Lab 1: basic UNIX tools

Author: Sharon Goldwater
Date: 2014-09-01, updated 2015-09-15

This lab is available as a web page[2] or pdf document[3].

Preamble: how to do the labs for this course

This course has a weekly one-hour lab session where you will work through the instructions and questions either from the web page or the PDF version, which we will have as a handout in the lab room. You should work **with a partner**, where the two of you share a single keyboard and screen. Each person needs to be responsible for making sure that both they *and* their partner understand what is going on. You should stay engaged with what your partner is doing and discuss what's going on all times. That means:

- If you know the answer and your partner doesn't, *don't* just type it in and move on to the next question---explain what you have done and why. This not only helps your partner to learn, but will also clarify your own understanding. Consider whether you could give other examples that would help your partner understand even better, and ask if they are satisfied with your explanation. Being able to think around a concept and explain it well to other people is an important skill to learn, and not always easy.
- If your partner knows the answer and you don't, *don't* let them move on until you understand too. And if you are at the keyboard, *don't* let them simply dictate what you should type if you don't understand why. *Do* ask questions, and let your partner know if their explanation makes sense or not. Remember, just because they think they know the answer doesn't necessarily mean they are right, and if it doesn't make sense to you, they need to work harder at explaining (and maybe discover that they are wrong!)
- If neither of you knows the answer, ask one of the demonstrators in the lab, and we can try to help you.
- If one person is much more familiar with Python than the other, consider putting the weaker person at the keyboard or at least switching frequently, so they will get more practice with basic coding skills.

The labs are designed so that you should be able to work through the main set of questions in an hour (we hope! This is the first year for the labs, so your feedback is appreciated). However, we would strongly encourage you to go beyond the basic set of questions to further improve your practical skills and understanding. We have included a number of additional questions at the end of each lab for this purpose. If you finish the main questions in less than an hour, choose one or two of the additional ones to work on.

NOTE: This week's lab is quite wordy; future labs should involve more doing and less reading, but due to this week's content, we weren't able to think of a way around the wordiness.

Preliminaries: to do before the first lab session

Please do the following **before** arriving at the first lab session.

[1] http://creativecommons.org/licenses/by-nc/4.0/.
[2] http://www.inf.ed.ac.uk/teaching/courses/anlp/labs/lab1.html
[3] http://www.inf.ed.ac.uk/teaching/courses/anlp/labs/lab1.pdf
[4] http://computing.help.inf.ed.ac.uk/new-taught-students
[5] http://computing.help.inf.ed.ac.uk/sites/default/files/IntroLecture201516.pdf
[6] http://www.researchgate.net/publication/228560160_Unix_for_Poets/file/79e415120007ae7efb.pdf

1. Make sure you have a DICE (Distributed Informatics Computing Environment) account. All students on an Informatics degree should already have an account. If you are on a degree in another school, you will automatically get a DICE account once you have registered for this class, so if you haven't yet registered, do so as soon as possible. If it looks like you might not have an account for a few days, we strongly encourage you to pair up with another student who already has an account and go through the lab together using their account.

2. Read the School's Computing Support information page for new taught students[4] and introduction to DICE[5]. In particular, in the rest of the lab we will assume you have read sections 1-4 and 8 (at least up to "other things to consider") of the introduction to DICE. You will not need to use Emacs (section 9) for this lab, but you *will* need to use it for later labs in the course (unless you already use some other UNIX-friendly text editor such as VI), so you may want to read that section now too.

Goals and motivation of this lab

This course is taken by students with many different backgrounds. Computer science students who are already familiar with UNIX may not need to do this lab, but we still suggest skimming through to make sure you know all the commands we discuss and can do the tasks at the end of the lab. You may also want to take a look at the Going Further section for more practice.

For everyone else: the purpose of this lab is to get you familiar with the command line tools available to you in the UNIX operating system. This operating system is commonly used by computing researchers and programmers; here in Informatics we are running a version called Scientific Linux, which is installed on all of the student lab machines and which you will be using for the practical labs in this course. (The UNIX tools described here are also available in Apple's OS X, and in Windows by installing additional software such as Cygwin.)

UNIX commands can be a bit confusing at first but provide a lot of powerful tools that can be used for manipulating files and data, exploratory analysis, and sanity checking when you are implementing a larger program. This lab will introduce you to some basic UNIX commands for exploring and processing files. You can do a surprising amount of useful stuff with just a few commands, and we hope you'll continue to use the skills from this lab in the remainder of the semester and in your other courses.

Specifically, we will explore the data from a corpus of parent-child interactions, building up to computing the child's "mean length of utterance" in one or two files. Mean length of utterance (MLU) is a measure of child language development which refers to the average number of words (or, sometimes, morphemes) in each of the child's utterances (spoken sentences).

Creating your lab directory

First, log into your DICE account and open a *shell* (also called a *terminal*) following the instructions in the Introduction to DICE document. You should see a *command prompt* of the form:

```
[hostname]username:
```

Now, create a directory called `anlp` that you will use for all your ANLP work. You can do this by entering the following command at the prompt. *(Remember,* `mkdir` *stands for 'make directory')*

```
mkdir anlp
```

Now, create a subdirectory for your labs, and then another one for this lab. *(You cannot create a subdirectory before creating the directory it belongs inside.)*

```
mkdir anlp/labs
mkdir anlp/labs/lab1
```

Next, `cd` (meaning `change directory`) into the directory you just created:

```
cd anlp/labs/lab1
```

If you ever get confused or forget which directory you are in, use the `pwd` (*print working directory*) command. Try it now:

```
pwd
```

Downloading the data

In this lab we will be working with data from CHILDES, the Child Language Data Exchange System. CHILDES is a large repository containing many different corpora in many different languages, all contributed by researchers interested in child language development. Since the particular interests of the different researchers vary considerably, the different corpora contain different types of data (transcriptions, audio, and/or video) and different kinds of annotations (ranging from detailed phonetic transcripts to morphological and syntactic annotations). However all the corpora are annotated using similar guidelines, so that tools can be developed to work with the annotations across different corpora.

We will use the Providence corpus for this lab. To download the corpus, click on the following URL or paste it into your web browser: http://homepages.inf.ed.ac.uk/sgwater/teaching/Providence.zip. You should get a dialog box asking what to do with the file; choose `Open with Archive Manager (default)`. In the next window, click `Extract` and then navigate to your `lab1` directory by double-clicking in the right-hand pane on the `anlp` folder, then `labs`, then `lab1`. Then click `Extract`.

ls

Go back to your terminal window. You should still be in your `lab1` directory. Now let's *list* the contents of that directory using `ls`:

```
ls
```

You should now see a subdirectory called `Providence`, which contains the data you just downloaded. What do you see in it? What about in the further subdirectories?

```
ls Providence
ls Providence/Ethan
```

Looking at the data

less

To get an idea of what is in the files you just downloaded, type:

```
cd Providence/Ethan
less eth01.cha
```

What information is in the metadata at the top of each file? *(Hint: child language researchers use the format y;m.d to indicate a child's age in years;months.days)*

(`less` may seem like a funny name for this command which shows you *more* of the file; it's because there was an earlier similar command called `more`, so when this newer version was developed the developers decided to give it a cute name. UNIX developers like cute names.)

When you are looking at a file using `less`, you can scroll up and down using the arrow keys or `<PageUp>`/`<PageDown>`. `<space>` also acts like `<PageDown>` and `<Enter>` acts like `<down-arrow>`.

What do you see in the rest of the file? *(Hint: the string of numbers at the end of each line is a code that links to a time point in the audio recording of this data. The audio isn't included here but can be obtained from the CHILDES database.)*

One final useful thing you can do with `less` is search for a particular string in the file by typing `/` followed by the string. Try it: type

```
/the
```

You should see all of the instances of `the` being highlighted.

To stop viewing the file and go back to the terminal command line, type `q`.

Actually, the `eth01.cha` file is maybe not one of the more interesting ones. Take a look now at `eth50.cha`. What are some of the main differences you see between the data in these two files? Is there an obvious explanation for those differences? (If you want to look at both files simultaneously, you could start a new terminal and open one file in each terminal.)

head, tail

Sometimes we just need a quick peek at part of a file. What does this command do?

```
head eth01.cha
```

What about this one?

```
tail eth01.cha
```

These commands can be useful, for example, if we want to look over the range of ages in the files we have. Try:

```
head *.cha
```

What does the * do?

man

But, we don't really need to look at the first 10 lines of each file to see the ages, we only need the first five lines. If we could print only the first five lines, we could see the information we want more compactly. Most UNIX commands, including `head`, have many possible options to change their behavior. To see what options are available, look at the *manual* for the command:

```
man head
```

You should see:

```
HEAD(1)                          User Commands                          HEAD(1)

NAME
       head - output the first part of files

SYNOPSIS
       head [OPTION]... [FILE]...

DESCRIPTION
       Print  the  first  10 lines of each FILE to standard output.  With more
       than one FILE, precede each with a header giving the file  name.   With
       no FILE, or when FILE is -, read standard input.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

       -c, --bytes=[-]K
              print the first K bytes of each  file;  with  the  leading  `-',
              print all but the last K bytes of each file

       -n, --lines=[-]K
              print  the first K lines instead of the first 10; with the lead-
              ing `-', print all but the last K lines of each file

       -q, --quiet, --silent
              never print headers giving file names

       -v, --verbose
              always print headers giving file names

       ...
```

Note: you should be able to move up and down in the man pages using the same keys you used for `less` *[I hope; the default for the student account might not allow you to move upwards...], and you can also return to the command line using* `q`.

The man page starts by giving the name of the command and a very brief synopsis of how to use it---in this case, showing that the `head` command can be followed by zero or more *options* (sometimes called *flags*) and then zero or more *files* as arguments (things to act on). The square brackets indicate that these options and files need not be included at all, and the `...` indicate that you can include more than one of each.

The description after the synopsis describes what will happen if you include zero or multiple files. (*Standard input* refers to the text you input in the terminal. Try entering `head` with no filename to see how this works. You will need to enter some more text after that! To quit, type Ctrl-c.)

The next part of the description tells you what the possible options are and if they require arguments themselves. Many (in this case all) options have both a short and a long form, which perform equivalently. For example, to print just the first 5 lines of each file, we could either use:

```
head -n5 eth01.cha
```

or:

```
head --lines=5 eth01.cha
```

(If you want to try this yourself, you will need to either use another terminal or quit the man page first by typing `q`.)

The K specified in the description is a variable indicating a required argument to the -n option. Here we use the value 5 for K. Notice that this option also has a non-required argument [-] so if we wanted to print all but the last 5 lines we could type:

```
head -n-5 eth01.cha
```

As noted above, you can specify multiple options at once. Compare the output of the following commands:

```
head -n5 *.cha
head -n5 -q *.cha
```

Now that you know how to read a man page, you may want to look at the man pages for some of the other commands we've seen to get an idea of what options are available for them.

It's also worth pointing out that there is a *lot* of information available on the Internet to give you help with these commands, and sometimes has more examples or is easier to understand than the man page. If you know the name of the command you want to use, say `grep`, try searching for `UNIX command grep`. If you know what you want to do but not the name of the command, searching would be a bit trickier, but try a few combinations like `UNIX search regular expression` or even `UNIX find string in file`. It's good to get in the habit of looking for help--you will probably start to recognize some of the main help forums and which ones are most useful to you.

Being lazy

As you may have noticed, most UNIX commands are very short to avoid lots of typing. You can avoid even more typing by using *tab completion* and *history*.

Tab completion

Type the following into your terminal. Instead of pressing <enter> at the end, press the <tab> key:

```
head e
```

You should find that the filename has been automatically extended to `eth`. If you type `01` and then <tab> again, you will have the complete filename. In this case the tabbing didn't save you that many keystrokes, but for long filenames it can be extremely useful. You can use tab completion for just about anything in UNIX (commands as well as filenames and directories), the operating system will always extend what you are typing as much as it can but will stop as soon as there are multiple possible extensions (as here, where all files in the current directory start with `eth` but we then had to type in `01` ourselves to disambiguate before tabbing again.) Under DICE, tab completion is set up so that if you hit <tab> again when there are multiple completions, the system will print out all possible completions for you to see.

History

Your terminal keeps track of all the commands you have entered previously in order. To access the previous command, just type the up-arrow key. If you hit <up> twice, you will get the command before that, and so forth. This can save a lot of typing.

Finding and counting things in files

wc

Try this command:

```
wc eth01.cha
```

What are the numbers that you got back? What do you think `wc` stands for? *(Hint: look at the man page!)*
Using a single command, can you figure out which of the first nine Ethan files has the most lines in it?

grep

Sometimes we want some more specific information. The number of lines in a file is a rough indication of how much language data is in there, but what if we want to know how many utterances are spoken by a particular person? Try this command:

```
grep 'MOT' eth01.cha
```

Note: those should be just regular single quotes, although they may render differently in the pdf version of this lab.

What did this command do?

Using the man page for `grep`, figure out what option you can use with `grep` to compute the number of utterances spoken by the mother in this file.

In case you are wondering, `grep` stands for *Globally search for a Regular Expression and Print*. We won't go into regular expressions (REs) in detail here, but they are covered in Section 2.1 of the Jurafsky and Martin textbook, and also in the Computer Programming for Speech and Language Processing course. (The `grep` man page also explains the syntax that `grep` uses for REs.) REs are a powerful way to match patterns in text and we hope you will learn to love them. For the moment, we will just show you a few simple REs to give you a flavor of what you can do.

First, look at this command but don't run it yet:

```
grep 'MOT.*the' eth01.cha
```

What do you think it will do? *(Hint: `.` matches any character, and `*` means "zero or more repetitions of the previous character")*

To see if you were right, run the command and look at the first few lines of output. Even better, let's only print out the first few lines of output. We can do that by combining two commands we already know, `grep` and `head`:

```
grep 'MOT.*the' eth01.cha | head
```

pipes and redirection

What just happened?

The `|` (vertical bar on the keyboard, called *pipe* in UNIX-world) tells the operating system to use the *output* of the first command as the *input* to the second command. (Another way to think of this is that it turns the output of the first command into "standard input", which as you will recall, is what is used as input to `head` if no filename is given.

Can you think of a way to use pipe to combine two commands we already know to count the number of lines that include the string `MOT` *without* using the `-c` option for grep?

Instead of sending the output of one command to the input of another, we can also just send the output of a command into a file. This can be useful if we want to save the output for later inspection, and we do it using the > (output redirect) character:

```
grep 'MOT.*the' eth01.cha > my_output.txt
less my_output.txt
```

grep again

Coming back to regular expressions, let's make sure you understand what the RE `'MOT.*the'` is matching in the grep command. What is the difference between the following two commands?

```
grep 'MOT.*the' eth01.cha
grep 'MOT.*\bthe\b' eth01.cha
```

You may want to look at just the start of the output by piping it to `head`. Or try using the `--color=auto` option for `grep` (although unfortunately the color is lost if you try to pipe the output).

In general, backslashes are used in regular expressions to indicate something special (like a word boundary or any whitespace character) and are also used to make characters that are *usually* special (like `*`) be interpreted literally. So, for example, if we want to match only the lines spoken by the mother, we should use `'^\*MOT'` rather than just `'MOT'`, because the former version only matches the exact string `*MOT` at the beginning of a line (indicated by the `^`). The latter version will also match lines that are not spoken by the mother but happen to contain the string `MOT`. (Are there any?)

As you can start to see, regular expressions can be very powerful but proper use takes time and practice, and we will not go any further here.

Computing MLU

Believe it or not, we now have all the commands we need to be able to compute the child's mean length of utterance in any given file. Remember, MLU is the average number of words spoken by the child in each of the utterances in the file. What is Ethan's MLU in the file `eth50.cha`? Assume for this question that a word is any whitespace-delimited string of characters (including punctuation) in the transcription. So, for example, you should count `fill him with pom+poms xxx .` as six words.

Hint: you can't get the answer entirely automatically, you will need to use the commands you know to get a few key numbers and then perform some basic arithmetic on them yourself. This method is fine if we only care about the MLU for one or two files. If we wanted to compute MLU for all *the files, we would probably want to write a short program to do it.*

Going Further

If you want, try your hand at one of the following:

1. We showed how to get all the lines from a file that contain the word 'the'. But what if we actually want to count the number of occurrences of 'the'? `grep` by itself doesn't really work since there could be multiple occurrences on a single line. By reading the man page or searching the Web, figure out how the `tr` command works, and use it together with commands we've seen to count the number of occurrences of 'the' in a file.

2. We've covered a few useful commands here, but there are quite a few more that can come in handy for text processing and other NLP-related tasks. To learn more about those, you may want to go through the Unix for Poets[6] tutorial. [Please note that you do *not* need to sign up to Research Gate to read this tutorial, simply click the 'view' button in the right-hand pane of the Research Gate landing page.]