Live coding demo - week 3

Demonstration 1: `ln` and `ln -s`

Note, the below example takes advantage of brace expansion (click me for the documentation page) to generate dummy data quickly. Brace expansion is handy in any situation where you want to generate lots of strings according to a pattern.

```
(base) $ mkdir dir{1,2}
(base) $ ls
dir1  dir2
(base) $ echo "file a content" > dir1/a
(base) $ echo "different stuff in b" > dir1/b
(base) $ ls -ail
total 16
29364 drwxr-xr-x 4 alan alan 4096 Aug 29 09:18 .
29358 drwxr-xr-x 5 alan alan 4096 Aug 28 18:05 ..
29373 drwxr-xr-x 2 alan alan 4096 Aug 29 09:20 dir1
46180 drwxr-xr-x 2 alan alan 4096 Aug 29 09:18 dir2
(base) $ ls -ail dir1/
total 16
29373 drwxr-xr-x 2 alan alan 4096 Aug 29 09:20 .
29364 drwxr-xr-x 4 alan alan 4096 Aug 29 09:18 ..
46279 -rw-r--r-- 1 alan alan   15 Aug 29 09:20 a
46280 -rw-r--r-- 1 alan alan   21 Aug 29 09:20 b
(base) $ ln dir1/a dir2/c
(base) $ # symlinks store a path so the first arg here
(base) $ # is path relative to destination (dir2/d)
(base) $ ln -s ../dir1/a dir2/d
(base) $ ls -ail dir2/
total 12
46180 drwxr-xr-x 2 alan alan 4096 Aug 29 09:56 .
29364 drwxr-xr-x 4 alan alan 4096 Aug 29 09:18 ..
46279 -rw-r--r-- 2 alan alan   15 Aug 29 09:20 c
46281 lrwxrwxrwx 1 alan alan    6 Aug 29 09:56 d -> ../dir1/a
(base) $ # Note that the link count of dir1/a has incremented once
(base) $ # Link count reflects the number of hard links to a file
(base) $ ls -ail dir1/a
443360 -rw-r--r-- 2 alan alan 15 Aug 29 09:20 dir1/a
(base) $ # dir1 and dir2 both have link counts of 2. Where do you think the
(base) $ # two links are?
(base) $ # Hint: they are shown in the above outputs and we didn't make them
(base) $ # The original file and both links behave the same then opened
(base) $ cat dir1/a
file a content
(base) $ cat dir2/c
file a content
(base) $ cat dir2/d
file a content
(base) $ # hard links point to the same inode
```

```
(base) $ # when you rm a file, if other links point to the same inode,
(base) $ # only the filename is deleted, but all data kept
(base) $ # rm is also sometimes called "unlink" for this reason
(base) $ # sotflinks point to a path.
(base) $ # If the path is changed, the symlink stops working
(base) $ rm dir1/a
(base) $ cat dir2/c
file a content
(base) $ cat dir2/d
cat: dir2/d: No such file or directory
(base) $ # symlinks are vulnerable to their targets being replaced
(base) $ echo "new a content" > dir1/a
(base) $ cat dir2/d
new a content
(base) $ # hard links are not
(base) $ # once a hard linked file is deleted, new files with the
(base) $ # same name will point to a new inode and are therefore
(base) $ # not hard linked
(base) $ cat dir2/c
file a content
(base) $ ls -i dir1/a
46282 dir1/a
(base) $ # however, changes to any hard linked files are
(base) $ # reflected in all other hard linked files
(base) $ rm dir1/a
(base) $ ln dir2/c dir1/a
(base) $ ls -i dir1/a dir2/c
46279 dir1/a  46279 dir2/c
(base) $ cat dir1/a
file a content
(base) $ echo "plus new stuff" >> dir1/a
(base) $ cat dir2/c
file a content
plus new stuff
(base) $ echo "replacement text" > dir2/c
(base) $ cat dir1/a
replacement text
```

Demonstration 2: `grep -E`

We'll start by echoing some strings and demonstrating that portions can be matched differently

```
(base) $ # with -o, grep only returns the matched portion
(base) $ echo "123abc123" | grep -Eo "[0-9][abc]"
3a
(base) $ # If there are multiple matches they are returned on separate lines
(base) $ echo "123abc123" | grep -Eo "[0-9]?[abc]"
3a
b
c
(base) $ echo "123abc123" | grep -Eo "[0-9]*[abc]"
123a
```

```
b
c
(base) $ echo "123abc123" | grep -Eo "[0-9]+[abc]"
123a
(base) $ # Character classes are equivalent to ranges defining the same characters
(base) $ echo "123abc123" | grep -Eo "[[:digit:]]+[abc]"
123a
```

Match groups can also be used to control what your pattern matches with grep

```
(base) $ echo "123abc123" | grep -Eo "([0-9]).*(\1)"
123abc1
(base) $ # Note that you might have expected match group 1
(base) $ # to also match "23abc12"
(base) $ # however, the first match consumed those characters so no other
(base) $ # matches were possible. Matches can't overlap
(base) $ # Add more characters to allow non-overlapping matches and
(base) $ # you get more than one
(base) $ echo "123abc123" | grep -Eo "([0-9]).{5}(\1)"
123abc1
(base) $ echo "123abc123abc123" | grep -Eo "([0-9]).{5}(\1)"
123abc1
23abc12
(base) $ # You can't use .* here as * is greedy. We'll discuss that later
```

`grep` can also be used with regexes to extract information from files. The below example uses the "Cordyceps_ncbi.xml" file in the "week_3_example_data.tar.gz" tarball.

```
(base) $ # There are a bunch of genbank accession numbers in this file.
(base) $ # These accessions all have a common format which can be
(base) $ # described with a regex
(base) $ grep Genbank Cordyceps_ncbi.xml | head
        <Genbank>GCA_000225605.1</Genbank>
        <Genbank>GCA_002968875.1</Genbank>
        <Genbank>GCA_001636725.1</Genbank>
        <Genbank>GCA_028828415.1</Genbank>
        <Genbank>GCA_025583755.1</Genbank>
        <Genbank>GCA_030779685.1</Genbank>
        <Genbank>GCA_006981975.1</Genbank>
        <Genbank>GCA_003025255.1</Genbank>
        <Genbank>GCA_002591385.1</Genbank>
        <Genbank>GCA_008080495.1</Genbank>
(base) $ grep -E "GCA_[[:digit:]]+.[[:digit:]]" Cordyceps_ncbi.xml | head -2
        <Genbank>GCA_000225605.1</Genbank>

<FtpPath_GenBank>ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/000/225/605/GCA_0002256
05.1_CmilitarisCM01_v01</FtpPath_GenBank>
(base) $ # The accessions occur more than once and we don't want the surrounding
text
(base) $ #  we can use the following to just get unique accessions
(base) $ grep -Eo "GCA_[[:digit:]]+.[[:digit:]]" Cordyceps_ncbi.xml | sort -u |
```

```
head
GCA_000225605.1
GCA_000733625.1
GCA_001636725.1
GCA_001644785.1
GCA_002591385.1
GCA_002968875.1
GCA_003025255.1
GCA_003025275.1
GCA_003025305.1
GCA_003332165.1
```

Demonstration 3: `find`

`find` is a powerful tool to search for files matching a set of criteria. It supports searching for patterns describing filenames using either glob or extended regex (as well as some different flavors of regex).

The following examples will illustrate how the `find` command can be used and

```
(base) $ # The example data for this week contains two directries
(base) $ # containing sequences and features
(base) $ # for samples with lots of different accession numbers. E.g.,
(base) $ ls sequences/ | column -c 120 | head -7
Anolis_carolinensis.fna         GCA_030781145.1.fna             SRR7813583.fna
Arabidopsis_thaliana.fna        GCA_030781225.1.fna             SRR7829179.fna
Caenorhabditis_elegans.fna      GCA_030781285.1.fna             SRR8182777.fna
Dictyostelium_discoideum.fna    GCA_030781325.1.fna             SRR8198781.fna
Drosophila_melanogaster.fna     GCA_030781485.1.fna             SRR8198829.fna
ERR1158330.fna                  GCA_030781505.1.fna             SRR8198901.fna
ERR1195556.fna                  GCA_030781525.1.fna             SRR8198968.fna
(base) $ # find can be used with a regex to isolate just the files
(base) $ # we want to analyze
(base) $ find . -regextype egrep -regex ".*/[[:alpha:]]+_[[:alpha:]]+.fna"
./sequences/Pseudomonas_aeruginosa.fna
./sequences/Xenopus_laevis.fna
./sequences/Saccharomyces_cerevisiae.fna
./sequences/Caenorhabditis_elegans.fna
./sequences/Dictyostelium_discoideum.fna
./sequences/Mus_musculus.fna
./sequences/Drosophila_melanogaster.fna
./sequences/Rat_rattus.fna
./sequences/Escherichia_coli.fna
./sequences/Gorilla_gorilla.fna
./sequences/Gallus_gallus.fna
./sequences/Anolis_carolinensis.fna
./sequences/Arabidopsis_thaliana.fna
(base) $ find . -regextype egrep -regex ".*/.*_[0-9]+.*.fna" | head
./sequences/GCA_000225605.1.fna
./sequences/GCA_000733625.1.fna
./sequences/GCA_001636725.1.fna
./sequences/GCA_001644785.1.fna
./sequences/GCA_002591385.1.fna
```

```
./sequences/GCA_002968875.1.fna
./sequences/GCA_003025255.1.fna
./sequences/GCA_003025275.1.fna
./sequences/GCA_003025305.1.fna
./sequences/GCA_003332165.1.fna
(base) $ # In addition, find can execute one or more commands on matched files
(base) $ # within the command being executed, the matched file is referred
(base) $ # to using {}
(base) $ # Note that each command must be ended with \; or an otherwise
(base) $ # escaped ";" character
(base) $ # In this example I am directing stderr to a null file to supress
(base) $ # errors from piping to head
(base) $ find . -regextype egrep -regex ".*/.*_[0-9]+.*.fna" -exec echo {} \; -exec
head {} \; 2>/dev/null  | head -9
./sequences/GCA_000225605.1.fna
>GCA_000225605.1_contig_1
ATCGCGTACGTCAAGACTACATCGAT
./sequences/GCA_000733625.1.fna
>GCA_000733625.1_contig_1
ATCGCGTACGTCAAGACTACATCGAT
./sequences/GCA_001636725.1.fna
>GCA_001636725.1_contig_1
ATCGCGTACGTCAAGACTACATCGAT
```

Demonstration 4: `sed -[rE]`

`sed` can be used to replace text using regexes. This is expecially powerful when using match groups to retain strings of interest from the input text.

The below examples use the model_orgs.txt file included in this week's example data

```
(base) $ # This is the command I showed in sublime text to illustrate the
(base) $ # power of regex.
(base) $ # It matches the whole line, but uses match groups to catch the
(base) $ # genus and species
(base) $ # The match groups are then used in a replacement operation
(base) $ sed -E 's/.*([[:upper:]][[:lower:]]+) ([[:lower:]]+).*/Genus: \1\tSpecies:
\2/g' model_orgs.txt | head
Genus: Anolis   Species: carolinensis
Genus: Caenorhabditis   Species: elegans
Genus: Arabidopsis      Species: thaliana
Genus: Pseudomonas      Species: aeruginosa
Genus: Escherichia      Species: coli
Genus: Mus      Species: musculus
Genus: Rat      Species: rattus
Genus: Gorilla  Species: gorilla
Genus: Xenopus  Species: laevis
Genus: Drosophila       Species: melanogaster
(base) $ # Again, character classes are equivalent to bracket expressions
(base) $ sed -E 's/.*([A-Z][a-z]+) ([a-z]+).*/Genus: \1\tSpecies: \2/g'
model_orgs.txt | head
Genus: Anolis   Species: carolinensis
```

```
Genus: Caenorhabditis   Species: elegans
Genus: Arabidopsis      Species: thaliana
Genus: Pseudomonas      Species: aeruginosa
Genus: Escherichia      Species: coli
Genus: Mus      Species: musculus
Genus: Rat      Species: rattus
Genus: Gorilla  Species: gorilla
Genus: Xenopus  Species: laevis
Genus: Drosophila       Species: melanogaster
```

Demonstration 5: `grep -P`

`grep` is one of the few unix tools to support perl regex. Luckily it's also one of the most useful unix tools to support it!

Perl regex supports a few useful behaviours that are not supported in the GNU ERE. Here we will illustrate three of the most useful: lazy quantifiers, and lookarounds.

For this demo we will `echo` different test strings and apply different regexes using either the `grep -E` GNU ERE mode or the `grep -P` perl mode. We will use the `-o` option to only output the matched portions of the input string. The default behaviour of `grep` is otherwise to return the whole line if a match was found anywhere in it.

```
(base) $ # grep -E only has greedy quantifiers so the .* matches
(base) $ # as much as possible
(base) $ echo "_____123_____456_____" | grep -Eo ".*[0-9]+"
_____123_____456
(base) $ # perl quantifiers are greedy by default
(base) $ echo "_____123_____456_____" | grep -Po ".*[0-9]+"
_____123_____456
(base) $ # Adding a ? after a quantifier makes it lazy
(base) $ # (match as little as possible)
(base) $ echo "_____123_____456_____" | grep -Po ".*?[0-9]+"
_____123
_____456
(base) $ echo "_____123_____456_____" | grep -Po ".*?[0-9]+?"
_____1
2
3
_____4
5
6
(base) $ # What do you think would be matched by the regex ".*?[0-9]*?"
```