Lecture 03 - Finding things in Unix

grep

- powerful search function
- searches for files or inside files?
- considers lines or matches?
- when would this be useful?

`grep` application

- building a simple gene finder

Copy and paste contents of *DNA.txt* from Lecture 3 directory on Sakai into `nano` and save as *DNA.txt*

Challenge: Use Unix to count the number of start codons (ATG), and therefore potential genes, in your DNA sequence

grep application - A simple gene finder

```
cat DNA.txt | wc
cat DNA.txt | grep -c "ATG"
```

grep application - A simple gene finder

```
cat DNA.txt | wc
cat DNA.txt | grep -c "ATG"
cat DNA.txt | grep -o "ATG"
```

grep application - A simple gene finder

```
cat DNA.txt | wc
cat DNA.txt | grep -c "ATG"
cat DNA.txt | grep -o "ATG"
cat DNA.txt | grep -o "ATG" | wc -l
```

grep application - A simple gene finder

```
cat DNA.txt | wc
```

```
cat DNA.txt | tr -d "\n" | wc
```

grep application - A simple gene finder

```
cat DNA.txt | tr -d "\n" | wc
cat DNA.txt | tr -d "\n" | grep -o "ATG" | wc -l
```

`find`

- powerful search function
- searches for files or inside files?
- when would this be useful?

`$( )` extends the utility of `find`

Challenge: Use Unix to count the number of lines in each of the *.pdb* files in the `data-shell/molecules` directory.

sed

`grep` is to find,

as `sed` is to find and replace

sed

An example with *DNA.txt*

Imagine we wanted to visualize all of our matches to the start codon *ATG*. We can use `sed` to do this.

```
cat DNA.txt | sed 's/ATG/---/g' | less -S
```

sed syntax

The sed utility reads the specified files, or the *standard input* if no files are specified, modifying the input as specified by a list of **commands**. The input is then written to the *standard output*.

Each line of a file, except the newline character is copied into a **pattern space** where it is acted on by sed commands.

sed commands

- `sed 's/regular expression/replacement/flags'`
  - flags: N - replace the Nth occurrence, g - replace all occurrences, not just first
  - when using complex regular expressions, back referencing allows one to use the matching string

- `sed 'y/string1/string2/'`
  - a multi-version of `tr`

`sed` application

Challenge: use `sed`, along with `tr` and `rev`, to create the reverse complement of the sequence contained in *DNA.txt*

function review

`grep`

`find`

`$( )`

`sed`