Introductory Linux Tutorial for Life Sciences

Session 4: Text processing

Teacher: Marcel Martin

Command: `echo`

- `echo` prints its arguments, separated by space (to stdout):

```
$ echo Hello, world
Hello, world
```

Command: `echo`

- `echo` prints its arguments, separated by space (to stdout):

```
$ echo Hello, world
Hello, world
```

- Using redirection, it can be used to create (short) files:

```
$ echo Hello, world > file.txt
$ cat file.txt
Hello, world
```

Command: `echo`

- `echo` prints its arguments, separated by space (to stdout):

```
$ echo Hello, world
Hello, world
```

- Using redirection, it can be used to create (short) files:

```
$ echo Hello, world > file.txt
$ cat file.txt
Hello, world
```

- Why does this happen?

```
$ echo Hello,    world
Hello, world
```

Command: `echo`

- `echo` prints its arguments, separated by space (to stdout):

```
$ echo Hello, world
Hello, world
```

- Using redirection, it can be used to create (short) files:

```
$ echo Hello, world > file.txt
$ cat file.txt
Hello, world
```

- Why does this happen?

```
$ echo Hello,    world
Hello, world
```

- The two arguments are `Hello,` and `world`.
  Fix with quotation marks:

```
$ echo "Hello,   world"
Hello,   world
```

Command: `tr` – "translate" characters

- Usage 1: `tr <SET1> <SET2>`
- Replace each character in SET1 with the corresponding character in SET2

```
$ echo 'Linux is a lot of fun' | tr "a-z" "A-Z"
LINUX IS A LOT OF FUN
```

Delete characters with `tr`

- Usage 2: `tr -d <SET1>` – delete characters

```
$ echo 'earth vs moon' | tr -d hnt
ear vs moo
```

```
$ echo 'my PIN is 4333434' | tr -d "[:digit:]"
my PIN is
```

basename

- `basename` – strips any 'path' name components from a filename
- Usage: `basename <filename> [<suffix>]`

```
$ basename /home/duck/myProject/myScript.sh
myScript.sh
```

- It can also strip a (known) suffix from a filename:

```
$ basename /home/duck/myProject/myScript.sh .sh
myScript
```

Command: `sed` – edit files

- `sed` is a "stream editor" for filtering and transforming text
- It processes the input line-by-line
- Usage:

```
sed [options] script [file]
```

Command: `sed` – edit files

- `sed` is a "stream editor" for filtering and transforming text
- It processes the input line-by-line
- Usage:

```
sed [options] script [file]
```

- The `script` is:
  `[<range>]<command>[<arguments>]`
- The `command` is just a single letter. We will look at two:
  - `d` – delete line
  - `s` – search and replace

Command: sed – edit files

- `sed` is a "stream editor" for filtering and transforming text
- It processes the input line-by-line
- Usage:

```
sed [options] script [file]
```

- The `script` is:
  `[<range>]<command>[<arguments>]`
- The `command` is just a single letter. We will look at two:
  - `d` – delete line
  - `s` – search and replace
- The `range` specifies which lines the command should work on. Useful ranges:
  - `n,m` with `n`, `m` being numbers: work on lines `n` through `m`
  - `n` with `n` a number: work on line `n`

Deleting lines with sed

- The `d` command deletes lines.
- Use it with a range (otherwise it deletes all lines)
- `sed <n>,<m>d` – delete lines `n` through `m`:

```
$ cat rhyme.txt
When witches go riding
and black cats are seen
the moon laughs and whispers
'tis near Halloween
```

```
$ sed 1,3d rhyme.txt
'tis near Halloween
```

Deleting lines with sed

- The `d` command deletes lines.
- Use it with a range (otherwise it deletes all lines)
- `sed <n>,<m>d` – delete lines `n` through `m`:

```
$ cat rhyme.txt
When witches go riding
and black cats are seen
the moon laughs and whispers
'tis near Halloween
```

```
$ sed 1,3d rhyme.txt
'tis near Halloween
```

- Delete first line only:

```
$ sed 1d rhyme.txt
and black cats are seen
the moon laughs and whispers
'tis near Halloween
```

Search and replace with sed

- The `s` command does search and replace
- This is what `sed` is used for most often
- Usage: `sed s/text-to-search/replacement/[flags]`

```
$ cat my_frog.txt
Down by the river is a little frog, frog, frog.
```

- Replace 'frog' with 'cat'

```
$ sed 's/frog/cat/' my_frog.txt
Down by the river is a little cat, frog, frog.
```

sed “s” command flags

- g – (globally) replace all occurrences (not only the first)

```
$ sed 's/frog/cat/g' my_frog.txt
Down by the river is a little cat, cat, cat.
```

- i – ignores case

```
$ sed 's/down/Up/i' my_frog.txt
Up by the river is a little frog, frog, frog.
```

More sed examples

- Replace 'and' with 'or', but only in line 3

```
$ cat rhyme.txt
When witches go riding
and black cats are seen
the moon laughs and whispers
'tis near Halloween
```

```
$ sed '3s/and/or/g' rhyme.txt
When witches go riding
and black cats are seen
the moon laughs or whispers
'tis near Halloween
```

Exercise

1. Still have your `recipe.txt`? Print it to the terminal with all vowels removed
2. Pick a unit that you used in `recipe.txt` and replace it with "Olympic-size swimming pool" (using `sed`)
3. Do this again, but send the result to `recipe-improved.txt`
4. Bonus: When using the `s` command of `sed`, the beginning of the line is written as $\hat{}$. Use this to add the text "`Please consider: `" to each line of the recipe.

Exercise

1. Still have your `recipe.txt`? Print it to the terminal with all vowels removed
2. Pick a unit that you used in `recipe.txt` and replace it with "Olympic-size swimming pool" (using `sed`)
3. Do this again, but send the result to `recipe-improved.txt`
4. Bonus: When using the `s` command of `sed`, the beginning of the line is written as ^. Use this to add the text "`Please consider: `" to each line of the recipe.

```
$ tr -d AEIOUaeiou recipe.txt
$ sed "s/spoon/Olympic-size swimming pool/g" recipe.txt >
    recipe-improved.txt
$ sed "s/^/Please consider: /" recipe.txt
```

Command: `sort` – sort the content of files

- `sort` takes in a file and outputs its lines in sorted order
- Like `sed`, the original remains unchanged
- Usage: `sort [OPTION] <FILE>`
- Sorting rules are determined by the configured language
- 1. numbers, 2. lowercase letters, 3. uppercase letters

Command: `sort` – sort the content of files

- `sort` takes in a file and outputs its lines in sorted order
- Like `sed`, the original remains unchanged
- Usage: `sort [OPTION] <FILE>`
- Sorting rules are determined by the configured language
- 1. numbers, 2. lowercase letters, 3. uppercase letters

```
$ cat lines.txt
zebra
ape
donkey
$ sort lines.txt
ape
donkey
zebra
```

Numerical sorting

- `sort` option `-n` compares according to string numerical value

```
$ cat counts.txt
5 apes
1 donkey
16 zebras
```

```
$ sort counts.txt
16 zebras
1 donkey
5 apes
```

- $\rightarrow$ We need `-n` to sort numbers:

```
$ sort -n counts.txt
1 donkey
5 apes
16 zebras
```

Sorting by a column

- `sort` option `-k` specifes a column to sort by

```
$ sort -k 2 counts.txt
5 apes
1 donkey
16 zebras
```

Command: `uniq` – remove duplicate lines

- `uniq` discards consecutive duplicate lines
- Usage: `uniq [OPTION] <INPUT>`

```
$ cat myfile.txt
This is a line.
This is a line.
This is a line.
This is also a line.
This is also a line.
This is also also a line.
```

```
$ uniq myfile.txt
This is a line.
This is also a line.
This is also also a line.
```

- The duplicates must be consecutive! If necessary, `sort` first.

Options for `uniq`

- `uniq -c` counts how often a line has been seen

```
$ uniq -c myfile.txt
  3 This is a line.
  2 This is also a line.
  1 This is also also a line.
```

Options for `uniq`

- `uniq -c` counts how often a line has been seen

```
$ uniq -c myfile.txt
  3 This is a line.
  2 This is also a line.
  1 This is also also a line.
```

- `uniq -d` prints only duplicate lines

```
$ uniq -d myfile.txt
This is a line.
This is also a line.
```

Options for `uniq`

- `uniq -c` counts how often a line has been seen

```
$ uniq -c myfile.txt
  3 This is a line.
  2 This is also a line.
  1 This is also also a line.
```

- `uniq -d` prints only duplicate lines

```
$ uniq -d myfile.txt
This is a line.
This is also a line.
```

- `uniq -u` prints only unique lines

```
$ uniq -u myfile.txt
This is also a line.
```

Command: cut – extract columns from files

- `cut` extracts columns of each line of a file
- `cut [OPTIONS] <FILE>`
- Use option `-f` to select a "field"
- Use `-d` to set the delimiter

```
$ cat employees.txt
Simon Strange 62
Pete Brown 37
Mark Brown 46
$ cut -d ' ' -f 3 employees.txt
62
37
46
```

Command: `grep` – search for patterns

- `grep` finds and prints lines in files that match a pattern
- Usage: `grep [OPTIONS] PATTERN [FILE]`

Command: `grep` – search for patterns

- `grep` finds and prints lines in files that match a pattern
- Usage: `grep [OPTIONS] PATTERN [FILE]`
- The pattern is given as *regular expression*
  - Normal text is interpreted as-is: `grep zebra animals.txt`
  - But some characters get a different meaning (e. g. wildcards): `grep episode[1-3] movies.txt`

grep: An example

```
$ cat input.txt
Welcome to Linux.
This is an introductory workshop
for those with little experience.
```

```
$ grep "workshop" input.txt
This is an introductory workshop
```

grep: Pattern matching is case sensitive

- With option `-i`, `grep` ignores letter case

```
$ cat input.txt
Welcome to Linux.
This is an introductory workshop
for those with little experience.
```

```
$ grep "LINUX" input.txt
$ grep -i "LINUX" input.txt
Welcome to Linux.
```

grep: Search in more that one file

- With multiple input files, `grep` searches all given inputs

```
$ grep "Linux" input.txt output.txt
input.txt: Welcome to Linux.
output.txt: I hope you enjoyed working on Linux.
```

- Using '*' to search an entire directory:

```
$ grep "Linux" *
input.txt: Welcome to Linux.
output.txt: I hope you enjoyed working on Linux.
Binary file Linux_exercises.pdf matches
grep: new_dir: Is a directory
```

grep: Search recursively

- With option `-r`, `grep` searches the working directory *and also subdirectories recursively*

```
$ grep -r "Linux"
input.txt: Welcome to Linux.
new_dir/new.txt: Linux vs Windows
output.txt: I hope you enjoyed working on Linux
Binary file Linux_exercises.pdf matches
```

grep: Search words only

- With option `-w`, the pattern is found only if it is a whole word

```
$ cat groceries.txt
butter
milk
potato
milkshake
```

```
$ grep "milk" groceries.txt
milk
milkshake
```

```
$ grep -w "milk" groceries.txt
milk
```

grep: Count the matching lines

- With option `-c`, only the number of matching lines is printed

```
$ cat groceries.txt
butter
milk
potato
milkshake
```

```
$ grep -c "milk" groceries.txt
2
```

grep: Invert match

- With option `-v`, `grep` prints the lines that don't match

```
$ grep -v "milk" groceries.txt
butter
potato
```

Regular expressions

- The `grep` pattern is a regular expression
- They (somewhat) resemble shell wildcards
- They allow to do complex searches
- Example: Find all lines that have 't' as the third character:

```
$ grep "^..t" groceries.txt
butter
potato
```

  - The `^` matches the beginning of the line
  - The `.` matches exactly one character
  - The `t` matches `t`

Regular expression operators

Operator	Effect
.	matches any single character
$	matches the end of a line
^	matches the beginning of a line
?	matches at most one occurrences of the preceding expression
*	matches zero or more occurrences of the preceding expression
+	matches one or more occurrences of the preceding expression
[ ]	matches any of the characters between the brackets
{N}	the preceding item is matched exactly N times
(pattern1|pattern2)	matches pattern1 *or* pattern2 (this|that)

Nucleotide sequence regular expressions

Pattern	Match
^ATG	find a pattern starting with ATG
^A[T,G,C]G	find sequences starting with ATG, AGG, or ACG
TAG$	find a sequence ending with TAG
TA[G,A]$	find a sequence matching either TAG or TAA
^A[T,G,C]G*TGTGAACT*TA[G,A]$	find a motif

Bioinformatics example: FASTA identifiers

- FASTA files contain (nucleotide) sequences. Example:

```
$ cd data/fasta
$ head -n 3 brachy_CDS.fa
>Bradi1g00200.1 chr01_pseudomolecule brac ...
ATGGCCGGTGACGATGAACTGAAGCTTCTGGGCACATGGGCC...
CCCTCCACCTCAAGGGCCTAAGCTACGAGTACGTCGAGCAGG...
```

Bioinformatics example: FASTA identifiers

- FASTA files contain (nucleotide) sequences. Example:

```
$ cd data/fasta
$ head -n 3 brachy_CDS.fa
>Bradi1g00200.1 chr01_pseudomolecule brac ...
ATGGCCGGTGACGATGAACTGAAGCTTCTGGGCACATGGGCC...
CCCTCCACCTCAAGGGCCTAAGCTACGAGTACGTCGAGCAGG...
```

- Use `grep` to get a list of all identifiers:

```
$ grep ">" brachy_CDS.fa | head -n 2
>Bradi1g00200.1 chr01_pseudomolecule brac ...
>Bradi1g00210.1 chr01_pseudomolecule brac ...
```

Bioinformatics example: FASTA identifiers

- FASTA files contain (nucleotide) sequences. Example:

```
$ cd data/fasta
$ head -n 3 brachy_CDS.fa
>Bradi1g00200.1 chr01_pseudomolecule brac ...
ATGGCCGGTGACGATGAACTGAAGCTTCTGGGCACATGGGCC...
CCCTCCACCTCAAGGGCCTAAGCTACGAGTACGTCGAGCAGG...
```

- Use `grep` to get a list of all identifiers:

```
$ grep ">" brachy_CDS.fa | head -n 2
>Bradi1g00200.1 chr01_pseudomolecule brac ...
>Bradi1g00210.1 chr01_pseudomolecule brac ...
```

- How do we count the identifiers?

Bioinformatics example: FASTA identifiers

- FASTA files contain (nucleotide) sequences. Example:

```
$ cd data/fasta
$ head -n 3 brachy_CDS.fa
>Bradi1g00200.1 chr01_pseudomolecule brac ...
ATGGCCGGTGACGATGAACTGAAGCTTCTGGGCACATGGGCC...
CCCTCCACCTCAAGGGCCTAAGCTACGAGTACGTCGAGCAGG...
```

- Use `grep` to get a list of all identifiers:

```
$ grep ">" brachy_CDS.fa | head -n 2
>Bradi1g00200.1 chr01_pseudomolecule brac ...
>Bradi1g00210.1 chr01_pseudomolecule brac ...
```

- How do we count the identifiers?

```
$ grep ">" brachy_CDS.fa | wc -l
31029

$ grep -c ">" brachy_CDS.fa
31029
```

- Question: What happens if you forget the quotation marks?

Summary

- `echo`
- `tr`
- `basename`
- `sed`
- `sort`
- `uniq`
- `cut`
- `grep`
- Regular expressions

The final command

```
exit
```