University of Portsmouth

Corpus Linguistics in the South
November 10th 2012

Unix for corpus users: a beginner's guide

1 Introduction

1.1 What is Unix?

Unix is a generic term for a family of operating systems (OSs) (OS = the underlying software in a computer that manages its hardware and memory resources), originally developed by Bell Labs in the 1960s. Until the late 80s / early 90s Unix was routinely used within universities and many other large organizations, being designed with multiple users and networking in mind. However, the growing popularity in the PC market of Microsoft Windows and its user-friendly (?) graphical user interface (GUI) gradually encroached on the professional, commercial, and educational world, causing Unix to retreat into computer science departments and into technical domains which involve the use of large-scale computer resources. Many of the earliest corpus linguistic projects, such as COBUILD, were developed on Unix systems.

Within the personal computing market, Unix lives on in the form of the Mac OS - and also Linux, which remains the best Unix option for users of Windows PCs. For technical reasons, Linux is best described as a *Unix-like* OS rather than a version of Unix itself. Nevertheless, the full range of Unix commands and capabilities are available under it. Linux itself comes in various 'flavours' (or 'distributions') such as RedHat, openSuse, Linux Mint, and Ubuntu. The examples in today's workshop are demonstrated using the Ubuntu distribution of Linux, but should work under any version of Unix that can run a command line (ie. all of them, including Mac OS).

Although a range of graphical Windows-like interfaces are now available for all versions of Unix and Linux, the most direct and efficient way for the user to communicate with the OS is by means of a *shell* programme which runs using a *command line* in a *terminal window*. This is rather like the old MS-DOS interface that can still be invoked under Windows by typing `cmd` into the *Programs → Search* field. Shells available under Unix include the Bourne shell, the Korn shell, the C shell, and the Bash shell. They all offer a large common set of commands with some syntactic variations around the edges which need not concern us today. Today we will be using the Bash shell, which is the default shell in Ubuntu and arguably combines the best features of the other shells.

Before moving on, mention should be made of the *cygwin* package which is a free Unix emulator that runs under the Windows OS (ie. you can install it without messing with your Windows installation, partitioning your hard drive, etc). Cygwin offers a comprehensive selection of Unix shells and command line utilities, so most of the examples demonstrated today should also work under cygwin.

1.2 Why Unix (for corpus linguistics)?

Most of the functions you are used to using in your familiar corpus software (frequency listing, sorting, pattern matching...) have their equivalents in Unix utilities (and may indeed incorporate these very utilities under the bonnet). By learning to use these commands in 'standalone' mode, together with the various additional options they offer, you will be able (among other things) to create customized frequency lists, carry out more targeted searches, manipulate downloaded concordance files, present your data in different ways, and even speed up the process of corpus creation.

2 Getting started

2.1 Finding your way around

Once you are logged into Unix/Linux, you will need to open a terminal window. In some versions, a terminal window will open automatically but in Ubuntu Linux you will need to type Ctrl-Alt-t (or click the Terminal icon in the sidebar, or find it in the Applications menu). The terminal will show a *prompt*, ending in a dollar sign ($) (or % , #, or > in some shells):

(1) some_text_identifying_the_user~$

followed by a flashing cursor. The flashing cursor is where you type your commands. Let's try a simple, traditional command. Type the following at the prompt, followed by the Return key. Try to predict the result before you hit Return:

(2) some_text_identifying_the_user:~$ echo "Hello World" *<Return>*

Whenever you use the command line, you are always 'in' a particular directory (=folder), known as the *current directory* or *working directory*. Unless otherwise indicated, any file names typed in the command line will be interpreted as referring to files in the current directory. When you log in, the default current directory will be your *home directory* or *homedir*. Each user will have their own *homedir* and this can always be referred to in Unix commands by the shorthand use of the tilde (~).

To find your way round the system, type the following commands in turn at the prompt, hitting Return at the end of each one to enter the command. As before, try to predict the outcome before hitting Return.

(3) ...$ pwd

(4) ...$ ls

(5) ...$ mkdir MyNewDir ; ls
If you have a graphic interface, now look at your homedir in the file manager.
(**Tip:** The semi-colon is used to separate two different commands on the same line; they could equally well be entered on two separate lines.)

(6) ...$ cd MyNewDir ; pwd

(7) ...$ cd ~ ; pwd

(8) ...$ rmdir MyNewDir ; ls
(**Tip:** A word of warning – Unix/Linux can be very unforgiving when it comes to deleting files or directories. However, by default, it will not let you remove a directory that has any files in it.)

(9) ...$ cd .. ; pwd (.. = two full stops)

(10) ...$ cd .. ; pwd

(11) ...$ cd ~

(12) ...$ man mkdir

2.2 Command options

You will see from the manual page for the mkdir command that the Description refers to various *options* that can be used to modify the behaviour of the mkdir command (don't worry about the detailed content for now). Each option (also known as a *flag* or a *switch*) is introduced by either a single minus sign (-) or a double minus sign (--). (It is possible, indeed very common, to string different options together in the same command line.)

Assuming you have carried out command (8) above, now try the following commands one after the other:

(13) ...$ mkdir MyNewDir/MySubDir

(14) ...$ mkdir -p MyNewDir/MySubDir

(**Tip:** Rather than retype whole the command, use the Up- (and Down-)Arrow keys to scroll backwards (and forwards) through the command history, then use the Left-/Right-Arrow keys and the Delete keys to edit it.)

How has the -p option modified the behaviour of mkdir? Use the cd and ls commands to find out.

For another illustration of a command option, use the ls command with the -l option. How does this affect the output?

Tip: The command to delete a file is rm *<filename>*. As noted above, Unix can be very unforgiving when it comes to file deletion, but many users get into the habit of using the rm -i option, which provides an additional confirmation prompt before deleting the file. (This habit can be automated by the use of an *alias* - see below.)

2.3 Standard input, standard output, pipes, and redirection

Many commands allow for input typed by the user, which is then processed by the command to produce output in the terminal window. In this scenario, the input is known as *standard input* (stdin) and the output is known as *standard output* (stdout)[1].

To illustrate how this works, type the command sort at the prompt then hit Return. The cursor moves onto the next line and is waiting for input from you. Type, for example, the days of the week, one by one, hitting Return after each one. When you have finished, type Control-d to terminate standard input[2] – then interpret the output. (Again, try to predict the output before hitting Control-d.)

(**Tip:** Compare the use of Control-d with that of Control-c , which can be used to abort most Unix commands at any stage.)

However, a given command need not take its input from stdin – it can take it from the output of a previous command. To illustrate how this works, execute this command:

(15) ...$ echo -e "Monday \nTuesday \nWednesday \nThursday \nFriday \nSaturday \nSunday"

(Read the man page for echo so that you understand what the -e option and and the \n escape sequence are doing.)

Now append to the command the *pipe* symbol (|) followed by the sort command like so (remember you can use the Up-Arrow key to avoid typing out the whole command again):

(16) ...$ echo -e "Monday \nTuesday \nWednesday \nThursday \nFriday \nSaturday \nSunday" | sort

The pipe tells the sort command to take its input from the output of the echo command. We will see examples later where we use several pipes and commands one after the other to produce the desired output. This is a very powerful feature of Unix shells.

We can also do things with standard output. Execute the following commands:

(17) ...$ echo -e "Monday \nTuesday \nWednesday \nThursday \nFriday \nSaturday \nSunday" > days.txt

(18) ...$ ls

(19) ...$ cat days.txt

---

1 An on-screen error message is known as standard error - stderr.

2 The situation is a little complicated by the fact that some commands, such as grep, which we will see later, process the standard input each time Return is hit, such that there is little difference in the effect of Control-d and Control-c. Such commands can be forced to work in a similar way to sort by the use of a feature called *here-commands*, but we don't have time to go into that today.

What has happened to the output from command (17)? What is the function of the greater-than (>) character? What is the function of the cat command? Predict the output of the following command:

(20) ...$ cat days.txt | sort

Note that the following command should produce the same results as (20):

(21) ...$ sort < days.txt

The less-than character (<) tells the sort command to take its input from the contents of the file days.txt , rather than from standard input.

Now try this command:

(22) ...$ echo "lundi" >> days.txt

What is the effect of the double greater-than symbol? Use cat to find out. Add some more French days of the week to your 'days' file. Sort the whole lot by alpha order and redirect the output to a new file called days_sorted.txt .

(**Tip:** Unix doesn't much like spaces in file or directory names – use an underscore where you would otherwise think of using a space.)

3 Using Unix commands with text files

3.1 Creating a customized word frequency list

3.1.1 Command substitution

Now we are ready to try a little basic corpus work. We are going to create a corpus of Unix manual pages. First, in your homedir, create a subdirectory called ManCorpus. For each Unix command we have looked at (plus any others you happen to know), redirect its manual page to a separate file in ManCorpus. Adopt a uniform file-naming system, and make sure there are no other files in your ManCorpus folder. We now have our corpus.

(**Tip:** The files we create will be plain-text files, so give each file the suffix .txt , for ease of portability with Windows systems.)

We are going to create an ordered frequency list of the words in our corpus. Let's say, for the sake of argument, that (a) we are not interested in punctuation symbols and that (b) we do not want to distinguish between upper- and lower-case.

First, let's have a look at our input. Change to the ManCorpus folder and use ls to check the files are there. Now type:

(23) ...$ cat $(ls)

This command makes use of what is known as *command substitution*. The $(*<command>*) syntax[3] takes the output of *<command>* and uses it as the argument of another command (in this case cat). Thus, ls sends to cat a list of all the files in our corpus, which then catenates their contents and prints them to stdout. Understandably, the output is far too large to fit on the screen and whizzes straight to the end. To control this behaviour we can pipe the output through a file/output viewer called pg:

(24) ...$ cat $(ls) | pg

(**Tip:** In pg, use Return to scroll down a page and the minus sign (-) to scroll up. Use q then Return to quit pg.)

3.1.2 Using sed with regular expressions

The first stage of processing our corpus is to get rid of the punctuation symbols. To do this, we are going to make use of a powerful utility called sed (= stream editor). The sed utility may be considered the search-and-replace option within Unix, but its usage goes much wider than that, and it can be considered almost a programming language in its own right, with many options and sub-commands. The sed utility alone could easily fill a whole workshop. Today, we will consider only the 's' sub-command in sed. To get a flavour of how this works, compare the output of these commands:

(25) ...$ echo "bye bye blackbird" | sed 's/bye/au revoir/'

(26) ...$ echo "bye bye blackbird" | sed 's/bye/au revoir/g'

The string between the first set of slashes represents the string-to-be-replaced, and the second set of slashes contains the replacement-string. The absence or presence of the 'g' switch determines whether the replacement occurs once per line or globally. As we want to get rid of all the punctuation symbols in our corpus, we will be using it globally.

Part of the power of sed derives from the fact that, like many Unix commands (and SketchEngine query syntax), it recognizes *regular expressions* (regexps). The best known regexp is probably dot-star (.*) to mean any number (*) of any character (.) . However, regexp syntax goes much wider than that and – again – would deserve a whole workshop in its own right.

To represent all punctuation characters we would need a regexp something like:

[!"#$%&'()*+,\-./:;<=>?@[\\\]^_`{|}~]

but fortunately Unix offers us a shortcut in the shape of the expression: [[:punct:]].

We have a choice of replacing each punctuation mark with either a white space or a null string. I am going to choose a white space because (a) it will distinguish two words separated by a punctuation mark where (accidentally) no space has been left, and (b) I notice that my corpus contains some

3 If this syntax doesn't work in your shell, try using the open-quote symbol (` , top left of keyboard) twice instead of the $ and brackets, thus: cat `ls` .

URLs and I am interested in the different strings that make up those URLs.

Thus, our command has now grown to:

(27) ...$ cat $(ls) | sed 's/[[:punct:]]/ /g' (There is a space between the second set of slashes.)

Pipe this command through pg to make sure it's doing what we want it to do.

Since eventually we want to produce an ordered frequency list, we need now to get each word onto a separate line. Now that the punctuation marks have gone, we can be fairly confident that any words that do not begin a line are separated from the preceding word only by white space. Thus, by replacing each white space by a newline we can be sure that each word will appear on its own line. For this, I am going to use another sed command. I could try /<space><space>*/ (= any number of spaces from 1 upward) as my string-to-be-replaced but I notice that some lines in my corpus are indented, which could mean that tab characters are involved. Thus, I also want to get the first word of each tabbed line onto its own newline, or they may still be sticking out when I try to create my frequency list.

Fortunately, regexp syntax comes to our rescue with another shorthand string, namely [[:space:]], which covers not only literal spaces but other non-printing characters such as tabs and newlines / carriage returns themselves.

Thus, our command now becomes:

(28) ...$ cat $(ls) | sed 's/[[:punct:]]/ /g' | sed 's/[[:space:]]/\n/g'

Note the backslashed \n character between the second set of slashes, to represent a newline. We have met this *escape sequence* in command (15).

3.1.3 Using grep with regexps

Piping (28) through pg, you will see that there are a rather a lot of empty lines that are of no use to us. We are going to use the important command grep (= get regular expression) to get rid of these.

The command grep is rather like the Filter function in SketchEngine; its default behaviour is to select and output lines matching a particular (regexp) pattern in the input. To give a simple example, if you haven't deleted your days.txt file, try this command:

(29) ...$ grep 'ur' < ~/days.txt

See if you can predict the output before running the command.

The grep command has many different options that make it much more powerful than a basic filter function (check the man page). However, for now we are going to stick with something relatively simple.

To make my frequency list, I am interested only in lines with words on them. The previous sed

command will have ensured that every word begins a new line. My task is therefore to select lines that begin with an alphabetic character. In regexp, the beginning of a line is represented by a 'hat' character (^)[4]; the set of alphabetic characters (upper + lower case) is represented by the regexp [A-Za-z]. Adding the grep expression to our lengthening command gives:

(30) ...$ cat $(ls) | sed 's/[[:punct:]]/ /g' | sed 's/[[:space:]]/\n/g' | grep '^[A-Za-z]'

(**Tip:** The quotation marks are not always necessary with grep, but it is safer to include them.)

Pipe the command through pg to satisfy yourself that all the empty lines have gone.

3.1.4 The tr command

According to our original requirements, we are not interested in the distinction between upper- and lower-case, so the next stage is to convert all the upper-case characters to lower-case. We could use another sed command here, but for the sake of variety we will use the (somewhat simpler) tr (=translate) command, which simply replaces one string of characters with another. The syntax tr "[A-Z]" "[a-z]" will achieve this. (Guess what tr "[a-z]" "[A-Z]" does. See the man page for further details.)

So...

(31) ...$ cat $(ls) | sed 's/[[:punct:]]/ /g' | sed 's/[[:space:]]/\n/g' | grep '^[A-Za-z]' | tr "[A-Z]" "[a-z]"

Again, pipe the output through pg to convince yourself that the upper-case characters have gone.

3.1.5 Final sorting and counting: sort and uniq

In preparation for making my frequency list, I now want to put my words in alphabetic order. Having lost the upper-case characters makes this much easier, and a simple sort command will do.

(32) ...$ cat $(ls) | sed 's/[[:punct:]]/ /g' | sed 's/[[:space:]]/\n/g' | grep '^[A-Za-z]' | tr "[A-Z]" "[a-z]" | sort

The next stage is to count the number of instances of each word. For this, we will use the uniq command, which deletes an identical following line, so that only one instance of each line is shown in the output. At first glance this does not seem much use for our purposes (although quite useful for deleting duplicate lines in a corpus). However, uniq comes with the crucial -c option, which counts the number of deletions it has made, adds 1, and prepends the resulting number to the line – in effect counting the number of instances of any given line. Thus...

(33) ...$ cat $(ls) | sed 's/[[:punct:]]/ /g' | sed 's/[[:space:]]/\n/g' | grep '^[A-Za-z]' | tr "[A-Z]" "[a-z]" | sort | uniq -c

4 The end of a line is represented by the dollar sign ($), thus grep d$ would find every line of the input ending in the letter 'd'.

Now we have our words and their frequencies, but they are not sorted in numerical order. Two of the options of the sort command will take care of that: -n sorts according to the leading numerical value, while -r sorts in reverse order. We can combine these commands thus:

(34) ...$ cat $(ls) | sed 's/[[:punct:]]/ /g' | sed 's/[[:space:]]/\n/g' | grep '^[A-Za-z]' | tr "[A-Z]" "[a-z]" | sort | uniq -c | sort -nr [5]

Pipe through pg to see the full list. If required, redirect the whole list to a file using the > character.

3.2 Working with a concordance file

You can use similar techniques to manipulate concordance files downloaded from corpora.

For instance, for the *Language in the Real World* conference at Portsmouth in May 2012, I presented a corpus analysis of the expression 'real world'. I had an inkling that the expression was frequently used between quotation marks and wanted to know where it appeared in the 'league table' of double-quoted bigrams. Using ukWaC through SketchEngine, I first created a CQL query to create a concordance of all double-quoted bigrams:

(35) [lemma = "\""] [word = "[a-zA-Z][a-zA-Z]*"] [word = "[a-zA-Z][a-zA-Z]*"] [lemma = "\""]

Note the use of regular expressions to denote 'any two alphabetic words' [6].

Saving my concordance as a text file, I piped it through the command:

(36) … grep '< \"' | sed 's/^.*<//g' | sed 's/ >.*$//g' | sort | uniq -c | sort -nr

The grep command was necessary to distinguish between a genuine bigram and two unquoted words between two quoted words; the spacing was different, cf.

#9272 offensive. If someone picked on a "goth **<" or an " >**indie kid" in a public forum I'm not sure
#9275 offensive. If someone picked on a "goth" or an **< "indie kid" >** in a public forum I'm not sure they would

Download the same file and try the Unix command stage by stage, using pg to see what each part of the pipe is doing.

3.3 Using alias and .rc files to save your commands

Once you have written a Unix command or pipe that you are likely to use in the future, there is no need to remember it or recreate it from scratch. If you are using a bash shell, you can store it under an *alias* in your .bashrc file in your homedir and it will automatically be available to you each time you invoke a bash shell.

Files whose names begins with a dot (.) are hidden by default; files whose names have the pattern

5 sort -n -r would have the same effect.
6 In theory, I should have been able to use simply [] [] , but this did not produce the desired output for some reason.

.*<program>*rc (rc = run commands) are used to save the user's preferences for *<program>*, and are loaded each time *<program>* is run. ([7]) Check if you have a .bashrc file in your homedir by running this command:

(37) ...$ cd ~ ; ls -f | grep bash (The -f option is necessary to make the hidden files visible.)

If not, you can copy (cp) the default .bashrc file from elsewhere on the system. In Ubuntu 12.04 this file can be found in the /etc folder under the filename bash.bashrc , thus:

(38) ...$ cp /etc/bash.bashrc ~/.bashrc (There is a space before the tilde.)

Now, let's give the name freqlist to (most of) the long pipe command we wrote in Section 3.1 . We do this using the alias command:

(39) ...$ alias freqlist=' sed "s/[[:punct:]]/ /g" | sed "s/[[:space:]]/\n/g" | grep "^[A-Za-z]" | tr "[A-Z]" "[a-z]" | sort | uniq -c | sort -nr '

Notice that:
(a) we have missed out the cat $(ls) part of the original command; this is so that we can be flexible regarding the input ;
(b) to avoid confusing the alias command, which requires single quotes for the alias-ed string, all the single quotes internal to the command have been replaced with double quotes[8]

The simplest way to copy this alias command into your .bashrc file is to use any text editor or word processor, such as gedit in Ubuntu:

(40) … $ gedit ~/.bashrc

Paste the command towards the end of the file, on its own line but well away from any forbidding-looking programming code.

Now save the file, close the bash terminal, open up a new one, and type:

(41) … $ cd ManCorpus ; cat $(ls) | freqlist

This should now produce exactly the same output as the previous long command.

4 Shell programming: add2corpus

For more complex operations, it is possible to save a sequence of Unix commands together in the same text file, known as a *shell script*. At this point, writing Unix commands becomes computer programming in the proper sense of the term. The Unix syntax allows for such programming

7 The set of .rc files can be likened to the Windows registry, except that they are easy to find and edit, and will not do any great harm if they contain errors.

8 The use of quotation marks in Unix commands is quite fraught. In many contexts, with some crucial exceptions, single and double quotes (and sometimes no quotes at all) will work in exactly the same way. However, certain traditions and conventions have grown up around which type of quotes are used with which commands.

concepts as variables, if-conditions, and for-loops.

In preparation for this workshop, I have written a rough-and-ready Unix shell script called add2corpus, which converts batches of .doc , .docx , and .pdf files to plain text for use in a corpus. This can be downloaded, together with its associated README file, from my public file depository:

http://bit.ly/jw-public

The full functioning of this shell script depends on the user having three separate utilities (not written by me) installed on their system:

a) catdoc , for converting .doc files – see the README for the download address; under Ubuntu Linux, I was able to download and install this simply using the command:

(42) ...$ sudo apt-get install catdoc

b) docx2txt.pl , for converting .docx files - see the README for the download address

c) pdftotext , for converting .pdf files ; this utility is native to my version of Ubuntu Linux

The utilities catdoc and docx2txt.pl are written in two different programming languages (C and perl) respectively. Both authors make life slightly easier for you by including with the download package something called a *Makefile*, which gives the computer automated instructions for compiling (in the case of catdoc) and installing the utility.

For each package, download and unzip it to its own folder, navigate to that folder, and simply type the command:

(43) ...$ make

This tells the computer to read the Makefile and follow the compilation/installation instructions[9].

By convention, the user's own programs are stored in the ~/bin/ (= binary) directory. Create this directory if it is not there already and store add2corpus in it. Before the computer will recognize add2corpus as a valid command, you may need to do two things:

(a) change its 'mode' so that the computer recognizes it as an 'executable' program rather than just a text file ; the command

(44) ...$ chmod a+x add2corpus

will achieve this (a = by all ; x = executable)[10].

(b) add the following line to your .bashrc file:

9 These steps are unnecessary for catdoc if command (42) works for you.
10 If you want to restrict usage of this program to yourself, replace a with u (= user).

(45) export PATH=$PATH:$HOME"/bin"

$PATH is a user-controlled *environmental variable* which tells the computer all the locations where it is permitted to look for executable programs. The effect of the above command is to add your bin directory to this list.

You should now be able to use add2corpus from the command line (see the README first).

5 Further reading

In this workshop, we have barely scratched the surface of the Unix command line, but you have been introduced to several of the most useful commands from a corpus linguist's point of view: grep, sed, sort, uniq...

Unix tutorials abound on the internet – unfortunately many of them are written in an opaque style that seems to assume that you know everything about Unix already! This portal is a good enough place to get started:

http://www.cyberciti.biz/tips/linux-unix-commands-cheat-sheets.html

The following site is very good on sed, but I wouldn't necessarily recommend it for anything else:

http://www.grymoire.com/Unix/Sed.html

The following site is a good guide to regular expressions, but it is also selling a product:

http://www.regular-expressions.info/tutorial.html

If you prefer your information in book form, you can't really beat the O'Reilly guides:

http://www.amazon.co.uk/s/ref=nb_sb_noss?url=search-alias%3Daps&field-keywords=o'reilly%20unix

And of course there is a volume in the For Dummies series (unseen):

http://www.amazon.co.uk/s/ref=nb_sb_noss?url=search-alias%3Daps&field-keywords=unix+dummies&rh=i%3Aaps%2Ck%3Aunix+dummies

John Williams
john.x.williams@port.ac.uk
November 2012