Unix Shell Software Carpentry Notes

- The Unix shell is both a command-line interface (CLI) and a scripting language. The most popular Unix shell is Bash, the default shell on most modern implementations of Unix and in most packages that provide Unix-like tools for Windows.
- `ls` prints the names of the files and directories in the current directory.
- About `ls`:
  - The `ls --help` option displays more information on how to use `ls`, whereas `man ls` turns the terminal into a page with a description of `ls` and its options.
    - Use ↑ and ↓ to move line-by-line.
    - Use `B` and `Space` to skip up and down by a full page.
    - Use `/` followed by the character or word to search. If there are multiple hits, use `N` and `Shift + N` to move forward and backward.
    - Use `Q` to quit the `man` page.
  - `-F` tells `ls` to classify the output by adding a marker to file and directory names to indicate what they are:
    - `/`: directory.
    - `@`: link.
    - `*`: executable.

    Any names in the output that don't have a classification symbol are plain old files.
  - `-l` makes `ls` use a long listing format, showing not only the file/directory names but also additional information, such as the file size and the time of its last modification. If one uses both `-h` and `-l`, then the file size becomes human readable (e.g.: displaying `5.3K` instead of `5369`).
  - `-t` lists items by time of last change rather than in alphabetical order by default.
  - `-r` lists the contents of a directory in reverse order.
  - `-s` displays the size of files and directories alongside the names.
  - `-S` sorts the files and directories by size.
  - `-R` lists all nested subdirectories within a directory.
  - `-a`[1] lists ALL directories, where
    - `./` refers to the current working directory.
    - `../` refers to the parent directory.

    One may also see other files starting with a dot (e.g.: `.bash_profile`). These files usually contain shell (or other programs) configuration settings. The prefix `.` is used to prevent these configuration files from cluttering the terminal when a standard `ls` command is used.

[1] Note that in most command line tools, multiple options can be combined with a single `-` and no spaces between the options (e.g.: `ls -F -a` is equivalent to `ls -Fa`).

  - `ls` can be given multiple paths at once.

  `ls Desktop` can also output the list of contents of a different directory. Pressing `Tab` twice more after a complete directory to see all of those files in the target directory.

- About `cd`:
  - `cd -` allows one to move to the previous directory.
  - `cd ~` allows one to move to the home directory[2].
  - `cd ../..` allows one to move back up two levels.
  - `cd /` allows one to move back to the root directory.
- About `mkdir`:
  - `-p` allows `mkdir` to create nested subdirectories. Note that Bash does not accept spaces within filenames and one needs to surround the name in double quotes (`""`) instead.
- About `nano`[3]:
  - `nano draft.txt` creates a new file called `draft.txt` and uses `nano` to edit it, whereas `touch draft.txt` simply creates the new file.
  - `Ctrl + O` followed by `Enter` saves the file, and `Ctrl + X` quits the editor.
- About `mv`:
  - The `mv` command can move a file to another directory or simply rename a file if the previous and new files are in the same directory.
  - `mv` silently overwrites any existing file with the same name. An additional option `-i` can be used to make `mv` ask for confirmation before overwriting.
  - `mv thesis/quotes.txt .` moves the `quotes.txt` file to the current directory, as per explained before.
- About `cp`:
  - The `cp` command copies a file instead of moving it. If it receives multiple arguments, it expects the last argument to be a directory in which the former files are to be moved.
  - `-r` copies a directory and all its contents.
- About `rm`:
  - `-i` brings up an interactive session that asks our permission before deleting a file. One needs to note that there is no recycle bin in Unix, so the deletion is always permanent.
  - `-r` removes a directory and all its contents.

[2] In Unix, the directory `/mnt/c` is equivalent to `C:` on Windows.

[3] `nano` is a text editor that is the simplest. Check `Emacs`, `Vim`, `Gedit`, `Notepad++` for more information.

- A regular expression specifies a set of strings required for a particular purpose. Operators and quantifiers involve:
  - `()` is for grouping.
  - `?` indicates zero or one occurrences of the preceding element[4].
  - `*` indicates zero or more occurrences of the preceding element[5].
  - `+` indicates one or more occurrences of the preceding element.
  - `{n}` indicates the preceding item being matched exactly *n* times.
  - `{min,}` indicates the preceding item being matched *min* or more times.
  - `{,max}` indicates the preceding item being matched *max* or less times.
  - `{min, max}` indicates the preceding item being matched more than *min* but less than *max* times.
  - `[]` matches a single character that is contained within the brackets (e.g.: [abcx-z] matches 'a', 'b', 'c', 'x', 'y', or 'z'). A starting `^` at the beginning within the bracket matches a single character other than the specified characters.
  - `^` matches the starting position.
  - `$` matches the ending position.
  - `.` is known as a wildcard[6]. It matches any character. For example,
    - `a.b` matches `acb` or `aob`.
    - `a.*b` matches `acb` or `aerb`.
- About `wc`:
  - The `wc` commands output the number of lines, words, and characters in files, which can be specified by `-l`, `-w`, and `-m`.
- About `cat`:
  - `cat` prints the contents of files one after another, but it has the disadvantage that it always dumps the whole file onto your screen. Instead, one can use `less`.
- About `sort`:
  - The `sort` commands sort the contents of a file in alphanumerical order. Note that the command does not change the original file but simply sends the sorted result to the screen.
  - `-n` specifies a numerical sort rather than the default alphanumerical sort.
- About `echo`;
  - The `echo` command prints strings.

[4] In Bash, `?` does not operate on the previous character.

[5] In Bash, `*` does not operate on the previous character.

[6] Note that in Bash, the wildcard character is `*` rather than `.`.

  - Two operators:
    * `>` tells the shell to overwrite the output to a file.
    * `>>` tells the shell to append the output to a file.

- About `cut`:
  - The `cut` command is used to remove certain sections of each line in the file. It expects the lines to be separated into columns by a `Tab` character (or generally speaking, a delimiter).
  - `-d` specifies a delimiter.
  - `-f` specifies the field we want to extract.

- About `uniq`:
  - The `uniq` command filters out adjacent[7] matching lines in a file.
  - `-c` gives a count of the number of times a line occurs in its input.

- About loops in Bash:
  - A for loop can look like

```
for filename in basilisk.dat minotaur.dat unicorn.dat
do
    head –n 2 $filename | tail –n 1
done
```

    Instead, one can write this code in one line by

```
for f in *.dat; do head –n 2 $f | tail –n 1; done
```

    where `;` is used to separate arguments. One should note that if the filename contains a space, then it is necessary to put `""` around both the loop body and the loop element.

- About `history`:
  - The `history` command gets a list of the last few hundred commands that have been executed, and then to use `!123` to execute again the 123 command.
  - `Ctrl + R` enters a history search mode 'reverse-i-search' and finds the most recent command in the history. Press `Ctrl + R` again can search for earlier matches. One can then edit the line by ← and → before hit `Enter` to run the command again.
  - `!!` retrieves the immediately preceding command, just as ↑.
  - `!$` retrieves the last word of the last command (e.g.: one may use `less !$` to look at the previous file).

- About shell scripts:

[7] It has to be used together with `sort`

- By creating a shell script via `nano tmp.sh` and then put commands in it, one can reuse the codes easily by `bash tmp.sh`.
- It is also possible to require inputs from the prompt. For example, let the file `bash tmp.sh` be

```
head –n 15 ”$1” | tail –n 5
```

  Here `$1` means the first argument on the command line[8]. Now, one can run the script by

```
bash tmp.sh file.txt
```

  Of course, one can add `$2`, `$3`, etc. to allow more inputs in the command line. However, one should add comments (by `#`) to indicate what to type in the command to improve readability.
- The special syntax `$@` is to handle the case how many files there are is unknown ex ante. Again, the character is usually surrounded by double quotes.
- One can do

```
history | tail –n 5 > his.sh
```

  to save the last 5 commands in the prompt, including the `history` command as well. Only can one reuse these codes before removing the preceding numbers in each line and the final `history` command.
- `-x` runs the script in debug mode, which prints out each command as it is running.

- About `grep`:
  - `-w` restricts to lines containing the word on its own rather than all strings containing it. It can also search for phrases:

```
grep –w ”is not” haiku.txt
```

  - `-n` also outputs the numbers the lines that match.
  - `-i` makes the search case-insensitive.
  - `-v` inverts a search.
  - `-r` can recursively search for a pattern through a set of files in subdirectories.

```
grep –r Yesterday .
```

  - `-E` enables one to select regular expressions:

```
grep –E ”^.o” haiku.txt
```

    This command selects any line whose second character is ’o’. Note that we need to put the regular expression inside double quotes to prevent the shell from trying to interpret it.

[8]The double quotes here is just in case the filename happens to contain any spaces.

  - `-o` prints only the matched (non-empty) parts of a matching line with each such part on a separate output line.
  - `-c` reports only the number of lines matched.

- About `find`:
  - The `find` command finds files and directories.
  - `-type d` finds directories in a specified directory, whereas `-type f` finds files.
  - `-name` finds files by their name. However, we expect to use the following command to find all `.txt` files in the current working directory (specified by `.`)

```
find . –name ∗.txt
```

    but the shell expands wildcard characters like `*` before commands run. Since `*.txt` in the current working directory expands to `./numbers.txt`, the command we actually ran was:

```
find . –name numbers.txt
```

    To solve this, again, we surround the argument by double quotes.

  - Though the `find` command is kind of similar to that of `ls`, it's much more useful in certain ways. For example, one wants to count the number of lines of all `.txt` files that span several directories, one can write

```
wc –l $(find . –name ”∗.txt”)
```

    It first executes the `find` command and then the `wc` command.