Unix exercises

> NB: When you see a `$` given in the example prompts, it is a metacharacter indicating that this is the prompt for a normal (not super-user) account. Your default prompt may be different, and it is highly configurable (search for "PS1 unix prompt" to learn more). Anyway, point is that you should type (copy/paste) all the stuff *after* the `$`. If you ever see a prompt with "#" in a tutorial, it's indicating a command that should be run as the super-user/root account, e.g., installing some software into a system-wide directory so it can be shared by all users.

Find the number of unique users on a shared system

We know that `w` will tell us the users logged in. Try it now on a system that has many users (i.e., not your laptop) and see the output. Likely there are dozens of users, so we'll connect the output of `w` to `head` using a pipe `|` so that we only see the first five lines:

```
[hpc:login2@~]$ w | head -5
 09:39:27 up 65 days, 20:05, 10 users,  load average: 0.72, 0.75, 0.78
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
kyclark  pts/2    gatekeeper.hpc.a 09:38    0.00s  0.05s  0.02s w
emsenhub pts/0    gatekeeper.hpc.a 04:05   14.00s  0.87s  0.87s -bash
joneska  pts/3    gatekeeper.hpc.a 08:25    1:12m  0.16s  0.12s vim results_x2r
```

Really we want to see the first five *users*, not the first five *lines* of output. To skip the first two lines of headers from `w`, we can pipe `w` into `awk` and tell it we only want to see output when the Number of Records (`NR`) is greater than 2:

```
[hpc:login2@~]$ w | awk 'NR>2' | head -5
kyclark  pts/2    gatekeeper.hpc.a 09:38    0.00s  0.07s  0.03s w
emsenhub pts/0    gatekeeper.hpc.a 04:05   26.00s  0.87s  0.87s -bash
joneska  pts/3    gatekeeper.hpc.a 08:25    1:13m  0.16s  0.12s vim results_x2r
shawtaro pts/4    gatekeeper.hpc.a 08:06   58:34   0.17s  0.17s -bash
darrenc  pts/5    gatekeeper.hpc.a 07:58   51:07   0.14s  0.07s qsub -I -N pipe
```

`awk` takes a PREDICATE and a CODE BLOCK (contained within curly brackets `{}`). Without a PREDICATE, `awk` prints the whole line. I only want to see the first column, so I can tell `awk` to `print` just column `$1`:

```
[hpc:login2@~]$ w | awk 'NR>2 {print $1}' | head -5
kyclark
emsenhub
joneska
shawtaro
darrenc
```

We can see that the some users like "joneska" are logged in multiple times:

```
[hpc:login2@~]$ w | awk 'NR>2 {print $1}'
kyclark
emsenhub
joneska
shawtaro
darrenc
guven
guven
guven
joneska
dmarrone
```

Let's `uniq` that output:

```
[hpc:login2@~]$ w | awk 'NR>2 {print $1}' | uniq
kyclark
emsenhub
joneska
shawtaro
darrenc
guven
joneska
dmarrone
```

Hmm, that's not right – "joneska" is listed twice, and that is not unique. Remember that `uniq` only works *on sorted input*? So let's sort those names first:

```
[hpc:login2@~]$ w | awk 'NR>2 {print $1}' | sort | uniq
darrenc
dmarrone
emsenhub
guven
joneska
kyclark
shawtaro
```

To count how many unique users are logged in, we can use the `wc` (word count) program with the `-l` (lines) flag to count just the lines from the previous command

```
[hpc:login2@~]$ w | awk 'NR>2 {print $1}' | sort | uniq | wc -l
7
```

So what you see is that we're connecting small, well-defined programs together using pipes to connect the "standard input" (STDIN) and "standard output (STDOUT) streams. There's a third basic file handle in Unix called"standard error" (STDERR) that we'll come across later. It's a way for programs to report problems without simply dying. You can redirect errors into a file like so:

```
$ program 2>err
$ program 1>out 2>err
```

The first example puts STDERR into a file called "err" and lets STDOUT print to the terminal. The second example captures STDOUT into a file called "out" while STDERR goes to "err."

NB: Sometimes a program will complain about things that you cannot fix, e.g., `find` may complain about file permissions that you don't care about. In those cases, you can redirect STDERR to a special filehandle called `/dev/null` where they are forgotten forever – kind of like the "memory hole" in 1984.

```
find / -name my-file.txt 2>/dev/null
```

Count "oo" words

On almost every Unix system, you can find `/usr/share/dict/words`. Let's use `grep` to find how many have the "oo" vowel combination. It's a long list, so I'll pipe it into "head" to see just the first five:

```
$ grep 'oo' /usr/share/dict/words | head -5
abloom
aboon
aboveproof
abrood
abrook
```

Yes, that works, so redirect those words into a file and count them. Notice the use of `!$` (bang-dollar) to reference the last argument of the previous line so that I don't have to type it again (really useful if it's a long path):

```
$ grep 'oo' /usr/share/dict/words > oo-words
$ wc -l !$
10460 oo-words
```

Let's count them directly out of `grep`:

```
$ grep 'oo' /usr/share/dict/words | wc -l
10460
```

Do any of those words additionally contain the "ow" sequence?

```
$ grep 'oo' /usr/share/dict/words | grep 'ow' | head -5
arrowroot
arrowwood
balloonflower
bloodflower
blowproof
```

How many are there?

```
$ grep 'oo' /usr/share/dict/words | grep 'ow' | wc -l
158
```

How many *do not* contain the "ow" sequence?

```
$ grep 'oo' /usr/share/dict/words | grep -v 'ow' | wc -l
10302
```

Do those numbers add up?

```
$ bc <<< 158+10302
10460
```

Excellent. Smithers, massage my brain.

Something with sequences

Now we will get some sequence data from the iMicrobe FTP site. Both `wget` and `ncftpget` will do the trick:

```
$ mkdir -p ~/contigs
$ cd !$
$ wget ftp://ftp.imicrobe.us/biosys-analytics/contigs/contigs.zip
```

> NB: How do we know we got the correct data? Go back and look at that FTP site, and you will see that there is a "contigs.zip.md5" file that we can `less` on the server to view the contents:

```
$ ncftp ftp://ftp.imicrobe.us/biosys-analytics/contigs
NcFTP 3.2.6 (Dec 04, 2016) by Mike Gleason (http://www.NcFTP.com/contact/).
Connecting to 150.135.44.10...
Welcome to the imicrobe.us repository
Logging in...
Login successful.
Logged in to ftp.imicrobe.us.
Current remote directory is /biosys-analytics/contigs.
ncftp /biosys-analytics/contigs > ls
contigs.zip        contigs.zip.md5
ncftp /biosys-analytics/contigs > cat contigs.zip.md5
1b7e58177edea28e6441843ddc3a68ab  contigs.zip
ncftp /biosys-analytics/contigs > exit
```

You can read up on MD5 (https://en.wikipedia.org/wiki/Md5sum) to understand that this is a signature of the file. If we calculate the MD5 of the file we dowloaded and it matches what we see on the server, then we can be sure that we have the exact file that is on the FTP site:

```
$ md5 contigs.zip
MD5 (contigs.zip) = 1b7e58177edea28e6441843ddc3a68ab
```

Yes, those two sums match. Note that sometimes the program is also named `md5sum`.

So, back to the exercise. Let's unpack the contigs:

```
$ unzip contigs.zip
Archive:  contigs.zip
  inflating: group12_contigs.fasta
  inflating: group20_contigs.fasta
  inflating: group24_contigs.fasta
$ rm contigs.zip
```

These files are in FASTA format (https://en.wikipedia.org/wiki/FASTA_format), which basically looks like this:

```
>MCHU - Calmodulin - Human, rabbit, bovine, rat, and chicken
ADQLTEEQIAEFKEAFSLFDKDGDGTITTKELGTVMRSLGQNPTEAELQDMINEVDADGNGTID
FPEFLTMMARKMKDTDSEEEIREAFRVFDKDGNGYISAAELRHVMTNLGEKLTDEEVDEMIREA
DIDGDGQVNYEEFVQMMTAK*
>gi|5524211|gb|AAD44166.1| cytochrome b [Elephas maximus maximus]
LCLYTHIGRNIYYGSYLYSETWNTGIMLLLITMATAFMGYVLPWGQMSFWGATVITNLFSAIPYIGTNLV
EWIWGGFSVDKATLNRFFAFHFILPFTMVALAGVHLTFLHETGSNNPLGLTSDSDKIPFHPYYTIKDFLG
LLILILLLLLLALLSPDMLGDPDNHMPADPLNTPLHIKPEWYFLFAYAILRSVPNKLGGVLALFLSIVIL
GLMPFLHTSKHRSMMLRPLSQALFWTLTMDLLTLTWIGSQPVEYPYTIIGQMASILYFSIILAFLPIAGX
IENY
```

Header lines start with ">", then the sequence follows. Sequences may be broken up over several lines of 50 or 80 characters, but it's just as common to see the sequences take only one (sometimes very long) line. Sequences may be nucleotides, proteins, very short DNA/RNA, longer contigs (shorter strands assembled into contiguous regions), or entire chromosomes or even genomes.

So, how many sequences are in "group12_contigs.fasta"? To answer, we just need to count how many times we see ">". We can do that with "grep":

```
$ grep > group12_contigs.fasta
Usage: grep [OPTION]... PATTERN [FILE]...
Try 'grep --help' for more information.
```

What is going on? Remember when we captured the "oo" words that we used the ">" symbol to tell Unix to *redirect* the output of `grep` into a file. We need to tell Unix that we mean a literal greater-than sign by placing it in single or double quotes or putting a backslash in front of it:

```
$ grep '>' group12_contigs.fasta
$ grep \> group12_contigs.fasta
```

You should actually see nothing because something quite insidious happened with that first "grep" statement – it overwrote our original "group12_contigs.fasta" with the result of "grep"ing for nothing, which is nothing:

```
$ ls -l group12_contigs.fasta
-rw-rw---- 1 kyclark staff 0 Aug 10 15:08 group12_contigs.fasta
```

Ugh, OK, I have to go back and `wget` the "contigs.zip" file to restore it. That's OK. Things like this happen all the time.

```
$ ls -lh group12_contigs.fasta
-rw-rw---- 1 kyclark staff 2.9M Aug 10 14:38 group12_contigs.fasta
```

Now that I have restored my data, I want to count how many greater-than signs in the file:

```
$ grep '>' group12_contigs.fasta | wc -l
132
```

Hey, I could see doing that often. Maybe we should make this into an "alias" (see above). The problem is that the "argument" to the function (the filename) is stuck in the middle of the chain of commands, so it would make it tricky to use an alias for this. We can create a bash function that we add to our `$HOME/.bashrc`:

```
function countseqs() {
  grep '>' $1 | wc -l
}
```

After you add this, remember to source this file to make it available:

```
$ source ~/.bashrc
$ countseqs group12_contigs.fasta
132
```

Same answer. Good. However, someone beat us to the punch. There is a powerful tool called "seqmagick" (https://github.com/fhcrc/seqmagick) that will do this (and much, much more). It's installed into the "hurwitzlab/bin" directory, or you can install it locally:

```
$ seqmagick info group12_contigs.fasta
name                  alignment    min_len   max_len   avg_len  num_seqs
group12_contigs.fasta FALSE           5136    116409  22974.30       132
```

Run "seqmagick -h" to see everything it can do.

Moving on, let's find how many contig IDs in "group12_contigs.fasta" contain the number "47":

```
$ grep 47 group12_contigs.fasta > group12_ids_with_47
[login3@~/work/sequences]$ cat !$
cat group12_ids_with_47
>Contig_247
>Contig_447
>Contig_476
>Contig_1947
```

```
>Contig_4764
>Contig_4767
>Contig_13471
```

Here we did a little "useless use of cat," but it's OK. We also could have used "less" to view the file. Here's another useless use of cat to copy a file:

```
$ cat group12_ids_with_47 > temp1_ids
```

Additionally, we want to copy the file again to make duplicates:

```
$ cp group12_ids_with_47 temp2_ids
```

How can we be sure these files are the same? Let's use "diff":

```
$ diff temp1_ids temp2_ids
```

You should see nothing, which is a case of "no news is good news." They don't differ in any way. We can verify this with "md5sum":

```
$ md5sum temp*
957390ab4c31db9500d148854f542eee  temp1_ids
957390ab4c31db9500d148854f542eee  temp2_ids
```

They are the same file. If there were even one character difference, they would generate different hashes.

Now we will create a file with duplicate IDs:

```
$ cat temp1_ids temp2_ids > duplicate_ids
```

Check contents of "duplicate_ids" using "less" or "cat." Now grab all of the contigs IDs from "group20_contigs.fasta" that contain the number "51." Concatenate the new IDs to the duplicate_ids file in a file called "multiple_ids":

```
$ cp duplicate_ids multiple_ids
$ grep 51 group20_contigs.fasta >> !$
grep 51 group20_contigs.fasta >> multiple_ids
```

Notice the ">>" arrows to indicate that we are *appending* to the existing "multiple_ids" file.

Remove the existing "temp" files using a "*" wildcard:

```
$ rm temp*
```

Now let's explore more of what "sort" and "uniq" can do for us. We want to find which IDs are unique and which are duplicated. If we read the manpage ("man uniq"), we see that there are "-d" and "-u" flags for doing just that. However, we've already seen that input to "uniq" needs to be sorted, so we need to remember to do that:

```
$ sort multiple_ids | uniq -d > temp1_ids
$ sort multiple_ids | uniq -u > temp2_ids
$ diff temp*
```

```
1,7c1,11
< >Contig_13471
< >Contig_1947
< >Contig_247
< >Contig_447
< >Contig_476
< >Contig_4764
< >Contig_4767
---
> >Contig_10051
> >Contig_1651
> >Contig_4851
> >Contig_5141
> >Contig_5143
> >Contig_5164
> >Contig_5170
> >Contig_5188
> >Contig_6351
> >Contig_9651
> >Contig_9851
```

Let's remove our temp files again and make a "clean_ids" file:

```
$ rm temp*
$ sort multiple_ids | uniq > clean_ids
$ wc -l multiple_ids clean_ids
 25 multiple_ids
 18 clean_ids
 43 total
```

We can use "sed" to alter the IDs. The "s//" command say to "substitute" the first thing with the second thing, e.g., to replace all occurences of "foo" with "bar", use "s/foo/bar" (http://stackoverflow.com/questions/4868904/what-is-the-origin-of-foo-and-bar).

```
$ sed 's/C/c/' clean_ids
$ sed 's/_/./' clean_ids
$ sed 's/>//' clean_ids > newclean_ids
```

That last one removes the FASTA file artifact that identifies the beginning of an ID but is not part of the ID. We can use this with "seqmagick" now to extract those sequences and find out how many were found:

```
$ seqmagick convert --include-from-file newclean_ids group12_contigs.fasta newgroup12_contig
$ seqmagick info !$
seqmagick info newgroup12_contigs.fasta
name                      alignment    min_len   max_len   avg_len  num_seqs
newgroup12_contigs.fasta FALSE           5587     30751  16768.14         7
```

We can get stats on all our files:

```
$ seqmagick info *fasta > fasta-info
$ cat !$
name                      alignment    min_len   max_len   avg_len  num_seqs
group12_contigs.fasta     FALSE           5136    116409  22974.30       132
group20_contigs.fasta     FALSE           5029     22601   7624.38       203
group24_contigs.fasta     FALSE           5024     81329  12115.70       139
newgroup12_contigs.fasta FALSE            5587     30751  16768.14         7
```

We can use "cut" to view various columns:

```
$ cut -f 2 fasta-info
$ cut -f 2,4 fasta-info
$ cut -f 2-4 fasta-info
```

But it does not line up very nicely. We can use "column" to fix this:

```
$ cut -f 2-4 fasta-info | column -t
alignment  min_len  max_len
FALSE      5136     116409
FALSE      5029     22601
FALSE      5024     81329
FALSE      5587     30751
```

Gapminder

Do the following:

```
$ git clone https://github.com/kyclark/metagenomics-book
$ cd metagenomics-book/problems/gapminder/data
```

How many files are in the "data" directory?

```
$ ls | wc -l
```

How many lines are in each/all of the files?

```
$ wc -l *
```

You can use `cat` to spew at the entire contents of a file into your shell, but if you'd just like to see the top of a file, you can use:

```
$ head Trinidad_and_Tobago.cc.txt
```

If you only want to see 5 lines, use `-n 5` or `-5` .

For our exercise, we'd like to combine all the files into one file we can analyze. That's easy enough with:

```
$ cat * > all.txt
```

Let's use `head` to look at the top of file:

```
$ head -5 all.txt
Afghanistan    1997    22227415    Asia    41.763    635.341351
Afghanistan    2002    25268405    Asia    42.129    726.7340548
Afghanistan    2007    31889923    Asia    43.828    974.5803384
Afghanistan    1952    8425333    Asia    28.801    779.4453145
Afghanistan    1957    9240934    Asia    30.332    820.8530296
```

Hmm, there are no column headers. Let's fix that. There's one file that's pretty different in content (it has only one line) and name ("country.cc.txt"):

```
$ cat country.cc.txt
country    year    pop    continent    lifeExp    gdpPercap
```

Those are the headers that you can combine to all the other files to get named columns, something very important if you want to look at the data in Excel and R/Python data frames.

```
$ rm all.txt
$ mv country.cc.txt headers
$ cat headers *.txt > all.txt
$ head -5 all.txt | column -t
country      year  pop       continent  lifeExp  gdpPercap
Afghanistan  1997  22227415  Asia       41.763   635.341351
Afghanistan  2002  25268405  Asia       42.129   726.7340548
Afghanistan  2007  31889923  Asia       43.828   974.5803384
Afghanistan  1952  8425333   Asia       28.801   779.4453145
```

Yes, that looks much better. Double-check that the number of lines in the `all.txt` match the number of lines of input:

```
$ wc -l *.cc.txt headers
$ wc -l all.txt
```

How many observations do we have for 1952?

```
$ grep 1952 all.txt | wc -l
$ cut -f 2 *.cc.txt | grep 1952 | wc -l
```

Those numbers aren't the same! Why is that?

```
$ grep 1952 all.txt | cut -f 2 | sort | uniq -c
 142 1952
   1 1982
   1 1987
$ grep 1952 all.txt | grep 198[27]
Lebanon    1982    3086876    Asia    66.983    7640.519521
Mozambique    1987    12891952    Africa    42.861    389.8761846
```

How many observations for every year?

```
$ cut -f 2 *.cc.txt | sort | uniq -c
```

How many observations are present for Africa?

```
$ grep Africa all.txt | wc -l
```

How many for each continent?

```
$ cut -f 4 *.cc.txt | sort | uniq -c
```

What was the world population in 1952? As we've seen, just using `grep 1952` is not sufficient. We want to take the 3rd column if the 2nd column is equal to "1952." `awk` will let us do just that. Normally `awk` will split on whitespace, so we need to use `-F"\t"` to tell it to split on the tab (`\t`) character. Use `man awk` to learn more.

```
$ awk -F"\t" '$2 == "1952" { print $3 }' *.cc.txt
```

I'll bet you didn't notice that one of those numbers was in scientific notation. That's going to cause a problem. Here it is:

```
$ awk -F"\t" '$2 == "1952" { print $3 }' *.cc.txt | grep [a-z]
3.72e+08
```

We have to throw in a `grep -v` to get rid of that (the `-v` reverses the match), then use the `paste` command is used to put a "+" in between all the numbers:

```
$ awk -F"\t" '$2 == "1952" { print $3 }' *.cc.txt | grep -v [a-z] | paste -sd+ -
```

and then we pipe that to the `bc` calculator:

```
$ awk -F"\t" '$2 == "1952" { print $3 }' *.cc.txt | grep -v [a-z] | paste -sd+ - | bc
2034957150.999989
```

It bothers me that it's not an integer, so I'm going to use `printf` in the `awk` command to trim that:

```
$ awk -F"\t" '$2 == "1952" { printf "%d\n", $3 }' *.cc.txt | grep -v [a-z] | paste -sd+ - |
2406957150
```

How did population change over the years? Let's put a list of the unique years into a file called "years" and then `cat` over that to run the above for each year:

```
$ cut -f 2 *.txt | sort | uniq > years
$ for year in `cat years`; do echo -n $year ": " && awk -F"\t" "\$2 == $year { printf \"%d\n
1952 : 2406957150
1957 : 2664404580
1962 : 2899782974
1967 : 3217478384
1972 : 3576977158
1977 : 3930045807
1982 : 4289436840
1987 : 4691477418
1992 : 5110710260
1997 : 5515204472
```

```
2002 : 5886977579
2007 : 6251013179
```

That's kind of useful! Here's how I might put that into a script:

```
$ cat pop-years.sh
#!/bin/bash

set -u

YEARS="years"

cut -f 2 ./*.cc.txt | sort | uniq > "$YEARS"
NUM=$(wc -l $YEARS | awk '{print $1}')

if [[ "$NUM" -lt 1 ]]; then
  echo "No years ($NUM)!"
  exit 1
fi

while read -r YEAR; do
    echo -n "$YEAR: "
    awk -F"\t" "\$2 == $YEAR { printf \"%d\\n\", \$3 }" ./*.cc.txt | grep -v "[a-z]" | paste
done < "$YEARS"
$ ./pop-years.sh
1952: 2406957150
1957: 2664404580
1962: 2899782974
1967: 3217478384
1972: 3576977158
1977: 3930045807
1982: 4289436840
1987: 4691477418
1992: 5110710260
1997: 5515204472
2002: 5886977579
2007: 6251013179
```

How has life expectancy changed over the years? For this we'll need to write a little Python program. I'll `cat` the program so you can see it. You can type this in with `nano` and then do `chmod +x avg.py` to make it executable (or use the one I added):

```
$ cat avg.py
#!/usr/bin/env python3

import sys
```

```
args = list(map(float, sys.argv[1:]))
print(str(sum(args) // len(args)))
$ for year in `cat years`; do echo -n "$year: " && grep $year *.txt | cut -f 5 | xargs ./avg
1952: 49.0
1957: 51.0
1962: 53.0
1967: 55.0
1972: 57.0
1977: 59.0
1982: 61.0
1987: 63.0
1992: 64.0
1997: 65.0
2002: 65.0
2007: 66.0
```

How many observations where the life expectancy ("lifeExp," field #5) is greater than 40? For this, let's use the `awk` tool.

```
$ awk -F"\t" '$5 > 40' all.txt | wc -l
```

How many of those are from Africa? We can either use `cut` to get the 4th field or ask `awk` to print the 4th field for us:

```
$ awk -F"\t" '$5 > 40' all.txt | cut -f 4 | grep Africa | wc -l
$ awk -F"\t" '$5 > 40 {print $4}' all.txt | grep Africa | wc -l
```

How many countries had a life expectancy greater than 70, grouped by year?

```
$ awk -F"\t" '$5 > 70 { print $2 }' *.cc.txt | sort | uniq -c
   5 1952
   9 1957
  16 1962
  25 1967
  30 1972
  38 1977
  44 1982
  49 1987
  54 1992
  65 1997
  75 2002
  83 2007
```

How could we add continent to this?

```
$ awk -F"\t" '$5 > 70 { print $2 ":" $4 }' *.cc.txt | sort | uniq -c
```

As you look at the data and want to ask more complicated questions like how does `gdpPercap` affect `lifeExp`, you'll find you need more advanced tools like

Python or R. Now that the data has been collated and the columns named, that will be much easier.

What if we want to add headers to each of the files?

```
$ mkdir wheaders
$ for file in *.txt; do cat headers $file > wheaders/$file; done
$ wc -l wheaders/* | head -5
      13 wheaders/Afghanistan.cc.txt
      13 wheaders/Albania.cc.txt
      13 wheaders/Algeria.cc.txt
      13 wheaders/Angola.cc.txt
      13 wheaders/Argentina.cc.txt
$ head wheaders/Vietnam.cc.txt
country    year    pop    continent    lifeExp    gdpPercap
Vietnam    1952    26246839    Asia    40.412    605.0664917
Vietnam    1957    28998543    Asia    42.887    676.2854478
Vietnam    1962    33796140    Asia    45.363    772.0491602
Vietnam    1967    39463910    Asia    47.838    637.1232887
Vietnam    1972    44655014    Asia    50.254    699.5016441
Vietnam    1977    50533506    Asia    55.764    713.5371196
Vietnam    1982    56142181    Asia    58.816    707.2357863
Vietnam    1987    62826491    Asia    62.82    820.7994449
Vietnam    1992    69940728    Asia    67.662    989.0231487
```