Module 4

Process Text Streams Using Text Processing Filters

4.1 Working with Text Files

- Unix-like systems are designed to manipulate text very well
- The same techniques can be used with plain text, or text-based formats
  - Most Unix configuration files are plain text
- Text is usually in the **ASCII** character set
  - Non-English text might use the ISO-8859 character sets
  - Unicode is better, but unfortunately many Linux command-line utilities don't (directly) support it yet

4.2 Lines of Text

- Text files are naturally divided into lines
- In Linux a line ends in a **line feed** character
  - Character number 10, hexadecimal 0x0A
- Other operating systems use different combinations
  - Windows and DOS use a carriage return followed by a line feed
  - Macintosh systems use only a carriage return
  - Programs are available to convert between the various formats

4.3 Filtering Text and Piping

- The Unix philosophy: use small programs, and link them together as needed
- Each tool should be good at one specific job
- Join programs together with **pipes**
  - Indicated with the pipe character: `|`
  - The first program prints text to its **standard output**
  - That gets fed into the second program's **standard input**
- For example, to connect the output of `echo` to the input of `wc`:

```
$ echo "count these words, boy" | wc
```

4.4 Displaying Files with `less`

- If a file is too long to fit in the terminal, display it with `less`:

```
$ less README
```

- `less` also makes it easy to clear the terminal of other things, so is useful even for small files
- Often used on the end of a pipe line, especially when it is not known how long the output will be:

```
$ wc *.txt | less
```

- Doesn't choke on strange characters, so it won't mess up your terminal (unlike `cat`)

4.5 Counting Words and Lines with `wc`

- `wc` counts characters, words and lines in a file
- If used with multiple files, outputs counts for each file, and a combined total
- Options:
  - `-c` output character count
  - `-l` output line count
  - `-w` output word count
  - Default is `-clw`
- Examples: display word count for *essay.txt*:

```
$ wc -w essay.txt
```

- Display the total number of lines in several text files:

```
$ wc -l *.txt
```


4.6 Sorting Lines of Text with `sort`

- The `sort` filter reads lines of text and prints them sorted into order
- For example, to sort a list of words into dictionary order:

```
$ sort words > sorted-words
```

- The `-f` option makes the sorting **case-insensitive**
- The `-n` option sorts numerically, rather than lexicographically

4.7 Removing Duplicate Lines with `uniq`

- Use `uniq` to find unique lines in a file
  - Removes *consecutive* duplicate lines
  - Usually give it sorted input, to remove all duplicates
- Example: find out how many unique words are in a dictionary:

```
$ sort /usr/dict/words | uniq | wc -w
```

- `sort` has a `-u` option to do this, without using a separate program:

```
$ sort -u /usr/dict/words | wc -w
```

- `sort | uniq` can do more than `sort -u`, though:
  - `uniq -c` counts how many times each line appeared
  - `uniq -u` prints only unique lines
  - `uniq -d` prints only duplicated lines

4.8 Selecting Parts of Lines with `cut`

- Used to select columns or fields from each line of input
- Select a range of
  - Characters, with `-c`
  - Fields, with `-f`
- Field separator specified with `-d` (defaults to tab)
- A range is written as start and end position: e.g., `3-5`
  - Either can be omitted
  - The first character or field is numbered 1, not 0
- Example: select usernames of logged in users:

```
$ who | cut -d"␣" -f1 | sort -u
```


4.9 Expanding Tabs to Spaces with `expand`

- Used to replace tabs with spaces in files
- Tab size (maximum number of spaces for each tab) can be set with `-t` *`number`*
  - Default tab size is 8
- To only change tabs at the beginning of lines, use `-i`
- Example: change all tabs in *foo.txt* to three spaces, display it to the screen:

```
$ expand -t 3 foo.txt
$ expand -3 foo.txt
```

4.10 Using `fmt` to Format Text Files

- Arranges words nicely into lines of consistent length
- Use `-u` to convert to uniform spacing
  - One space between words, two between sentences
- Use `-w` *`width`* to set the maximum line width in characters
  - Defaults to 75
- Example: change the line length of *notes.txt* to a maximum of 70 characters, and display it on the screen:

```
$ fmt -w 70 notes.txt | less
```

4.11 Reading the Start of a File with `head`

- Prints the top of its input, and discards the rest
- Set the number of lines to print with `-n` *`lines`* or `-`*`lines`*
  - Defaults to ten lines
- View the headers of a HTML document called *homepage.html*:

```
$ head homepage.html
```

- Print the first line of a text file (two alternatives):

```
$ head -n 1 notes.txt
$ head -1 notes.txt
```


4.12 Reading the End of a File with `tail`

- Similar to `head`, but prints lines at the end of a file
- The `-f` option watches the file forever
  - Continually updates the display as new entries are appended to the end of the file
  - Kill it with `Ctrl+C`
- The option `-n` is the same as in `head` (number of lines to print)
- Example: monitor HTTP requests on a webserver:

```
$ tail -f /var/log/httpd/access_log
```

4.13 Numbering Lines of a File with `nl` or `cat`

- Display the input with line numbers against each line
- There are options to finely control the formating
- By default, blank lines aren't numbered
  - The option `-ba` numbers every line
  - `cat -n` also numbers lines, including blank ones

4.14 Dumping Bytes of Binary Data with `od`

- Prints the numeric values of the bytes in a file
- Useful for studying files with non-text characters
- By default, prints two-byte words in octal
- Specify an alternative with the `-t` option
  - Give a letter to indicate base: `o` for octal, `x` for hexadecimal, `u` for unsigned decimal, etc.
  - Can be followed by the number of bytes per word
  - Add `z` to show ASCII equivalents alongside the numbers
  - A useful format is given by `od -t x1z` — hexadecimal, one byte words, with ASCII
- Alternatives to `od` include `xxd` and `hexdump`

4.15 Paginating Text Files with `pr`

- Convert a text file into paginated text, with headers and page fills
- Rarely useful for modern printers
- Options:
  - `-d` double spaced output
  - `-h` *`header`* change from the default header to *`header`*
  - `-l` *`lines`* change the default lines on a page from 66 to *`lines`*
  - `-o` *`width`* set ('offset') the left margin to *`width`*
- Example:

```
$ pr -h "My Thesis" thesis.txt | lpr
```

4.16 Dividing Files into Chunks with `split`

- Splits files into equal-sized segments
- Syntax: `split [`*`options`*`] [`*`input`*`] [`*`output-prefix`*`]`
- Use `-l` *`n`* to split a file into *`n`*-line chunks
- Use `-b` *`n`* to split into chunks of *`n`* bytes each
- Output files are named using the specified output name with *aa*, *ab*, *ac*, etc., added to the end of the prefix
- Example: Split *essay.txt* into 30-line files, and save the output to files *short_aa*, *short_ab*, etc:

```
$ split -l 30 essay.txt short_
```

4.17 Using `split` to Span Disks

- If a file is too big to fit on a single floppy, Zip or CD-ROM disk, it can be split into small enough chunks
- Use the `-b` option, and with the `k` and `m` sufixes to give the chunk size in kilobytes or megabytes
- For example, to split the file *database.tar.gz* into pieces small enough to fit on Zip disks:

```
$ split -b 90m database.tar.gz zip-
```

- Use `cat` to put the pieces back together:

```
$ cat zip-* > database.tar.gz
```

4.18 Reversing Files with `tac`

- Similar to `cat`, but in reverse
- Prints the last line of the input first, the penultimate line second, and so on
- Example: show a list of logins and logouts, but with the most recent events at the end:

```
$ last | tac
```


4.19 Translating Sets of Characters with `tr`

- `tr` translates one set of characters to another
- Usage: `tr` *`start-set end-set`*
- Replaces all characters in *`start-set`* with the corresponding characters in *`end-set`*
- Cannot accept a file as an argument, but uses the standard input and output
- Options:
  - `-d` deletes characters in *`start-set`* instead of translating them
  - `-s` replaces sequences of identical characters with just one (squeezes them)

4.20 `tr` Examples

- Replace all uppercase characters in *input-file* with lowercase characters (two alternatives):

```
$ cat input-file | tr A-Z a-z
$ tr A-Z a-z < input-file
```

- Delete all occurrences of `z` in *story.txt*:

```
$ cat story.txt | tr -d z
```

- Run together each sequence of repeated `f` characters in *lullaby.txt* to with just one `f`:

```
$ tr -s f < lullaby.txt
```

4.21 Modifying Files with `sed`

- `sed` uses a simple script to process each line of a file
- Specify the script file with `-f` *`filename`*
- Or give individual commands with `-e` *`command`*
- For example, if you have a script called *spelling.sed* which corrects your most common mistakes, you can feed a file through it:

```
$ sed -f spelling.sed < report.txt > corrected.txt
```

4.22 Substituting with `sed`

- Use the `s/`*`pattern`*`/`*`replacement`*`/` command to substitute text matching the *`pattern`* with the *`replacement`*
  - Add the `/g` modifier to replace every occurrence on each line, rather than just the first one
- For example, replace 'thru' with 'through':

```
$ sed -e 's/thru/through/g' input-file > output-file
```

- `sed` has more complicated facilities which allow commands to be executed conditionally
  - Can be used as a very basic (but unpleasantly difficult!) programming language

4.23 Put Files Side-by-Side with `paste`

- `paste` takes lines from two or more files and puts them in columns of the output
- Use `-d` *`char`* to set the delimiter between fields in the output
  - The default is tab
  - Giving `-d` more than one character sets different delimiters between each pair of columns
- Example: assign passwords to users, separating them with a colon:

```
$ paste -d: usernames passwords > .htpasswd
```

4.24 Performing Database Joins with `join`

- Does a database-style 'inner join' on two tables, stored in text files
- The `-t` option sets the field delimiter
  - By default, fields are separated by any number of spaces or tabs
- Example: show details of suppliers and their products:

```
$ join suppliers.txt products.txt | less
```

- The input files must be sorted!
- This command is rarely used — databases have this facility built in

4.25 Exercises

1. **a.** Type in the example on the `cut` slide to display a list of users logged in. (Try just `who` on its own first to see what is happening.)
   b. Arrange for the list of usernames in `who`'s output to be sorted, and remove any duplicates.
   c. Try the command `last` to display a record of login sessions, and then try reversing it with `tac`. Which is more useful? What if you pipe the output into `less`?
   d. Use `sed` to correct the misspelling 'enviroment' to 'environment'. Use it on a test file, containing a few lines of text, to check it. Does it work if the misspelling occurs more than once on the same line?
   e. Use `nl` to number the lines in the output of the previous question.
2. **a.** Try making an empty file and using `tail -f` to monitor it. Then add lines to it from a different terminal, using a command like this:

   ```
   $ echo "testing" >>filename
   ```

   b. Once you have written some lines into your file, use `tr` to display it with all occurances of the letters `A`–`F` changed to the numbers `0`–`5`.
   c. Try looking at the binary for the `ls` command (*/bin/ls*) with `less`. You can use the `-f` option to force it to display the file, even though it isn't text.
   d. Try viewing the same binary with `od`. Try it in its default mode, as well as with the options shown on the slide for outputting in hexadecimal.
3. **a.** Use the `split` command to split the binary of the `ls` command into 1Kb chunks. You might want to

create a directory especially for the split files, so that it can all be easily deleted later.

b. Put your split `ls` command back together again, and run it to make sure it still works. You will have to make sure you are running the new copy of it, for example `./my_ls`, and make sure that the program is marked as 'executable' to run it, with the following command:

```
$ chmod a+rx my_ls
```