NCGAS Spring Workshop Day 2: Annotation Demo

Part I: Tool Review

Before we get to downstream analyses, we're going to do a quick introduction/review to some really useful UNIX tools that you will need. All of these can use "regular expression" or "regex" - a type of pattern matching. We will not cover this today, as we will only be searching for discrete terms, but it is a skill to look into:

Tool 1: grep - "Grab Regular Expression and Print"

The purpose of **grep** is to print lines matching a search term or pattern. For example, if I was looking for all lines that match "Gene10001", I could use the following command:

```
grep "Gene10001" file.txt
```

The format is pretty simple - the name of the program, the search term, and the search target. However, there are two options we will be using:

```
grep -v "Gene10001" file.txt  #returns all NON-matching lines
grep -f file.list file.txt    #returns all matches to a list of search terms in a file (i.e. file.list)
```

Tool 2: sed - Stream EDitor

The purpose of **sed** is to allow you to edit lines of a file without having to do it manually. 80% of the usage of **sed** is to replace one term with another. For example:

```
sed 's/this/that/g' file.txt
```

This command will replace all instances of "this" with "that" in file.txt. Note that this by default outputs your information to the command line. In order to do replacements "in place", there is a useful option:

```
sed -i 's/this/that/g' file.txt
```

Notice how nothing is output? That is because the replacements are done within the file.

awk - named for it's authors

awk is an entire language, and can be a bit intimidating to learn. However, we are going to use an extremely simple aspect of awk - printing by column. **awk** commands look like this:

```
awk '{print $1}' file.txt
```

The command always starts with the name of the tool and is followed by '{command}' and then the target. In this case, print $1 prints the first column. $2 would print the second column, $3 would print the third, and so on. The only special case is $0 prints the entire line. You can also use this to combine columns if you want, using the same general format:

```
awk '{print $1"_"$2}' file.txt    #will combine column 1 and 2, separated by a "_"
```

Review:

grep - grab reg ex and print:	grep "search term" file
	-v reverse
	-f from file
sed - stream editor:	sed 's/this/that/g' file
	-i in place
awk - column manipulation:	awk '{print $1}' file
	$0 full line
	$1 first column
	can do $1"_"$2 to combine columns

Part II: Annotation

Okay, now back to our task at hand - Annotation. Inside the final_assemblies directory, you will find an annotations directory. We currently use Trinotate (https://github.com/Trinotate/Trinotate.github.io/wiki) to annotate our cleaned up assemblies.

Trinotate isn't really doing much itself - the workflow requires that you run blast to search for homology against swissprot, domain information in pfam, signal proteins with signalP, transmembrane protein with tmhmm. Trinotate will also let you add custom blast results, such as if you have a closely related organism with great annotation (model systems like fly, arabidopsis, mouse, etc). See the website for more information!

The Trinotate script itself does do two major things:

- pulls eggNOG, GO, and KEGG terms from the blast homology searches
- glues everything together in a single usable table.

Trinotate requires sequence processing by TransDecoder, which is doesn't like our SOAP output as it includes Ns. Also, the headers are a bit weird coming out of EviGene (useful, but weird), so we have to clean them up. So our first step is to use **RunAnnotation.sh**, which allows us to make a better name for the transcripts, as well as make them compatible with Trinotate without the use of TransDecoder.

NOTE: **This is a breakable step in the pipeline - and we are working on making this more stable.** If you have issues, please contact us. But we are working on making the output compatable with transdecoder.

Next, we run **RunTrinotate.sh** - which will run and combine all the above analyses. The input files are "transcripts.main.fa" and "transcripts.reformatted.aa" by default, and the output is called Report.xls by default.

The Report is easily used in Excel, however, because it has spaces... we have to replace all those spaces with "_"s. We can do this with **sed**!

```
sed 's/\ /_/g' Report.xls > Report_wospaces.xls #or
sed -i 's/\ /_/g' Report.xls
```

Part III: Customizing with Tools

If we want to add more information to the table, now that only columns are separated by spaces, we can use **awk** and **join**. For instance:

```
awk '{print $1, $2"_"$3}' blastp.out | sort > tmp
```

tmp will now hold some quality information for the blastp. Let's merge that into the table... but first we will need to learn about **join**.

join allows you to merge tables together by some column that they share. The command looks like this:

```
join -1 5 -2 1 file1.txt file2.txt
```

where -1 5 indicates that the column to merge by in file 1 is column 5, and -2 1 indicates that the column to merge by in file 2 is column 1. Column 5 of file 1 should have the same content as column 1 in file 2.

There are two major considerations when using join:
1) join assumes sorting
2) are there missing entries in either of the files?

The first consideration is easy enough to work around - you can sort files first with **sort**:

```
sort file.txt > file.sorted
```

or if you want to **sort** by just column 1-3:

```
sort -k1,3 file.txt > file.sorted
```

or by column 1 alone:

```
sort -k1,1 file.txt > file.sorted
```

You can **sort** inside the join command, to prevent you from having an extra file each time. I end up doing this almost every time, because I forget to sort first ^_^.

```
join -1 5 -2 1 <(sort file1.txt) <(sort file2.txt)
```

The **<(...)** creates a "sub process" that happens before the main process. So file1.txt is sorted before being fed to **join**, as is file2.txt. NOTE: be care of this for very large files!

To take care of the second consideration of missing data, you can define which file to keep all entries from using the -a option. So if you want to make sure you keep all of the entries from file 1, even if file 2 is missing them, you can do the following:

```
join -a 2 -1 5 -2 1 <(sort file1.txt) <(sort file2.txt)
```

Let's give this a whirl on our data:

```
join -a 1 -1 5 -2 1 <(sort -k5,5 Report_wospace.xls) tmp | less -S
        #-a keep all entries from "1"
        #-1 what column to match against in "1"
        #-2 what column to match against in "2"
        #<() subprocess
```

NOTE: this will reorder your columns

Now we have an identity associated with the output of our blastp results!

That should get you working with the annotation information pretty well. Next we will talk specifically about KEGG analysis and visualizing pathways, using this annotation information.