Using Unix tools - Examples from PDB

The Unix toolkit is built round a simple philosophy. Each application performs a simple task well. Combining multiple applications allows for the building of complex applications that can answer complex questions.

This tutorial will introduce various tools and demonstrate the functionality that can be obtained by combining them. As an example we will be using predominantly data obtained from X-ray crystal structures.

This tutorial is not intended to provide a 'cookbook' to cover all eventualities. It is intended to show the sorts of things that can be done with the tools available and how they can be made to work together in a powerful and effective way.

Files:

You will need the following files in an appropriate directory (it may be a good idea to make one now for this exercise).

```
1abm.pdb
1boy.pdb
1cyo.pdb
1sup.pdb
1tph.pdb
1trn.pdb
2src.pdb
```

Your course instructor will tell you where these can be obtained, or you can get them on the web using SRS (eg. `http://srs.ebi.ac.uk`) or from the protein databank (`http://www.rcsb.org`).

Conventions:

Throughout this tutorial the % sign indicates a Unix prompt. Your prompt may vary. Do not type the %, just type what follows it. Input (that you type) is in **`bold courier`**. Output (that the computer shows) is in `normal courier.`

1. Basic tools

1.1.Counting.

The command `wc` allows us to count the lines, words and characters in a file. Giving the command multiple filename arguments will result in a summary for each of the files.

```
% wc *.pdb
   3474   42339  281394 1abm.pdb
   2106   21855  170586 1boy.pdb
   1059   12106   85779 1cyo.pdb
   2397   26866  194157 1sup.pdb
   4162   50293  337122 1tph.pdb
   3931   47375  318411 1trn.pdb
   4286   45698  347166 2src.pdb
```

```
  21415  246532 1734615 total
```

You can probably guess that the three numbers are lines, words, and characters respectively.

`wc` takes several options to limit the output to words, characters or lines. We will be using the `-l` (that is the letter `l`, not the number `1`) option to just count the number of lines. You can use the `man` command to read about the other options (`% man wc`).

```
% wc -l *.pdb
   3474 1abm.pdb
   2106 1boy.pdb
   1059 1cyo.pdb
   2397 1sup.pdb
   4162 1tph.pdb
   3931 1trn.pdb
   4286 2src.pdb
  21415 total
```

Instead of taking filenames as arguments, many of the unix tools will act on standard input (`stdin`) and print the results to standard output (`stdout`), allowing them to be chained together in pipelines.

1.2. Sorting

Not surprisingly the unix command sort will sort the contents of a file (or `stdin`). It will also merge and sort multiple files. We can sort our list of files by length.

```
% wc -l *.pdb | sort
   1059 1cyo.pdb
   2106 1boy.pdb
   2397 1sup.pdb
   3474 1abm.pdb
   3931 1trn.pdb
   4162 1tph.pdb
   4286 2src.pdb
  21415 total
```

We have actually been quite fortunate here. `sort` will sort alphabetically by default and our list just happens to be sorted alphabetically (this isn't strictly true. Alphabetical in this case means in the order of the ascii character value. Space is the lowest number used, followed by uppercase characters then lowercase. `man ascii` will give you the codes for all the characters). To show this is the case we can tell `sort` to ignore blank characters (spaces and tabs) at the beginning of the line with the `-b` option.

```
% wc -l *.pdb | sort -b
   1059 1cyo.pdb
   2106 1boy.pdb
  21415 total
   2397 1sup.pdb
   3474 1abm.pdb
   3931 1trn.pdb
   4162 1tph.pdb
   4286 2src.pdb
```

This is still in alphabetical(ish) order but ignoring the blank characters at the beginning of

the line.

To sort in numerical order we need to use the option `-n` which implies the `-b` option. This gives the same output seen at first but obtained using the correct rules.

1.3. Heads or tails

It is often useful to be able to rapidly scan a number of files to find the one you want. Supposing you are looking for the structure of chicken triosephosphate isomerase and know it is one of the seven files in this directory. It is perfectly possible to use `more` to look at each file but this is somewhat tedious, especially as we know that a PDB file contains a header in the first line that describes the structure. The command to look at the first few lines of a file is `head`.

```
% head *.pdb
==> 1abm.pdb <==
HEADER    OXIDOREDUCTASE                          27-AUG-92   1ABM      1ABM   2
COMPND    MANGANESE SUPEROXIDE DISMUTASE (E.C.1.15.1.1)                 1ABM   3
SOURCE    HUMAN (HOMO SAPIENS) KIDNEY RECOMBINANT FORM EXPRESSED        1ABM   4
SOURCE   2 IN (ESCHERICHIA COLI, STRAIN SODASODB)                       1ABM   5
AUTHOR    G.E.O.BORGSTAHL,H.E.PARGE,J.A.TAINER                          1ABM   6
REVDAT   1   31-OCT-93 1ABM    0                                        1ABM   7
JRNL        AUTH   G.E.O.BORGSTAHL,H.E.PARGE,M.J.HICKEY,               1ABM   8
JRNL        AUTH 2 W.F.BEYER JUNIOR,R.A.HALLEWELL,J.A.TAINER           1ABM   9
JRNL        TITL   THE STRUCTURE OF HUMAN MITOCHONDRIAL MN3+           1ABM  10
JRNL        TITL 2 SUPEROXIDE DISMUTASE REVEALS A NOVEL TETRAMERIC     1ABM  11
==> 1boy.pdb <==
HEADER    CLASS 2 CYTOKINE RECEPTOR               11-JAN-96   1BOY
TITLE     EXTRACELLULAR REGION OF HUMAN TISSUE FACTOR
COMPND    MOL_ID: 1;

.
.
.

COMPND   5 DOMAINS AND C-TERMINAL TAIL;
COMPND   6 SYNONYM: C-SRC, P60-SRC;
COMPND   7 EC: 2.7.1.112;
```

This is firstly more information than we need (we only want the first lines) and secondly has run off the top of the screen. We can address both these points. To set the number of lines shown we can use the `-n` option and we can pipe the output into `more` (using the pipe character `|`) to view the results a page at a time.

```
% head -n 1 *.pdb | more
==> 1abm.pdb <==
HEADER    OXIDOREDUCTASE                          27-AUG-92   1ABM      1ABM   2

==> 1boy.pdb <==
HEADER    CLASS 2 CYTOKINE RECEPTOR               11-JAN-96   1BOY

==> 1cyo.pdb <==
HEADER    ELECTRON TRANSPORT                      03-AUG-94   1CYO      1CYO   2

==> 1sup.pdb <==
HEADER    HYDROLASE (SERINE PROTEASE)             14-AUG-95   1SUP      1SUP   2

==> 1tph.pdb <==
HEADER    TRIOSEPHOSPHATE ISOMERASE               22-DEC-93   1TPH      1TPH   2

==> 1trn.pdb <==
HEADER    HYDROLASE (SERINE PROTEINASE)           16-MAR-95   1TRN      1TRN   2

==> 2src.pdb <==
```

```
HEADER    TYROSINE-PROTEIN KINASE                 29-DEC-98   2SRC
```

We can now see our structure of interest is `1tph.pdb`.

A similar command to `head` is `tail` which outputs the last few lines of a file and takes most of the same options as `head`. Again, using the `man` command will give you far more information on these commands.

1.3.1. Exercise

Output lines 900-904 of any one of the files in this directory. Additionally count the number of characters in those lines.

1.4. Finding

It is often necessary to locate particular lines in a file based on the words in that line. For example one may wish to find the lines containing a particular journal name or author in a long text file. Or one may wish to extract the alpha carbon coordinates from a pdb file. The program `grep` can help us here. We may also wish to locate files modified on or after a particular date, or by other attributes. `find` can help us here.

1.4.1. `grep`

`grep` is a shortened version of the phrase '*get regular expression*'. A regular expression is a pattern which can match a particular sequence of characters. More information on regular expressions (and indeed on all these commands) can be found by consulting the bibliography at the end of this tutorial. The syntax for grep is

`grep` *`options searchexpression filelist`*

where *`options`* are the options that `grep` can take, *`searchexpression`* is the phrase we are looking for (remember to put phrases containing spaces in quotes) and *`filelist`* is a list of files to search.

If we are looking for Dr Martin in this set of structures then we can use the following command

```
% grep MARTIN *.pdb
1boy.pdb:JRNL        AUTH   K.HARLOS,D.M.MARTIN,D.P.O'BRIEN,E.Y.JONES,
1boy.pdb:REMARK   1  AUTH   C.W.BOYS,A.MILLER,K.HARLOS,D.M.MARTIN,
1cyo.pdb:REMARK   1  EDIT   A.N.MARTINOSI
1CYO  17
```

The output is every line containing that sequence of characters prefaced by the filename (if multiple files were specified).

Note that we have found a line that is not quite what we were looking for. The string (a string is a sequence of characters) `MARTIN` matches `MARTINOSI` as well. Also note that the search is case sensitive. This search will not match lines containing `Martin`.

We can count the number of atoms in a structure using `grep` in combination with the `wc` command. Each atom in a normal residue is prefixed by the string `ATOM`. Searching for this string with `grep` then counting the lines found with `wc` will hopefully count the number of atoms in the structure. By now you should be familiar with the pipe and be able to put together the following command

```
% grep ATOM 1boy.pdb | wc -l
    1693
```

The file however states that there are only 1685 protein atoms.(you can of course view the file with `more 1boy.pdb`) Where are the extra atoms coming from? Taking the output from `grep` and looking at it with `more` should give you the answer. How can we get rid of the extra lines?

`grep` has an option `-v` that reverses the selection. It means *'select all lines except those that match the pattern'*. We can use this in combination with the earlier command to weed out most of the extras.
Many of the extra ATOM matches are to the word ATOMS so we can use that as a filter.

```
% grep ATOM 1boy.pdb | grep -v ATOMS | wc -l
    1686
```

Still too many, but looking at the first line (by looking at the output with more) we see a match to `ATOM` in the middle of a line. All the other matches are at the start of a line. We can add a space before `ATOM` as a filter.

```
% grep ATOM 1boy.pdb | grep -v ATOMS | grep -v ' ATOM' | wc -l
    1685
```

Note that in order to get the command to include the space as part of the search string we had to surround it with quotes.

`grep` has many other options that are listed in the `man` page. There are other ways to get to this particular result. One of the joys of unix is that it is very flexible, providing many ways to get to the same eventual answer.

1.4.2. Exercise

The format of an `ATOM` record is like this:

```
ATOM      9  CA  THR     4      10.468  -5.167 -12.900  1.00 84.44           C
```

The third field (the `CA`) is the atom type, in this case an alpha carbon. Count the number of alpha carbon atoms in each of the pdb files.

1.4.3. `find`

`find` can be used to locate files by date. If we wished to find which files have been added to the local copy of the PDB database in the last two weeks then we can use a command like the following (note that the pdb files are located in `/site/databases/pdbdiv/pdb` and subdirectories thereof):

```
% find /site/databases/pdbdiv/pdb -mtime -14 -name '*.ent.Z'
```

(output not shown)

More information on `find` can be found in the `man` page.

1.5. Chopping and Changing

So far we have seen how to select lines from files. Now we will see how to manipulate them. A simple command to take part of a line is `cut`. A more advanced program to manipulate parts of a line is `awk` and `sed` can be used to modify and otherwise manipulate lines. The latter two are extremely powerful and we will just get a flavour of what they can do.

1.5.1. `cut`

`cut` allows part of a line to be selected. It will chop out a defined portion of the line, either between certain character positions or by splitting the line based on a specific character. As PDB records are based on a fixed width it is relatively easy to select the portion of the line we want to select. Suppose we want to extract the `x` coordinate for the atoms. This is in columns 32-38 of each record. We also want the residue name and number (columns 17-25). Using essentially the same command as before to select the ATOM lines, we can then cut out the bits we want.

```
% grep ATOM 1boy.pdb | grep -v ATOMS | grep -v ' ATOM' | cut -c
 17-25,32-38
```

This command should be all on one line. (output not shown)

One disadvantage of `cut` is that it just cuts.. it doesn't allow one to change the output, merely select it. The next tool is far more powerful.

1.5.2. `awk`

`awk` is a tool that manipulates records. It assumes that every text file is some sort of structured document containing *records* that are divided into *fields*. Each *record* is separated by a *record separator* and each *field* by a *field separator*. `awk` allows the individual fields to be manipulated and incorporates a programming language to allow decision making. `awk` is a complex tool and you are strongly advised to refer to the book *'sed & awk'* from O'Reilly press. (See the bibliography)

By default the *record separator* is a newline and the f*ield separator* is whitespace (spaces or tabs). This will allow us to do the same manipulation as we did for `cut`. When `awk` separates each *record* into *fields*, these *fields* can be selected as the variables `$1`, `$2` and so on. `$0` is the whole record.

```
% grep ATOM 1boy.pdb | grep -v ATOMS | grep -v ' ATOM' | awk '{ print
$4, $5, $6 }'
```

This command should be written on one line. (output not shown)

One subtle difference is that `cut` left multiple spaces between the characters. `awk` has reduced each space to just one character. This is the effect of putting the commas between the variables in the output. (A variable is a named value, just like using letters to stand for numbers in algebra except that variable names can be any length. You can get the value of a variable by putting the $ in front of the name.) If we try this again without the commas then we lose the spaces. `awk` replaces each comma with the .

```
% grep ATOM 1boy.pdb | grep -v ATOMS | grep -v ' ATOM' | awk '{ print $4
$5 $6 }'
```

This command should be written on one line. (output not shown)

We can also add extra text into each field as desired by including that text between quotes in the print statement.

```
% grep ATOM 1boy.pdb | grep -v ATOMS | grep -v ' ATOM' | awk '{ print $4
,$5,"X:", $6 }'
```

This command should be written on one line. (output not shown)

1.5.3 Exercise

Persuade awk to print the x, y and z coordinates for each alpha carbon atom as

x: *x* y: *y* z: *z*

where *x*, *y* and *z* are the values for the x, y and z coordinates. What problems do you find and why?

1.5.4 More awk

awk also allows commands to be run at the beginning and at the end of the report. The commands inside the {} are run for every line. To run commands at the beginning we place them inside another set of braces prefixed by the key word BEGIN. Lets add a title to the report and change it a little by only selecting threonine residues.

```
% grep ATOM 1boy.pdb | grep -v ATOMS | grep -v ' ATOM' | grep THR | awk
' BEGIN{ print "Threonine residues"} { print $4, $5}'
```

This command should be typed on one line. (output not shown)

This has printed every threonine residue. It has done more than that, it has printed the residue name and number for each atom in the threonine residues. To shrink this down to one per residue we can either select just eg. alpha carbons or we can use the unix command uniq. uniq will reduce multiple adjacent lines with the same output to just one line. The lines must be adjacent so we would normally precede uniq with sort but it isn't neccessary in this case as the lines are already appropriately sorted.

```
% grep ATOM 1boy.pdb | grep -v ATOMS | grep -v ' ATOM' | grep THR | awk
' BEGIN{ print "Threonine residues"} { print $4, $5}' | uniq
```

This command should be typed on one line. (output not shown)

Counting these residues by appending wc -l will not give the right number as we have added a header line. We can however get awk to do the counting for us. In this case we will count the total number of atoms present. In the BEGIN block we can set a variable to be a counter and start it at 0. Each time we print a line we can add 1 to the counter variable. At the end we can print a summary line by using an END block that does the same as the BEGIN block but at the end. We can get the value of the counter just by putting the name of the variable (no $ needed). note that individual commands are separated by semicolons (;).

```
% grep ATOM 1boy.pdb | grep -v ATOMS | grep -v ' ATOM' | grep THR | awk
' BEGIN{ print "Threonine residues"; counter=0;} { print $4, $5; counter
= counter+1} END{print "Total:", counter, "atoms"}' | uniq
```

This command should be typed on one line. (output not shown)

1.5.5. Example

Modify the above command to count the number of residues instead. (Hint: you will need to use more than one awk command)

1.5.6 Making choices in awk

This is getting rather complex. However we are still using a set of basic tools that can be put together in many ways to answer difficult questions.

In a crystal structure the quality of an atoms position is stated as the B-factor. This is included in the PDB file as field 10. The higher the B-factor the worse the quality of the position. If we want to select from our threonine residues those with a poor B-factor then we can do this using a conditional statement, `if`. We put the condition we want to compare in brackets and the statements to execute if the condition is met in braces afterwards. (you can also add an `else {}` but we are not going to do that here. Read the sed & awk book to find out more!.) The syntax for the `if` statement in `awk` is

```
if ( condition ) { do if true }
          or
if ( condition ) { do if true } else { do if false }
```

where *`condition`* is a statement that evaluates to true or false and *`do if true`* and *`do if false`* are blocks of commands to execute if *`condition`* evaluates to true or false respectively.

```
% grep ATOM 1boy.pdb | grep -v ATOMS | grep -v ' ATOM' | grep THR | awk
' BEGIN{ print "Threonine residues"; counter=0;} { if ($10>50) {print
$4, $5; counter = counter+1}}  END{print "Total:", counter, "atoms"}' |
uniq
```

This command should be typed on one line. (output not shown).
awk is an extremely powerful tool and this hopefully has given you a taste of what it can do.

1.5.7. sed

Sometimes we will need to make many identical changes at various points through a file.

sed looks for lines matching a certain pattern (or between lines matching certain patterns) and can carry out various operations on them. The one we are interested in is substitution. One should note that sed can read from a file (or `stdin`) but writes to `stdout` so if you want to keep the results then you will need to redirect the output to a file (with > *`filename`*).

Let us assume for some reason or other that we have determined that all the methioine residues in our structure are selenomethionine and we wish to rename them. Try something like the following:

```
% sed -e ' /^ATOM/s/MET/SEL/' 1boy.pdb
```
(output not shown)

If you were smart and used `more` or `grep` to view the output then you will have seen that all the MET residues were changed to SEL.

1.5.8. Example

Try to change all the water atoms (HOH) to WAT. (hint: look at how you are choosing the lines).

`sed` can do a lot of other things too. Read the book!