Introduction to Labs (Lab Intro 2)
Using the Departmental Linux Systems

1 Introduction

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 7.7), which are available to all members of the department. The assignments on COMP284 will ask you to use *scripting languages* to produce *web-based applications*. The web server hosting those applications and providing them to the public is itself a Linux system. You are not able to directly work on the web server, but there are other Linux systems that provide an easy way to set up web-based applications on the web server via their shared filestore. All these systems together can be considered to be the *production servers* for your applications, that is, they host the applications and make them accessible to the public, typically after extensive development and testing. Normally, such development would be conducted on a separate set of systems, the *development servers*. To simplify matters, we will not make that distinction. We assume that all development work is conducted on the production servers.

This practical is intended to familiarise you with the departmental Linux systems. The tasks described below will guide you through the process of accessing and logging onto a Linux system, using the command line interface and editing text files.

This document is available in PDF format at

`https://cgi.csc.liv.ac.uk/~ullrich/COMP284/notes/labintro02.pdf`

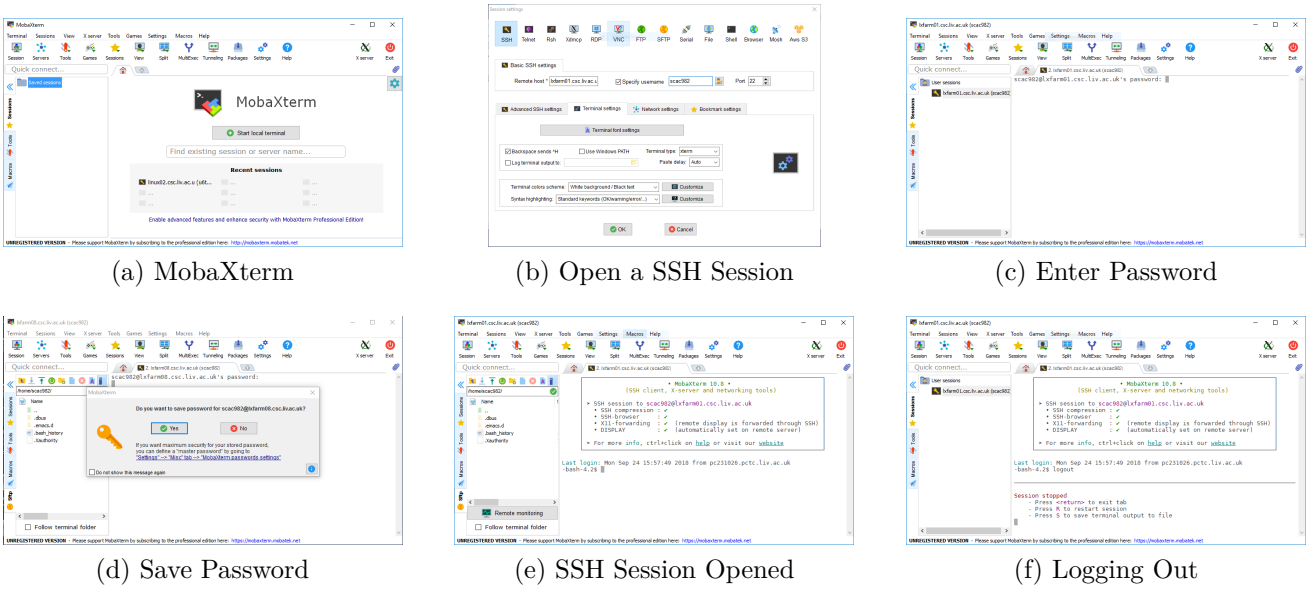
While you work through the tasks below compare your results with those of your fellow students and ask one of the demonstrators for help and comments if required.

2 Logging in to the Linux systems

The Departmental Linux systems are not physically accessible, but can be accessed over the network. You use the same University (MWS) username and password to log in to both the Windows and Linux systems, but the personal filestore on the Linux systems, called your '*home directory*', is separate from the `M:` drive on the Windows systems.

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 servers. In the text field to the right of the label "`Remote Host`" enter the name of a Linux server, `lxfarm01.csc.liv.ac.uk` to `lxfarm08.csc.liv.ac.uk`, click on the button to the left of "`Specify username`", then enter your University 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



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

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

Figure 2: Using MobaXterm for SSH sessions

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 University (MWS) account. Note that there will be no visiual indication that you are typing anything. Press RETURN once you have typed in all characters of your password. You will then be asked whether you want MobaXterm to store your password so that next time you connect to this particular Linux server 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 server.

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 text you have entered, and this can be used to correct typing errors.

The *shell* is a command language interpreter that executes commands read from the standard input device, in this case your keyboard, or from a file. So, anything you type is intended as command to the operating system. When you have finished typing a command, press the <RETURN> key. This tells the shell to run the command you specified, relative to your current *working directory*, and to show any output produced by the command. It will then display another prompt, ready for your next command.

Here is an example that uses the `hostname` command to discover which Linux server you are using:

```
▶ hostname
```

lxfarm06.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 command 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').

Linux	MS Windows	
`cd`	`M:`	Change your *working directory* to your *home directory*
`pwd`	`pwd`	Show the name of the *working directory*
`ls -l`	`dir`	Long listing of the files and folders in this directory
`mkdir COMP284`	`mkdir COMP284`	Create a new folder (directory)
`cd COMP284`	`cd COMP284`	Change the working directory to the named sub-folder
`ls`	`dir /w`	Short listing of files and folders in this directory
`cd ..`	`cd ..`	Change the working directory up a level (.. = "parent directory")
`cp /etc/php.ini t1`	`copy \etc\php.ini t1`	Make a copy of a file
`cat t1`	`type t1`	View the contents of the file `t1`

Linux	MS Windows	
`more t1`	`more t1`	View the contents one page at a time (press the 'q' key to quit)
`mv t1 php.ini`	`rename t1 php.ini`	Rename a file
`cp php.ini COMP284`	`copy php.ini COMP284`	Make a copy of a file in a different folder
`mv php.ini COMP284/t2`	`move php.ini COMP284\t2`	Move a file into another folder (and rename it)
`ls COMP284`	`dir COMP284`	List the contents of a folder
`rmdir COMP284`	`rmdir COMP284`	Try to delete a (non-empty) folder (which fails)
`mv COMP284/* .`	`move COMP284\* .`	Empty the folder (by moving the contents to the current directory)
`rmdir COMP284`	`rmdir COMP284`	Delete the (empty) folder
`rm php.ini t2`	`del php.ini t2`	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 `COMP284` is different to one called `comp284`. 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 the Windows File Manager will often hide the *filename extension*, that is, 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.

Using the command

- `mkdir ~/COMP284; cd ~/COMP284`
- `touch test1 test2 test3.txt test10`

create a directory and four files in it. Next, try the following sequence of commands. Think about the results you see, and what these patterns might mean.

- `ls test?`
- `ls *`
- `ls *.*`
- `ls *.txt`

```
▶ ls test*
▶ ls *st?
```

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 test* > out1.txt
▶ cat out1.txt
```

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

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

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" logFile
```

(where *command* is the command whose output you want to capture and *logFile* is the name of a file in which you want to store that output). Try experimenting with this, using some of the commands above, for example, try

```
▶ script -c "ls -l /etc/" out3.txt
```

(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 out3.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*.

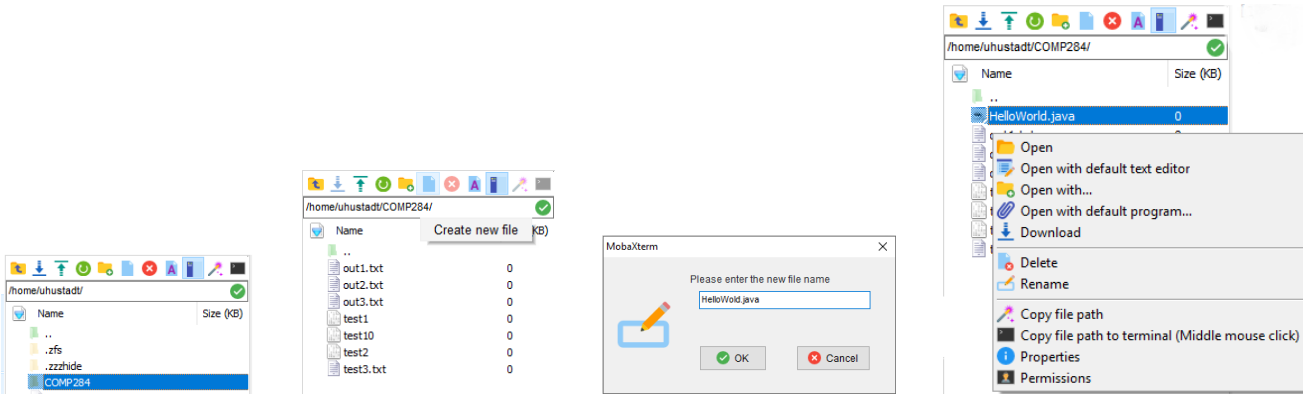
Quite often you will have several different files in the same folder, with significantly different names. The Linux shell allows you to type the first few characters of a filename (sufficient to uniquely identify that file), and then hit the `<TAB>` key. This will automatically complete the name of the 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

In the course of these practicals we will create many files, mainly, HTML documents, PHP scripts and JavaScript scripts. The main complication in doing so is that for these files to be of any use to us they will need to be on the Linux systems in a separate filestore from the Windows PC that you are currently interacting with. There are basically, three approaches to solving this problem:

1. You first create and edit a file on the Windows PC in your MWS filestore and then transfer the file to the Linux filestore. We will cover file transfer in the next worksheet.
2. You create and edit a file on the Linux PC in the Linux filestore using a Linux text editor whose graphical user interface is shown on the Windows PC. Linux supports a plethora of text editors. In Sections 4.2 to 4.4 we will just cover three of them: The default text editor gedit, Atom, and Emacs.
3. You create a file in the Linux filestore using MobaXterm's file browser and edit it using MobaXterm's built-in editor (or you start an alternative editor from MobaXterm's file browser). Whenever you save changes to the file, MobaXterm will automatically update the file in the Linux filestore. We cover this approach in Section 4.1



(a) Step 1 (b) Step 2 (c) Step 3 (d) Step 4

Figure 3: Creating/editing a file with MobaXterm

4.1 MobaXterm's built-in editor

Let us first explore how we can use MobaXterm's functionality to create and edit a file, in our case, a Java program. In the file browser pane of MobaXterm you should see the contents of your home directory in the Linux filestore which should include a directory COMP284 (Figure 3a). Double-click on that directory and MobaXterm will now should you the contents of COMP284 which should consist of various files we have created in previous exercise. You can now create an empty new file by clicking on the "`New File`" icon in the toolbar of the file brower pane (Figure 3b). Alternatively, you could right-click on an empty space in the file browser pane and select "`New empty file`" in the pop-up menu that opens. Either of these actions will open a dialogue window that asks you to enter the name of the new file, enter '`HelloWorld.java`' (Figure 3c). The file browser pane will then show the new file. Right-click on the file and in the file context menu select the entry "`Open with default editor`" (Figure 3d). This opens MobaXterm's built-in editor. Open the URL

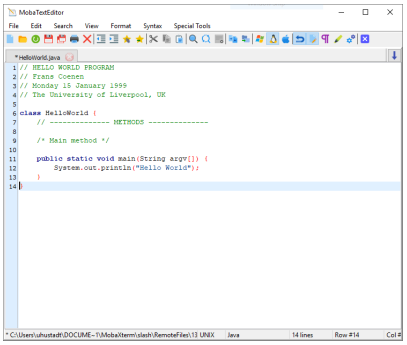


Figure 4: MobaXterm Editor

`http://cgi.csc.liv.ac.uk/~ullrich/COMP284/examples/HelloWorld.java`

in a web browser. Copy the Java program that the web browser will show you into the editor (Figure 4). This allows you to get a better impression of the functionality that the editor provides.

The editor 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.

The editor supports *syntax highlighting* to indicate the structure of the code and it indicates line numbers on the left. If you hover with over an opening parenthesis or bracket, the editor will highlight the corresponding closing parenthesis or bracket, and vice versa.

Explore the toolbar and menus of the editor. For instance, note that the editor allows you to select two 'formats', "`DOS format`" (Windows icon in the toolbar) and "`Linux format`" (Linux penguin icon in the toolbar). DOS/Windows and Unix/Linux use different end-of-line character sequences and these two entries in the toolbar allow you select the right one. For our purposes that will always be 'Linux format'.

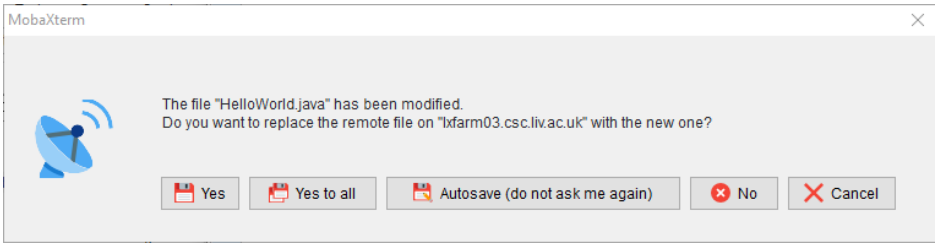


Figure 5: Saving a remote file

Make some change to the program. Notice that in the editor tab there is now a star * to the left of the name of the file, indicating that there are unsaved changes. Save the file by clicking on the floppy disk icon in the toolbar. This opens in a new dialogue window (Figure 5). MobaXterm indicates that the file will not be saved locally, but remotely in the Linux filestore on the Linux server that you are connected to. You can either permit this once, by selecting "`Yes`", or for all future saves in this editing session, by selecting "`Autosave`". Choose either of the two options then close the editor. It is worth pointing out that the file context menu (Figure 3d), via the entry "`Open with...`", also allows you to use other editors or IDEs, for example, Atom or Notepad++. The remote saving functionality of MobaXterm will still work with those editors.

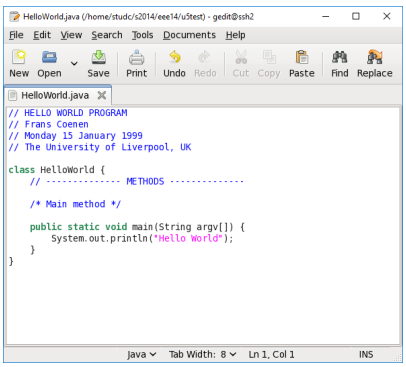
4.2 gedit

Next we consider the use of a Linux editor that runs on a Linux server and edits files in the Linux filestore but with its graphical user interface being displayed on your Windows PC.

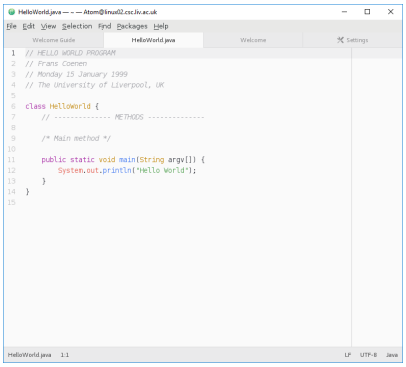
In the MobaXterm terminal window, change to the directory in which you have stored the program `HelloWorld.java`. Make sure that you are in the right directory by using the `ls` command. Now open the file in gedit by using the command

```
▶ gedit HelloWorld.java 2> /dev/null &
```

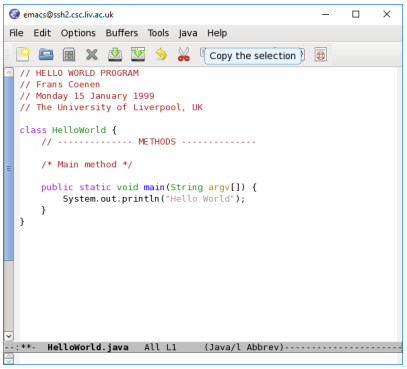
The redirection `2> /dev/null` disposes of error messages. The ampersand `&` at the end of the line tells the shell to start the command 'in the background' so that you can continue to use the shell and do not have to wait until the command has finished. This makes sense since you normally intend to 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 the editor



(a) gedit



(b) Atom



(c) Emacs

Figure 6: Linux text editors

to open as it is running remotely on a Linux server.

Initially, gedit, with `HelloWorld.java` open in it, probably looks as shown in Figure 6a. gedit uses *syntax highlighting* to indicate the structure of your code.

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`.

4.3 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 6b). When you open Atom for the first time it is likely there are several tabs open, possibly including "`Welcome`", "`Welcome Guide`", "`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. Then switch to the tab with the `HelloWorld.java` file.

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.

4.4 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 *buffer*, 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 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. Use "File→Open File..." to 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; also, to be able to see the file .emacs in the file browser, you need to enable the option 'Show hidden files' in it.). 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" or via the key combination <CTRL>-x <CTRL>-c. 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 expressions in this buffer (basically, we execute the Lisp program). To do so, click on "Emacs-Lisp" in the toolbar and then on "Evaluate Buffer". Line numbers 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.

As mentioned, Emacs can be used as a *command line editor*. In the MobaXterm terminal window use

```
▶ emacs -nw HelloWorld.java
```

to open Emacs without a graphical user interface. Instead of using menus to load and save files and switch between buffers, you now have to use key combinations to control the editor. The most important such key combination are:

- <CTRL>-x <CTRL>-f: load file into buffer
- <CTRL>-x <CTRL>-s: save buffer to file
- <CTRL>-x <CTRL>-c: exit Emacs
- <CTRL>-k: kill line (basically, a 'cut' operation)
- <CTRL>-y: yank back line (basically, a 'paste' operation)

For a much more comprehensive list see

https://www.gnu.org/software/emacs/refcards/pdf/refcard.pdf

5 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 application with a graphical user interface, for example, text editor, has 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.