.txt`

will give the owner of `year1/semester1Timetable.txt` read and write permission, the group read permission, and others no permissions.

In numeric notation, permissions are given by three digits: The first digit is for owner/user permissions, the second digit for group permissions, and the third digit for the permissions of others.

Each digit is the sum of one or more of the following values:

4	set read	permission
2	set write	permission
1	set execute	permission

In our example above, the first digit was `6`, the sum of `4` and `2`. Thus, we were instructing `chmod` to set read and write permission for the owner. The second digit was `4`, so the group was given read permission. Finally, the third digit was `0`, meaning other users have no permissions with respect to the file.

Execute the command `chmod 640 year1/semester1Timetable.txt` and check whether the permissions change as described above. Then, using numeric notation for permissions, set the permissions for the file `year1/semester1Timetable.txt` back to '`rw-------`'.

3 Synchronisation

You may have your own PC and/or laptop that you want to use for your studies, in which case you will have to transfer files between the Departmental filestore and your PC/laptop. You might use a USB pen drive, but students quite often forget these, or forget to copy all the necessary files onto the pen drive, or the pen drive gets lost or ceases to work.

An alternative is the use of `rsync` or of MobaXterm to transfer files between a PC or laptop at home and the departmental computers.

3.1 rsync

`rsync` is a program that transfers files and directories from one location to another in a way that minimises the amount of data that needs to be transferred. The two locations do not necessarily have to be on the same computer. `rsync` has several advantages over other, simpler, file transfer programs: (i) `rsync` can restrict transfers to files that have been changed instead of blindly transferring all files and (ii) even where a file has to be transferred, `rsync` will by default only transfer parts of the file that have changed, not the complete file. This not only minimises the amount of data that needs to be transferred but also minimises the time the transfer takes.

The protocol that is typically used for the transfer of files between locations on different computers is SSH (Secure Shell), a cryptographic network protocol. SSH uses the client-server model and requires that on the computer that will act as server for the transfer of files is running a SSH daemon that accepts con

in the Department, with two L

Department: `ssh1.csc` ... `ssh2.csc.liv.ac.uk` (aka `linux2.csc.liv.ac.u`

Let us first assume that your PC/laptop at home is also using a Linux all likelihood `rsync` will already be installed. Let us use the back ... s also assume that your Departmental username is u6xyz. Finally ... ental filestore you have all currently relevant files in a subdirectory `year1` of your home directory while on your own PC/laptop the corresponding files are in `~/uni/year1/`.

Then to synchronise the contents of these two directories and all their subdirectories, execute the following command in a terminal window on your PC/laptop at home:

▶ `rsync -auvz -e ssh 'u6xyz@ssh1.csc.liv.ac.uk:year1/' ~/uni/year1/`
`u6xyz@ssh1.csc.liv.ac.uk's password:`

After you have entered your Departmental password, changed files will be transferred, for example:

```
receiving incremental file list
year1Timetable.txt
myBooks.text
sent 614 bytes received 278252 bytes 17991.35 bytes/sec
total size is 893970880 speedup is 3205.74
```

The options `-auvz -e ssh` tell `rsync`:

- to transfer files in archive mode (`-a`), which ensures that symbolic links, devices, attributes, permissions, ownerships, etc are preserved in the transfer,

- to skip files that are newer on the receiver (`-u`),

- to show which files are transferred (`-v`),

6

- to compress the data that is to be transferred (`-z`) and
- to use ssh as network protocol (`-e ssh`).

Used in this way, `rsync` does not delete files on the receiver, i.e., your PC/laptop, that have been deleted on the Departmental systems. In order to delete those files on the receiver, the `--delete` option can be used:

▶ `rsync --delete -auvz -e ssh 'u6xyz@ssh1.csc.liv.ac.uk:year1/' ~/uni/year1/`

Obviously, this operation can easily lead to the loss of files. To see which files will be deleted on the receiver, without them being deleted right away, one can ask `rsync` to perform a dry run by adding the `-n` option to the command:

```
▶ rsync -n --delete -auvz -e ssh 'u6xyz@ssh1.csc.liv.ac.uk:year1/' ~/uni/year1/
receiving incremental file list
deleting HelloWorld.java
sent 23 bytes received 18445 bytes 7387.20 bytes/sec
total size is 417446325 speedup is 22603.76 (DRY RUN)
```

Finally, if you have changed files on your PC/laptop and want to transfer those files back to the Departmental systems, you simply have to swap destination and source when using `rsync`:

▶ `rsync -auvz -e ssh ~/uni/year1/ 'u6xyz@ssh1.csc.liv.ac.uk:year1/'`

The only drawback of `rsync` is that you must not forget this step if you want continue to work on those files while in the Department.

The manual page for `rsync` contains a description of all the options of `rsyn`.

To experiment with `rsync`:

1. Create directories `~/uni/` and `~/uni/year1`.
2. Execute the command

    ▶ `rsync -auvz -e ssh 'u6xyz@ssh1.csc.liv.ac.uk:year1/' ~/uni/year1/`

    replacing `u6xyz` with your own user id and making adjustments to the directory names as necessary.

3. Inspect the contents of `~/uni/year1`. Does it now contain all the files and subdirectories of `~/year1`? Compare the file permissions and modification times of those files and subdirectories with those of the sources files and subdirectories.

4. Change the modification time of `~/year1/myBooks.txt` and add a new file to `~/year1/comp110/` by executing the commands

    ▶ `touch ~/year1/myBooks.txt`
    ▶ `ls -l ~/year1/ > ~/year1/modules.txt`

5. Execute the command

    ▶ `rsync -auvz -e ssh 'u6xyz@ssh1.csc.liv.ac.uk:year1/' ~/uni/year1/`

    again, with the necessary modifications. See what files are transferred.

6. Create a file `~/uni/year1/coursework.txt` by executing the command

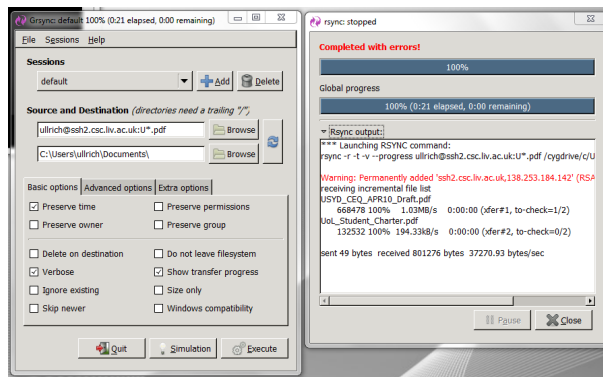   ▶ `cp ~/uni/year1/myBooks.txt ~/uni/year1/coursework.txt`

Figure 7: Grsync for Windows

7. Use a dry-run of `rsync` to see what were to happen if you sync `~/uni/year1` back to `~/year1`:

   ▶ `rsync -n -auvz -e ssh ~/uni/year1/ 'u6xyz@ssh1.csc.liv.ac.uk:year1/'`

8. Finally, use another dry-run of `rsync` to see what were to happen if you sync `~/year1` with `~/uni/year1` with the `--delete` option:

   ▶ `rsync -n --delete -auvz -e ssh 'u6xyz@ssh1.csc.liv.ac.uk:year1/' ~/uni/year1/`

   You should see that `~/uni/year1/coursework.txt` would be deleted.

9. Delete the directory

   Let us now assume that [...] ed to install `rsync`. There are var [...] sync for Windows in the following. The installation file for Grsync is available for downl

   `http://[...]rsync-[...].sou[...]`

Grsync provides a graphical user interface for `rsync` [...] nc has been completed, it can be started from the Start Menu, and it presents itself as shown in Figure 7. Grsync provides two input fields that are used to specify source and destination. The syntax for the specification of remote sources is the same as for `rsync`. So, as in our examples for the use of `rsync` on Linux systems, in order to transfer files from subdirectory `year1` in your home directory on the Departmental systems, you would specify the source as `u6xyz@ssh1.csc.liv.ac.uk:year1/`. To specify the destination directory, simply click on the '`Browse`' button to the right of the destination input field and navigate to the desired directory in the file selector window that opens. In order the specify whether permissions and ownership should be preserved and whether files present on the destination but not on the source should be deleted, `Grsync` provides check boxes. Make sure that the right options are set, in particular, that 'Skip newer' is set.

The graphical user interface will invoke `rsync` once you press the '`Execute`' button, pressing the '`Simulation`' will start a dry run. Two additional windows will open, one showing the progress of the synchronisation process, another asking you to enter your password for the Departmental systems. Once the password has been entered and accepted, the progress window will show diagnostic output by `rsync`.

Note that Grsync is also available for Linux systems. Again, if Grsync is not already installed on your PC/laptop, use the Package Manager to do so.

For further help on the use of both the Windows and the Linux port of Grsync it is best to refer to the manual pages for `rsync`.
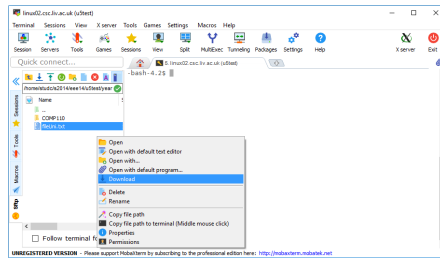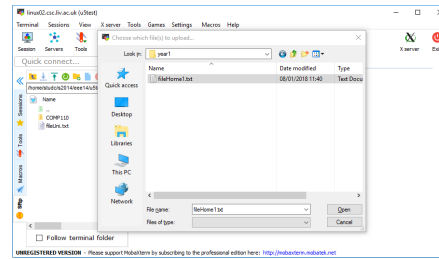
(a) Download file   (b) Upload file

Figure 8: Transferring files using MobaXterm

3.2 MobaXterm

We have already mentioned in the last practical that MobaXterm also allows you to transfer files. Let us quickly see how that works. First, create two new files using the commands

▶ `ls -1 $HOME > ~/year1/fileUni.txt`

▶ `ls -1 $HOME > ~/uni/year1/fileHome.txt`

We want to transfer `fileUni.txt` to `~/uni/year1/` using MobaXterm. To do so, in the file browser in the left pane of MobXterm, find the directory `year1`, double-click on it to open the directory in the file browser. Then right-click on the file `fileUni.txt`. This opens a menu (Figure 8a) in which you select "Download". Another file browser opens which allows you to select the directory to which you want to downloa `~/uni/year1/`.

To upload the file the directory in the file browser of MobaXterm. n, another file browser opens which allows you to select the file or files tha the file browsers to select the file `fileHome.txt`, then c obaXterm will then transfer the file. Explore the other options that MobaXterm directories on the remote system, delete files, and create new files.

4 Remote access

The two Linux systems `ssh1.csc.liv.ac.uk` (aka `linux1.csc.liv.ac.uk`) and `ssh2.csc.liv.ac.uk` (aka `linux2.csc.liv.ac.uk`) also provide you with a possibility to remotely access Departmental computing facilities.

If your own PC/laptop is running Linux and has SSH installed, then executing the command

▶ `ssh ssh1.csc.liv.ac.uk`

in a terminal window will establish a secure terminal session to `ssh1`. MacOS also comes with its own implementation of SSH and the command shown above should also work under MacOS.

If your own PC/laptop is running MS Windows, you will first have to download and install an SSH client. Two alternatives are MobaXterm and PuTTY, with MobaXterm being the preferred alternative. You can find these at

● `http://mobaxterm.mobatek.net/download-home-edition.html`

● `https://the.earth.li/~sgtatham/putty/0.70/w64/putty-64bit-0.70-installer.msi`

After installing a SSH client, follow the instructions at `http://cgi.csc.liv.ac.uk/guides/network/ssh/ssh.html` on how to establish a remote terminal session to `ssh1` or `ssh2`.

apropos	aspell	bash	cat	cd	chsh	clear	cp	date
diff	diffpp	egrep	enscript	find	history	kill	ln	mv
quota	rename	sort	ssh	tail	tar	tcsh	time	top
uptime	vim	zile	zip					

Table 1: Useful Linux commands

Also, only low impact jobs should be run on the two externally-visible systems `ssh1` and `ssh2`. Anything that might run for a long time or might require significant memory, should be run on `linux01` to `linux08`. These servers can be reached via SSH from `ssh1` and `ssh2`, that is, once you have established an SSH session with one of `ssh1` or `ssh2`, you can use the command `ssh linux01` within the MobaXterm or PuTTY terminal to connect to `linux01`. Analogously for all other Linux systems within the department.

Note also that if you are using MobaXterm at home, you can interact with the Linux system that you are connected to from a laptop or PC at home in the same way as you can from a Windows system within the department. So, you can remotely edit files or upload files that you have created at home and you can execute commands on the departmental Linux systems.

The best way of working remotely will depend on the speed and reliability of your internet connection. If your connection is fast enough, then you might be able to open an editor with graphical user interface on `ssh1` or `ssh2` and edit files directly on our systems. With a slower connection you might want to use a command line editor instead. Note that you can have more than one connection / termin
command line editor a
programs. The third altern
create on your system.

5 Further Study

This concludes our short introduction to Windows and Linux. There is obviously still a lot to learn, even about the commands that you have used in these practicals.

To explore the possibilities of Linux further you can first of all take advantage of the *manual pages* that are available for almost every Linux command. In a terminal, manual pages can be read using the command `man`. In its simplest form `man` takes a single argument, namely, the name of the manual page that you would like to read; the name of the manual page is typically identical to that of the corresponding command or program. So, for example,

▶ `man chmod`

will show you the manual page for the command `chmod`. To display this manual page, `man` uses `less -s` by default. So, you can use the key bindings for `less` to navigate through a manual page. Of course, you can use

▶ `man less`

to learn what those key bindings are. Table 1 lists a number of other commands that you should explore by reading their manual pages and experimenting with them.

To learn more about the Linux command line and shell scripting you can refer to the following e-book available from the library:

W. E. Shotts: The Linux command line: A complete introduction.

No Starch Press, 2012. `http://library.liv.ac.uk/record=b2626812~S8`
[last modification 16 December 2012, accessed 02 October 2015]

For the Gnome Desktop additional help is available via 'System > Help' in the tool bar at the top of the desktop. Gnome tools will typically also have a 'Help' button in their tool bar that allows you to access the manual pages for that specific tool.

Longer term you should aim to be able to install and set up an XAMPP stack (Apache HTTP Server, MariaDB or MySQL, PHP, Perl) on a Linux Virtual Machine under Windows 7/10 as well as directly on a Windows 7/10 system. You will not be taught how to do this, but by the beginning of the final year you should have acquired the necessary knowledge and skills through self-study. VirtualBox, a free and open-source hypervisor for x86 computers, is installed on our Windows 10 PCs and ready-to-use virtual machines can be found at `https://virtualboxes.org/`. Alternatives are Hyper-V and VMware Workstation Player, each with their advantages and disadvantages.

Assignment Project Exam Help

https://eduassistpro.github.io/

Add WeChat edu_assist_pro