COMP110 Practical 2
Using the Departmental Linux Systems

1 Introduction

This practical is intended to familiarise you with the departmental Linux systems 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.

The tasks described below will guide you through the process of accessing and logging onto a Linux system, using the Linux desktop, using command line interfaces, editing text files, and creating, compiling and running Java programs under Linux.

This document is available in PDF format at

`http://intranet.csc.liv.ac.uk/~ullrich/COMP110/notes/practical02.pdf`

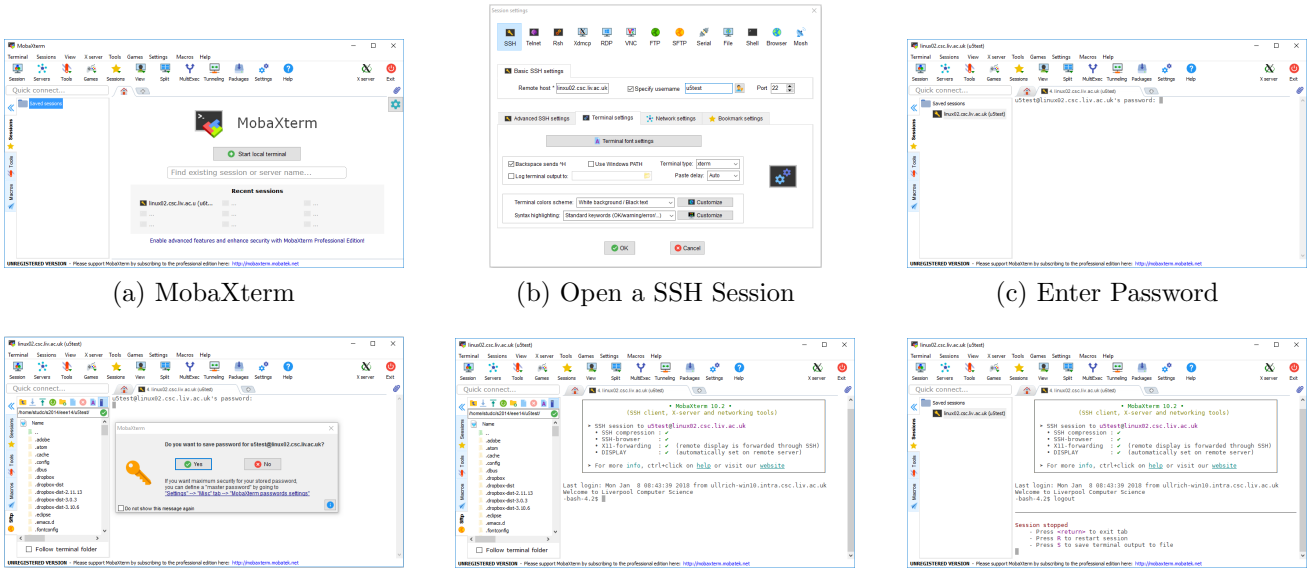
While you work through the tasks below compare your results with those of your fellow students and ask one of the demonstrators for help and comments if required.

2 Logging in to the Linux systems

The lab PCs running Windows are only part of the department's computing facilities. There are also a number of systems running the Linux operating system (currently Scientific Linux 7.4), which are available to all members of the department. These machines are not physically accessible, but can be accessed over the network. You use the same (Computer Science) username and password to log in to both the Windows and Linux systems (and also the Apple Mac lab), and they all provide access to the same shared filestore.

Figure 1: MobaXterm Shortcut

Commence by logging in to the Windows PC in the same way as you did previously. Now double-click on the "`MobaXterm`" shortcut (Figure 1) on the left-hand side of the desktop to open the MobaXterm application (Figure 2a). Click on "`Session`" in the toolbar, this will open a window for session settings. In the toolbar of that window click on "SSH", this will open a new window in which you can enter the connection details for a SSH connection to one of our Linux PCs. In the text field to the right of the label "`Remote Host`" enter the name of a Linux PC, `linux01.csc.liv.ac.uk` to `linux06.csc.liv.ac.uk`, click on the button to the left of "`Specify username`", then enter your departmental username into the text field to the right of that label. Click on the tab "`Terminal Settings`" below the text fields (Figure 2b). You will find an option "`Terminal colors scheme`" with a drop-down menu to the right of it. In that menu chose the option "`White background / Black text`" (unless you are really into retro colour schemes, in which case you should start with "`Black background / White text`" and customise that to use green text). Then click on the "`OK`" button at the bottom of the window. A new tab will open in the main window pane of MobaXterm in which you will see a prompt asking for your password (Figure 2c). Enter the password for your departmental account. You will then be asked whether you want MobaXterm to store your password so that next time you connect to this particular



(a) MobaXterm (b) Open a SSH Session (c) Enter Password

(d) Save Password (e) SSH Session Opened (f) Logging Out

Figure 2: Using MobaXterm

Linux PC you do not need to enter your password again (Figure 2d). It is up to you whether you do so, it is the typical trade-off between convenience and security. Independent of the choice you make you now see a shell prompt in the main window pane of MobaXterm (Figure 2e). Also, note that on the left to the main window pane you have a pane with a file browser showing the files in your home directory on the Linux PC.

3 Using the Command Line Interface

As mentioned, in the main window pane of MobaXterm you see a *shell prompt*, typically, `bash-4.2$`. This prompt, which will be represented throughout this practical by ▶, is followed by a square or underline—the *cursor*. Anything you type on the keyboard will appear here. You can use the left and right arrow keys to move the cursor back and forth within the command you have entered, and this can be used to correct typing errors.

When you have finished typing the command, press the <RETURN> key. This tells the shell to run the command you specified, and report the results. It will then display another prompt, ready for your next command. Here is an example that uses the `hostname` command to discover which Linux machine you are using:

```
▶ hostname
linux06.csc.liv.ac.uk
▶
```

Throughout the rest of this practical, we will not always show the command prompt in the examples. However, if you type in a command, press RETURN, and no new command prompt appears, then either you have not completed the command yet, say, there is an open string that you have not closed yet, or the command is still running, say, you have opened an editor 'in the foreground' and have not closed it yet. There are then three possibilities:

- If the command is not being executed yet, then you can try to properly complete the com-

mand and press RETURN again.

- If the command is being executed, but it does not terminate, then you can terminate the execution of the command by using the key combination CTRL-C. Note that if the command you are executing is an editor or software development environment, the text/program you were working on might be lost.

- If the command is being executed, but it does not terminate, then you can suspend the execution of the command by using the key combination CTRL-Z. You should then see a command prompt again. With the command bg you can move the just-suspended command to the background where it continues the execute, or, with the command fg bring it back to the foreground. The combination CTRL-Z followed by the command bg is the most reasonable cause of action if the command you are executing is a text editor, software development environment, or similar.

3.1 Basic Commands

The simplest commands are those concerned with viewing and manipulating files and folders. The following series of commands illustrate this for both Linux and Windows. Try working through each set in turn and watch what happens using the file manager (note: 'l' is the letter ℓ, not the number '1'). The examples assume that in Practical 1 you have created a directory structure exactly as suggested. You have to adjust the examples if you have used different directory names or a different directory structure.

Linux	MS Windows	
`cd`	`cd H:`	Start in your home filestore
`pwd`	`cd`	Show the *current directory*
`ls -l`	`dir`	Long listing of the files and folders in this directory
`cd year1/comp110`	`cd year1\comp110`	Move into a sub-folder
`ls`	`dir /w`	Short listing of files and folders in this directory
`cd ..`	`cd ..`	Move back up a level (.. = "parent directory")
`cat myBooks.txt`	`type myBooks.txt`	View the contents of a file
`more myBooks.txt`	`more myBooks.txt`	View the contents one page at a time (press the 'q' key to quit)
`cp myBooks.txt test`	`copy myBooks.txt test`	Make a copy of a file
`mv test test1`	`rename test test1`	Rename a file
`mkdir myDir`	`mkdir myDir`	Create a new folder (directory)

Linux	MS Windows	
`cp test1 myDir`	`copy test1 myDir`	Make a copy of a file in a different folder
`mv test1 myDir/test2`	`move test1 myDir\test2`	Move a file into another folder (and rename it)
`ls myDir`	`dir myDir`	List the contents of a folder
`rmdir myDir`	`rmdir myDir`	Try to delete a (non-empty) folder (which fails)
`mv myDir/* .`	`move myDir\* .`	Empty the folder (by moving the contents to the current directory)
`rmdir myDir`	`rmdir myDir`	Delete the (empty) folder
`rm test1 test2`	`del test1 test2`	Delete the test files

Be careful when using `rm`. Linux assumes you know what you are doing. So, if you ask it to delete a file, then it will be deleted. It will *not* simply be moved to the "`Wastebasket`" folder (which is what the file manager does). So it's typically not possible to "undelete" a file that has been deleted by mistake—when it's gone, it's gone. The same holds for Windows and `del`.

Note that Linux uses a filesystem with *case-sensitive* identifiers—the folder `Year1` is different to one called `year1`. Be very careful to type the names of files or folders *exactly* as they appear in the file manager, or in the output of `ls` or `dir`. Windows is more forgiving, and will ignore case differences.

Also be aware that by default Windows Explorer will often hide the *filename extension* - the last three or four characters after the final `.`. When using the command line, you typically need to give the full filename, *including* the filename extension.

3.2 Wildcards

The third-from-last command above (emptying the test directory before deleting it) illustrates a new idea—the use of *wildcards*.

Most of the commands above specify the name of a file or folder to work with. And as the last command shows, it's often possible to manipulate several files at the same time, by listing them on the command line. But if there are large number of files to manipulate, typing all of them out would be both time consuming and subject to errors. So both Linux and Windows provide a wildcard mechanism, to match the names of multiple files (or folders) at once.

Create two copies of the textbook file, using `cp myBooks.txt myFile1` and `cp myBooks.txt myFile2`. Note that you may have given the textbook file a different name and that it may be in one of the subdirectories of your home directory. Make sure that you are in the right directory and that you adjust the commands to fit with the filename that you have used.

Next, try the following sequence of commands. Think about the results you see, and what these patterns might mean.

- `ls myFile?`
- `ls *`
- `ls *.*`
- `ls *.txt`
- `ls myFile*`

```
▶ ls *File?
```

Also try the corresponding commands with the Windows command line (using `dir` instead of `ls`).

3.3 Redirection

Another useful technique is *redirection*, where the output of a command can be saved to a file.

```
▶ ls myFile* > out.file
▶ cat out.file
```

This can be very useful when you come to test your Java programs. But note that any errors will still be displayed

```
▶ ls yourFile* > out2.file
ls:  cannot access yourFile*:  No such file or directory
▶ cat out2.file
```

Redirection works on both Linux and Windows command line terminals.

The main problem with this approach is that it only saves the *output* of the program, and not anything that is typed on the keyboard. Redirecting output to a file also means that you will not see any prompts or instructions that might be displayed by the program. In one sense, this does not really matter - as long as you know what information you need to supply, you can type this "blindly" and the program should run correctly.

But it would be better if you could see the output (including any prompts) and still have everything saved to a file. On Linux, this can be done using the `script` command:


```
▶ script -c "command" out3.file
```

(where the `-c` option specifies the command to be run and capture). Try experimenting with this, using some of the commands above. (Although it will only really become useful when you start running programs that expect input from the keyboard). Note that `script` is only available on the Linux systems - there is not an equivalent mechanism under Windows.

3.4 Pipes

A variation of output redirection is the idea of a "pipe" - using the output of one command as the input to another. Try the following in the Linux terminal window

```
▶ ls | sort -r
▶ cat myBooks.txt | tail -5
▶ ls /lib | less      (press the 'q' key to quit)
```

(The symbol '`|`' is to the left of the character 'z' on UK keyboards.) Compare the results of these pipeline commands, with the output generated by the first command in each pipeline on its own. Also, find out what the difference between `less` and `more` is.

Pipes are also available under Windows, though they are much less widely used. The last command would work in much the same way (`dir C:\Windows\System | more`). Windows does not typically include the same range of "filter" commands such as `sort`, `head` and `tail`.

Most (traditional) Linux commands are designed to process the contents of the specified files, or (if no files are listed) to work on "standard input" as part of just such a pipeline of commands.

3.5 Command Line History

When developing a computer program, you will typically find yourself repeating the same sequence of commands again and again - editing the file containig the source code, compiling this file, running the resulting program, editing the file to fix any errors, compiling the corrected file, running the program again, and so on. This means that you will end up typing the same commands over and over again.

Both Linux and Windows command shells include a *command history* mechanism, which remembers the previous commands that you have typed and allows you to recall them and run them again. Try using the up and down arrow keys to step through this list.

3.6 Filename Completion

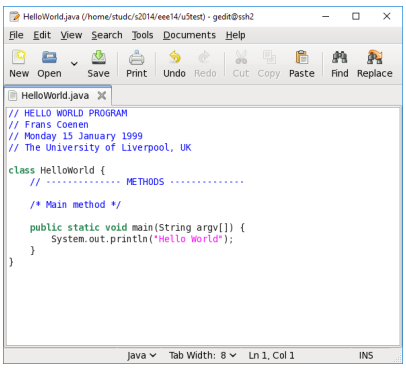
There is one final function of the shell to mention, namely, *filename completion*.

Quite often you will have several different files in the same folder, with significantly different names. The Linux shell allows you to type the first few characters of a filename (sufficient to uniquely identify that file), and then hit the `<TAB>` key. This will automatically complete the name of the file, just as if you had typed it at the keyboard. This is extremely useful - particularly if you are using meaningful filenames (which can be relatively long), or if your typing is not particularly accurate!

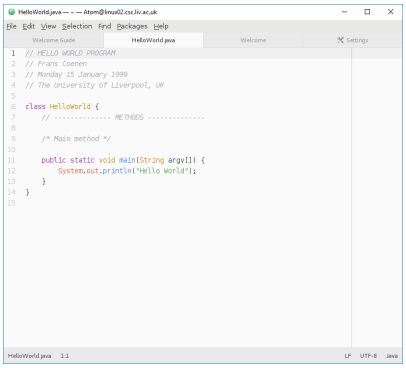
If the prefix you have supplied is not unique, and there are two files that could possibly match, the shell will complete as much as it can, and leave you to complete it.

4 Editing Text Files

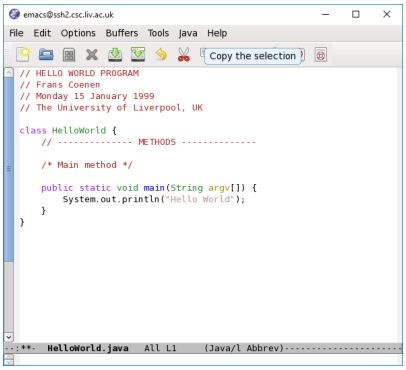
Linux offers a plethora of text editors. We will just mention three here: The default text editor gedit, Atom, and Emacs. From the command line you can start these editors using the commands `gedit`, `atom` and `emacs`, respectively.



(a) gedit



(b) Atom



(c) Emacs

Figure 3: Linux text editors

4.1 gedit

In the previous practical you have created a file `HelloWorld.java` containing a Java program. Change to the directory in which you have stored this file. Make sure that you are in the right directory by using the `ls` command. Now open the file in gedit by using the command

```
▶ gedit HelloWorld.java &
```

The ampersand `&` at the end of the line tells the shell to start the command 'in the background' so that you can continue to use the shell and do not have to wait until the command has finished. This makes sense since you normally intend to use the text editor for a long time and might want to do other things in parallel, such as compiling and running the program that you edit. It will take about 3 seconds for editor to open as it is running remotely on a Linux PC.

Initially, gedit, with `HelloWorld.java` open in it, probably looks as shown in Figure 3a. gedit uses *syntax highlighting* to indicate the structure of your code (as you saw with Notepad++ or Atom in Practical 1). gedit uses tabs to manage the files you have open. Right now there is one tab for the one file `HelloWorld.java`. If you were to open another file, additional tabs would appear and allow you to easily switch from one file to the next. For program development it is often helpful to have the lines of the code numbered, as error messages by a compiler often indicate the line number at which an error has been found. To enable line numbers in gedit, click on the so-called hamburger menu, then on the entry "`Preferences`" in that menu. In the "`View`" tab, click on the option "`Display Line Numbers`". Also explore other options available in the preferences:

- In the "`View`" tab you could choose to enable the option that matching brackets should be highlighted;

- In the "`Editor`" tab you could enable the option that a backup copy of files are created before saving a file;
- In the "`Font & Colours`" tab you could change the colour scheme that is used by gedit.

If you make changes to a file, then you can save those changes by clicking on the "`Save`" button or via the key combination `<CTRL>-s`. You can have more than one file open in gedit at the same time. You open files from within gedit via the "`Open`" menu. The menu allows you to either search for a file or to navigate to it via "`Other Documents...`" which will open a file browser. For each open file, there will be a *tab* entitled with the name of the file and by clicking on a tab you switch between open files. A *tab* containing a file with unsaved changes will have a star `*` to the left of the file name in the title.

Once you feel that you understand how gedit works, close it by clicking on the cross in the top right corner, via the entry "`Quit`" in the hamburger menu, or via the key combination `<CTRL>-q`.

4.2 Atom

Next try Atom. Open `HelloWorld.java` in Atom using the command

```
▶ atom HelloWorld.java &
```

Here the ampersand `&` is optional, Atom would start in the background even without it. Just like gedit, Atom uses tabs to manage several files at the same time (Figure 3b). When you open Atom for the first time it is likely there are several tabs open, possibly including "`Welcome`", "`Telemetry Consent`", and a tab for the file `HelloWorld.java` you have opened. You might want to start with "`Telemetry Consent`" and decide whether you want to send usage stats to the developers of Atom.

Just as the other editors, Atom uses *syntax highlighting* to indicate the structure of your code. Atom should already show line numbers next to your code. Likewise, it should highlight matching brackets: place the cursor on any curly bracket in the code; the bracket should then be underlined and so is the opening or closing bracket matching it.

If you make changes to a file, then you can save those changes using "`File→Save`" or via the key combination `<CTRL>-s`. A *tab* containing a file with unsaved changes will have a blue dot on the right-hand side of the title bar of the tab.

It is still worth exploring the configuration options of Atom.

- Use "`Edit→Preferences`". A new tab "`Settings`" will appear in the main window pane of Atom. In the left pane click on "`Editor`", then in the right pane scroll down to "`Show Line Numbers`". Make sure that this option is enabled.
- Next click on "`Themes`" in the left pane and choose the colour scheme that you prefer. For the "`UI Theme`" choose "`One Light`" among the options and for the "`Syntax Theme`" again choose "`One Light`" among the options.
- Finally, explore "`Install`". Atom is an extensible editor for which a lot of packages are available that make program development easier. In the search field enter "`Java`" and press `<RETURN>`. Atom will list a number of packages for Java and JavaScript. Install the package `autocomplete-java`.

Once you feel that you understand how Atom works, close it by clicking on the cross in the top right corner, via "`File→Quit`", or via the key combination `<CTRL>-q`.

4.3 Emacs

The third editor we want to explore is Emacs. Open `HelloWorld.java` in Emacs using the command


```
▶ emacs HelloWorld.java &
```

Emacs has quite a distinct look and feel from the other editors, some of the differences are due to the fact that it can be used as a *command line editor* without a graphical user interface (using the option `-nw`) and controlled solely via the keyboard.

Open files, as well as other things that Emacs is capable of, are held in *buffers*. *Buffers* are shown in *windows* which in turn are shown in *frames*. *Frames* correspond to GUI windows and contain one or more *window*. When you first open Emacs you probably see one *frame*, containing two *windows* arranged vertically within the *frame*. Each *window* shows a different *buffers*, one for the file `HelloWorld.java` and one called the *startup screen*. The later has 'links' to useful information such as a tutorial and a manual. You should explore those later, for the moment close the *window* with the *startup screen* by clicking on "`Dismiss this startup screen`".

This leaves the *window* showing the *buffer* with the file `HelloWorld.java`. Just as the other editors, Emacs uses *syntax highlighting* to indicate the structure of your code. At the bottom of each window is a *modeline*. Among other things it indicates the name of the buffer / file and via `L` followed by a number shows you in which line of the buffer / file the cursor is currently placed. Move the cursor around a bit so that you can see how that number changes. By default, matching brackets are not indicated. To change that, click on "`Options`" then click on the box next to "`Highlight Matching Parentheses`", and finally on "`Save Options`". Once you have done that, check whether matching brackets are not indicated: Place the cursor on an opening curly bracket or parenthesis, then the matching closing bracket / parenthesis should be highlighted. For the

reverse you have to place the cursor just after a closing bracket / parenthesis, instead of on it. Give it a try.

If we want to have permanent line numbers to the right of our program code, then we have to directly edit the configuration file of Emacs. Click on the "`File`" menu in the toolbar, then on "`Open File...`". This will open a file browser. Use the file browser to locate the file `.emacs` in your home directory (note the dot at the start of the file name). Once you have selected the file in the file browser click on "`Open`" and it will appear in a new buffer and window in Emacs. It should look as follows:

```
(custom-set-variables
;; custom-set-variables was added by Custom.
;; If you edit it by hand, you could mess it up, so be careful.
;; Your init file should contain only one such instance.
;; If there is more than one, they won't work right.
'(show-paren-mode t))
(custom-set-faces
;; custom-set-faces was added by Custom.
;; If you edit it by hand, you could mess it up, so be careful.
;; Your init file should contain only one such instance.
;; If there is more than one, they won't work right.
)
```


Most of Emacs is written in the functional programming language Emacs Lisp. What you see above are expressions in that language. Add the following two expressions at the end of the buffer:


```
(global-linum-mode t)
(column-number-mode t)
```


The first enables line numbers in all buffers, the second column numbers. As you type in the expressions, note that `**` has appeared to the left of the file name in the modeline: This indicates that the file has been changed and needs to be saved. Do so, using "`File`→`Save`". Without further action, the changes you have made would only take effect the next time you start Emacs. In order to force an immediate effect, we need to evaluate the expression (basically, we execute the Lisp program). To do so, click on "`Emacs-Lisp`" in the toolbar and then on "`Evaluate Buffer`". Line number should appear and in the modeline a pair of number consisting of a line number and a column number has replaced the single line number.

Switch back to the buffer containing your program code. You do so via the "`Buffers`" menu in the toolbar where you select the entry for `HelloWorld.java`. You can see that in this buffer too you now have line numbers and column numbers. Note that the toolbar now has an entry "`Java`". Explore this menu.

Once you feel that you understand how Emacs works, close it by clicking on the cross in the top right corner, via "`File`→`Quit`", or via the key combination `<CTRL>-x <CTRL>-c`.

5 Compiling Java programs

In the last practical you created a Java program called `HelloWorld.java`. This file contains the *source code* which is readable text, designed to be understood by the programmer. In order for the computer to be able to run the program, this code must first be *compiled* into Java Byte Code.

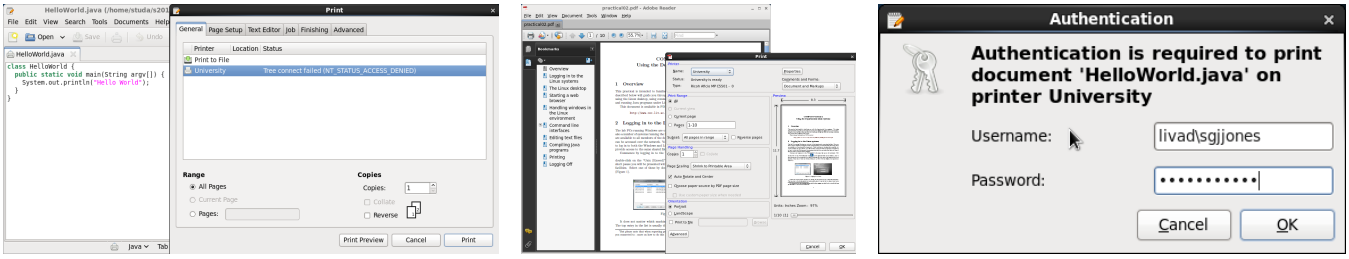


Figure 4: Printing in gedit and Adobe Reader

This binary "class" can then be executed by the computer in order to actually run the program. Both of these steps involve typing commands into the (Linux or Windows) terminal window.

Start a terminal window under Linux, change to the directory in which you have stored your Java program, and ensure you can see the file `HelloWorld.java` in the output of `ls`. If it's not there, use the file manager to locate this file, and then use `cd` in the terminal window to move to the same folder. Ask a demonstrator for help if you are not sure about this.

Then run the command

```
▶ javac HelloWorld.java
```

If this compiles successfully, then the command will complete without displaying any further output, and you should now have a file `HelloWorld.class` in the same folder. If you see errors produced by the compiler, or the class file is not present, then open the `HelloWorld.java` file in your preferred text editor, and check that the contents still match the code from last week's practical. Remember that on Linux, case matters—if the file is called `HelloWorld.java`, and you type the command `javac helloworld.java`, then the compiler will not be able to find it. The Windows shell ignores case differences, so this command would be successful under Windows (that's a bug, not a feature).

Again, ask a demonstrator for help if you get stuck.

Assuming that your code file has compiled correctly, you should now run the program, using the Java interpreter command:

```
▶ java HelloWorld
```

(*without* the `.class` suffix). This should produce the expected output string ("Hello, World!").

Again, be careful about case—you should type the name of the class file *exactly* as it appears in the file manager (or `ls`). Note that the class filename will always be the same as the name that you have in the statement

```
class HelloWorld {
```

at the start of the code in the Java file. This will normally match the name of the Java file, but it does not have to. If they are different, the class file will follow the code, rather than the filename.

6 Printing

If you want to print your Java program, in gedit you can do so as follows: Click on the hamburger menu in the toolbar, then on the printer symbol at top of the drop-down menu that opens. This will open the print dialogue that lets you choose a printer and set your print preferences (Figure 4). The two main printers available to you are '`Print to File`' and '`University`'. '`Print to File`'

converts your file to PDF or PostScript, `University` refers to the "Follow-You" printing service of the University just as we have seen under MS Windows in the previous practical. Again, make sure before printing that you choose the correct printer setting, e.g. single-sided or double-sided. Once you press the "Print" button, another dialogue window will open and ask for authentication. Enter your University username, in the form `livad\`⟨*userId*⟩, and University password; see the right-most image in Figure 4.

Most Linux applications have print dialogues identical or similar to the one that gedit uses.

7 Logging Out

You end the SSH session with one of the commands `exit` or `logout`. You will then see a message in the main window pane telling you that the session has been stopped (Figure 2f). If you do not see this message, then there are still commands running in the background and the connection has not properly been terminated. Make sure that any applications with a graphical user interface, for example, text editors, have been closed, then click on the cross symbol in the top right corner of the tab above the main window pane. Typically, you will be shown a prompt informing you that one or more processes are still running and asking you whether you are sure that you want to close the tab, and thereby close the connection. If you confirm, the tab will be closed and the connection is properly terminated.

Note that the pane to the left of the main window pane changes. It now shows a list of previous sessions that you have established, in particular, you see the names the computers that you connected to and the user name you have used. You can reconnect to a particular computer by double-clicking on the corresponding entry in the list. If you have authorised MobaXterm to save the password that you have used, then the connection is reestablished without prompting you for a password. Give it a try, check that a SSH connection is indeed established, then log out again.