Appendix A: Quick UNIX reference

UNIX is a common name for a family of operating systems, most popular of which in this epoch is Linux. This document is by no means a UNIX tutorial. Sophisticated UNIX users may find some of the material here grossly simplified. Only a small subset of the shell commands available is given and very few options are mentioned.

A.1 Files

A file is a collection of data with a unique name. Names are case-sensitive (lowercase and uppercase letter are considered distinct). All permanent storage on UNIX consists of files. In this course you will encounter several different types of files that are recognized by their *extensions*. The extension is the ending of the file name following the (last) dot. Some extensions are expected or enforced by certain programs, while others are merely conventions.

Some file types, most relevant to this course, are:

- *Text files.* They are human-readable and can be edited by means of a text editor. There are many different kinds of text files:

  Program source files

  These files are source code written by programmers in a high-level language like FORTRAN or C. The compilers for these languages require the source file names to end in `.f` or `.c` respectively.

  Data files

  Input data files contain data that will be fed into a program. Output data files contain output produced by a program that you want to keep in a file.

  LaTeX files

  These files are "source" files for documents such as the one you are reading. LaTeX software requires that the file name end in `.tex`.

  Webpages

  are files with extensions **htm**, **html**. They stand for **H**yper**t**ext **M**arkup **L**anguage.

© MMIX Department of Mathematics and Statistics, Memorial University of Newfoundland
September 4, 2009

- *Executable files*

  These files are programs that can be run directly; that is you can type their names into the shell like the **UNIX** commands. UNIX shell commands, compilers, text editors, Internet browsers belong to this category. Also this type of file is produced by a *compiler* from a *source* file written in FORTRAN, C, or another language. The FORTRAN and C compilers name this file `a.out` by default.

- **dvi** *files*

  These files are created by LaTeX to be printed on various devices, such as the screen or a laser printer. These files' names will end in `.dvi`. This stands for **de**vice **i**ndependent.

- **pdf** *files*

  The extension pdf stands for **P**ortable **D**ata **F**ormat, developed by Adobe, Inc. Files in this format are produced by many word and data processors, including LaTeX. *Acrobat Reader* is the most popular viewer for these files.

- **ps** *and* **eps** *files*

  The extension `ps` denotes **P**ost**s**cript, another page description format developed by Adobe, Inc. In fact, Postscript is pretty much a programming language. Postscript graphics files can even be created by hand, but usually their creation is helped by a software. Encapsulated postscript (eps) format is almost identical. The only difference is that an `eps` file must contain the Bounding Box information (see Sect. **??**). This tiny difference is important when you have to incorporate a Postscript graphics into your LaTeX document.

- **mw** *and* **mws**

  These are the extensions assigned to Maple worksheets.

Depending on context, the term *file name* can refer either to the full name or to the part preceding the dot after which the extension follows. So, we would often say "the LaTeX file `project3`"; here the full file name `project3.tex` is implicit.

Each file must have a unique name which distinguishes it from all other files. If a file is named sensibly, the name will give a clue as to the contents. One important point is to distinguish between various input and output data files, as these can become quite numerous.

A.2 Directories

A group of files is stored in a *directory*. A directory can in turn contain other directories. The UNIX *file system* contains all files on the computer, and as such is made up of a hierarchy of directories. It can be thought of as an upside-down "tree" of directories (Fig. **??**).

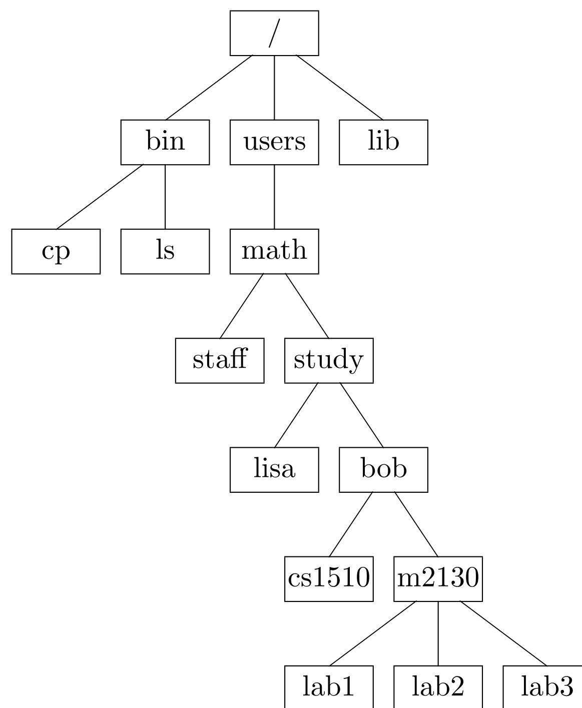

Figure A.1: Sample directory tree

The purpose of directories is to help users organize their files. Each user can create any number of subdirectories within their home directory, to any depth in the tree. If you look at the depicted directory structure, you can see that the directory `bob` contains two first-level subdirectories named `cs1510` and `m2130` and the latter contains three second-level subdirectories named `lab1`, `lab2` and `lab3`.

A.3 Pathnames

There can be files with the same name in different directories. For example, there are likely dozens of files by the name `lab1.tex` on the system, created by different students. Yet there is a method to identify any file in the file system unambiguously. The way do this is to use a *pathname*. This name specifies the path you must travel through the tree to reach the file. The full name of a file, as understood by the system, is the pathname from the root directory. A full name begins with an initial slash (`/`), with further slashes separating each directory name. The actual name of the file, by which it is known to its owner, is the part of the pathname after the final slash. For example, the full pathname of the directory `bob` in the above example is `/users/math/study/bob`.

Note that UNIX is different from Microsoft Windows, where the backslash (\) is the separator for pathnames and the top-level directory is the drive name (for example, `C:\`).

Along with **absolute pathnames** just described, there are **relative pathnames**. They describe how you reach the required file or directory from the current directory. Sometimes you may have to go up the reversed tree (towards the root). The directory one level higher the current directory is denoted `../` (double dot). Thus, the relative pathname from Bob's directory `lab1` to his directory `cs1510` is `../../cs1510`. In the same situation, the pathname `../lab1` points at the directory we are in. The standard way to refer to a directory from itself is by the relative pathname `./` (the "dot directory", means — stay here). Obviously, in practice, if a file is in the current directory, you just call it by its name.

A.4 Shell

While a graphical user interface is available for most modern programs, it doesn't hurt to be familiar with an old-fashioned but always reliable command-line mode.

To enter the command-line mode, you need to open a terminal shell window (click on the icon that looks like a display). Now, what you type is interpreted by a special program called the *shell*. It launches and runs other programs for you. The shell tells you it is waiting for input by displaying a prompt (like `lumsden $`) at the beginning of a line.

Two combinations of keys deserve a special mention:

- Almost any program started from the command line can be stopped by typing **Ctrl C**.

- The combination **Esc K** returns the most recent command entered in the shell. Pressing **Esc K** twice will quickly bring back the second most recent command, etc. Knowing this is especially convenient for those who run compilers in the command line. In another type of shell, the single key ↑ scrolls up the commands history.

A command followed by the ampersand character (&) launches a program in a mode detached from shell's interactive session, so that you can enter other commands in the same window while the program is running. For example, you can open a text editor and modify your program in it, while the command line will be available for compilation of the program:

`lumsden $` *kile myprog.f &*

A.5 Basic UNIX commands

A UNIX command consists of one or more words separated by spaces. The first word is the name of the program you want to run, either a standard system program or one you created yourself. The rest of the words are *arguments* to the program, which usually are either the names of files, or *options*, which tell the program to modify its usual behaviour. Options usually begin with a dash (`-`), for instance `-l`.

Notation used. *This typeface* is anything you type in literally. Anything in *italics* is not typed in literally. For example, *file* means that you should not type the word "file", but that you should type a file name instead. Anything followed by the ellipses (. . .) can be repeated. For example, *file . . .* means you can type in several file names. In the next section examples, `this typeface` will denote what the computer displays, as opposed to what the user types.

Commands

ls

list the names of all files in the current directory.

cp original-name new-name

create a copy of the original file with a different name.

mv original-name new-name

rename the file with a different name, or move the file to another place.

rm file-name . . .

remove the file(s).

cat file-name . . .

display the contents of the file(s).

p file-name . . . or *more file-name . . .*

display the contents of the file(s), one screen at a time.

cd directory-name ...

changes to the directory named.

mkdir directory-name

create a new directory.

rmdir directory-name

remove a directory.

pwd

display the current directory.

man subject

display the UNIX manual for the subject, can be used to find out details about a command's syntax.

lpq and *lprm request-ID*

lpq can be used to obtain a listing of the jobs in the printer queue waiting to be executed. If by misfortune you have asked to print a document and suddenly wish to cancel the print job, use *lpq* to obtain the request-ID number associated with your printing job and remove it from the printer queue with the command *lprm request-ID*.

A.6 Working with directories and files

You change your current directory with the *cd* command. *cd* takes one argument, the pathname of the directory you want to change to. For example, the command

`lumsden $` *cd /users/math/study*

makes `/users/math/study` the current directory. In this case, an absolute pathname was used, which isn't a common practice for ordinary users.

Much more practically useful are relative pathnames. You must, of course, know what your current directory is. If not sure, use the *pwd* command. For instance, if the current directory is `/users/math/study/bob/m2130/lab1`, then typing *pwd* you get system's response:

```
Current directory is /users/math/study/bob/m2130/lab1
```

Now, typing *cd ..* will make `/users/math/study/bob/m2130` the current directory. Note that if the user Bob types *cd /users/math/study/lisa*, the system will reject this command as Bob is not authorized to access Lisa's home directory.

The special variant

cd ␣˜

makes the user's home directory the current directory. Thus, if Lisa types *cd ˜*, the current directory becomes `/users/math/study/lisa`, no matter what it was before.

The command *ls* will list the names of all the files in the current directory. If you give it a directory name, it will list the names of all the files in that directory. If you give it a file name, it will list that name, which can be useful if you just want to check if such a file exists.

For our sample directory tree, if the current directory was `/users/math/study` and you ran *ls*, it would display

```
lumsden $ ls
    bob   lisa
```

The command *ls ␣ -l ␣ name* will give you a detailed information about a file or files.

The command *ls ␣ -a* displays, unlike the bare *ls*, those files and directories whose names begin with a dot, like the name `.www`, which often denotes the directory in user's account accessible from the Internet.

Wildcards can be used to specify a pattern that we want a file name or a directory name to match. The most useful wildcard character is the asterisk `*`, which matches any, even empty, combination of characters. The command

*ls ␣ *tex*

will display all files with names ending in `tex`. The command

*ls ␣ *tex**

will, in addition, display all files with names containing `tex` at the beginning or in the middle, like `text1.doc`, `latexnotes`, etc.

The command

*cd ␣ ../*2*

issued from Bob's directory `m2130/lab1` will change directory to Bob's `math2130/lab2`, because only one directory name matches the pattern `../`*2.

The *cp* command creates a new copy of a file. For example, if the current directory contains one file, `lab1.tex`, and you ran *cp lab1.tex newlab.tex*, then *ls* would display

```
lumsden $ cp ␣ lab1.tex ␣ newlab.tex
lumsden $ ls
    lab1.tex   newlab.tex
```

and the contents of `newlab.tex` would be identical to `lab1.tex`. If a file named `newlab.tex` had already existed, that file would have been overwritten.

The command *cp* with wildcards in its arguments is very convenient when you copy your files from the working directory to the submission directory (see Sect. 5.1). The command *cp ␣ * ␣ m2130-a1/* will copy all files from the current directory to its subdirectory named `m2130-a1`. This may not be exactly the desired outcome: for instance, `.log`, `.aux`, `.dvi`, and `a.out` files need not be included. So a better command is

*ls ␣ *tex ␣ *eps ␣ *.f*

which will copy all (possibly, single) LaTeX source file(s), any encapsulated Postscript figures, and any Fortran programs.

The *mv* command is similar to the copy command. However the original name of the file will be lost after the command is finished. You can also use *mv* to move a file from one directory to another. The *mv* command has similar behaviour to *cp* if its last argument is a directory. One difference between *mv* and *cp* is that *cp* will not copy a directory, whereas *mv* can rename it or move it into another directory.

The command to remove a file is *rm*. The arguments to *rm* are the files you want to remove. Be careful with this command. Once you remove a file, it is difficult to get it back. All the files on the system are saved every night, so if you do accidentally remove a very important file, the system administrator may be able to help you. Be especially careful with *rm*␣* command!

The command to create a directory is *mkdir*. It takes one argument, the pathname of the directory you wish to create. For instance, if Bob ran *mkdir lab4* in his `m2130` directory, he would see

```
lumsden $ mkdir lab4
lumsden $ ls
    lab1   lab2   lab3   lab4
```

The command *rmdir* takes one argument, the pathname of a directory you wish to remove. If the directory is not empty, *rmdir* will not function.

The command *p* is actually a shorter name for the command *less*. This is a program you can use to quickly look at text files. *p* will display a screenful of the file and then stop with a ":" prompt at the bottom of the screen. You can hit the space bar to see the next page, or the `u` key to go upwards in half-screenfuls. When you reach the end of the file, `p` will prompt with the name of the file followed by `(END)`. Type the `q` key to quit from `p`.

A.7 Redirection of output

Most students have little trouble making their programs to print on the screen, but they experience more difficulties in creating programs that would flush their output to a file. Not that creating such a program in any language is very difficult, but it certainly requires more effort and time. Redirection is a trick that can help to work around this problem.

Every program running under UNIX has a "file" already opened: *standard output*. Normally, it is your shell terminal's window. You can *redirect* it to a text file in your directory. For instance, if you want your program `a.out` to print the output to a file `data-out` instead of the screen, you can run it like this:

`lumsden $` *a.out > data-out*

Note that this could be a problem: if you have messages printed out prompting for certain kinds of input, those messages will also be redirected into the file instead of the screen. You should in this case write your programs so that they not be prompting for input. Or at least know how many numbers and in what sequence the program wants you to enter; you can then feed the data into your program blindfold.

Note also that if you run the same command again, the old contents of the file `data-out` will be lost.

A.8 Access privileges

We mentioned previously that the system will deny Bob to access Lisa's directory and vise versa. Technically, the access is governed by certain attributes assigned to each directory and file on the system. Each file has its *owner*, who belongs to a *group*. The owner can open or close access to his/her files independently for *self*, *group*, *and others* (or for *all* at once), and for three purposes: *to read*, *to write*, and *to execute*. The command is *chmode*. Consider a few examples.

The following prohibits writing to file `project1.tex` to all, even to its owner. (A possible purpose is to protect the important work after it is finished from accidental deletion.)

```
lumsden $ chmode a -w project1.tex
```

To prohibit group members and others (i.e. everbody but the file owner) from any kind of access to the file `project1.tex`:

```
lumsden $ chmode go -wrx project1.tex
```

To grant everybody a permission to read all existing `.pdf` files in current directory:

```
lumsden $ chmode a +r *.pdf
```

Programs that you routinely work with (editors, compilers, etc.) properly set the access privileges to the files that they create or modify. If, by any chance, a program says "Access denied", you may need to explore the status of the file concerned. A possible reason is that the file is currently being used by another program, which temporarily closed an access to it.