The Beauty of Linux I

by Imad Rahal

A. **Redirecting Output**: Every Linux process has three file descriptors:

- *standard in* ***stdin*** used as the source of input for the process (e.g. keyboard) … ***file descriptor 0***
- *standard out* ***stdout*** used as the destination for process's output (e.g. screen) … ***file descriptor 1***
- *standard error* ***stderr*** used as the destination for process's errors messages (e.g. screen) … ***file descriptor 2***

Processes are written generically to read from *keyboard* as ***stdin*** and write to *screen* as ***stdout*** but that can be changed. Redirecting I/O can be accomplished on Linux via **<** and **>**, respectively (**>&** for ***stderr***). E.g.:

- (1) Command `ls` lists the contents of the current folder. Open a Linux terminal and try it out.
- (2) Now, issue command `ls > my.file` [this redirects the output of the `ls` command to a new file called `my.file`] and then issue command `cat my.file` to see the contents of the file [`cat` displays the contents of the specified file in the terminal]
- (3) Issue command `sort` and hit enter followed by a number of text lines to sort (when done entering data, press `CTRL-d`) to see how it works. For example, try the following lines:

```
   once upon a time
   a small creature
   came to live in
   the forest
   ^D
   a small creature
   came to live in
   once upon a time
   the forest
```

  [This `sort` command sorted the lines!]
- (4) Let us repeat the above by issuing command `sort > story.txt` first followed by the text data from above (press `CTRL-d` when done) and then issue command `cat story.txt`
- (5) Now, place the same data to sort in a file called `story2.txt` and issue command `sort < story2.txt`

B. **Linux Pipes**: Output of one Linux command can be sent as input to another making a connection known as a *pipe. Linux pipes* are created using the **|** character (PS: Linux command `wc` prints the number of *lines*, *words*, and *bytes* in a given file)

- (1) issue command `ls | wc > wc.fields` [this example forwards the output of the `ls` Linux command to the `wc` command which counts the number of lines, words and bytes and sends output to file `wc.fields`].
- (2) Use the `cat` command to see the contents of file `wc.fields`

C. **File Types & Contents**: In Linux, file names are made up of alphanumeric characters, periods and underscores (BTW, a dot or period in a file name means nothing in Linux! Extensions aren't recognized on Linux). Names are case sensitive. To view the contents of a given file, the following commands can be used:

- `cat filename`
- `more filename` (used with large files – use space bar to go thru file)
- `head -x filename` (displays the first **x** lines in the file)
- `tail -x filename` (displays the last **x** lines in the file)
- (1) Try the above commands using file `story2.txt` created earlier in **A**.

The command **`file filename`** looks at the first several hundred bytes of a given file (i.e. **`filename`**) and does a statistical analysis of the types of characters that it finds there.

- (2) issue command **`file wc.fields`**
- (3) issue command **`file`** on a **`.java`** and a **`.class`** file

D. Permissions: Permissions in Linux are divided into three categories: *User* (or Owner), *Group* and *Others*. Any file belongs to a single user and thus to a single group. Files have separate *Read*/*Write*/*Execute* permissions for each of the above categories in this order: *User* permissions*, Group* permissions and then *Others* permissions. E.g. **`drwxr-xr--`** means that this a directory not a file (starts with a *d*) where the owner can **`r`** (i.e. read), **`w`** (write) and **`x`** (execute), group members can **`r`** and **`x`** while others can only **`r`**).

- (1) Use **`ls -l`** to view current permissions on files and folders in your home directory.
- Command **`chmod`** ***`who=permissions filename`*** changes the permissions on a given file based on the following
  - **Who:** A list of letters specifying whom you're going to be giving permissions to
    - ✓ **`u`** The **u**ser who owns the file (this means "you.")
    - ✓ **`g`** The **g**roup the file belongs to
    - ✓ **`o`** The **o**ther users
    - ✓ **`a`** **a**ll of the above (an abbreviation for **`ugo`**)
  - **Permissions**
    - ✓ **r** Permission to **r**ead the file
    - ✓ **w** Permission to **w**rite (or delete) the file
    - ✓ **x** Permission to e**x**ecute the file, or, in the case of a directory, search it

For example, to:

- prevent outsiders from writing or executing file **`archive.sh`** one can issue command **`chmod o=r archive.sh`**
- take away all group permissions on **`topsecret.inf`** one can issue command **`chmod g= topsecret.inf`** (i.e. leave the permissions part of the command empty)
- allow reading and writing of **`publicity.html`** by anyone issue command **`chmod og=rw publicity.html`**
- (2) Change permissions on **`wc.fields`** so that the owner can **`rwx`** while group and others can **`r`** (PS: you might need to run two different **`chmod`** commands

E. Generalized Regular Expression Processor – *grep*: A great tool for searching through the contents of a file for fixed sequences of characters or regular expressions: e.g. **`grep 'myString' Main.java`**

- Searches for and displays all lines in file **`Main.java`** that contain the string **`myString`**
- The following flags maybe used
  - **`-i`**: Ignore case differences
  - **`-l`**: only list filenames not actual lines
  - **`-n`**: show line numbers where matches were found
  - **`-v`**: reverse meaning of the search (i.e. all lines that don't match the pattern)
    - ✓ To search for all lines containing **`println`** except those using standard I/O (i.e. **`System.out`**)
    - ✓ **`grep 'println' *.java | grep -v 'System.out'`**
- (1) search for all lines (display line numbers as well) in file **`story.txt`** (created earlier) containing the string **`'i'`**
- To match a selection of characters, use `[]: e.g.` **`[Hh]ello`** matches lines containing **`hello`** or **`Hello`**
- Ranges of characters are also permitted

- `[0-3]` is the same as `[0123]`
  - `[a-k]` is the same as `[abcdefghijk]`
  - `[A-C]` is the same as `[ABC]`
  - `[A-Ca-k]` is the same as `[ABCabcdefghijk]`

F. The `find` Command:

- E.g.: `find . -name '*JDBC*' -print` looks for any file whose name contains **JDBC**. It starts looking in the current directory (note the `.` in the command) and descends into all subdirectories. `-name` is called a predicate. It takes a regular expression as an argument. Any file that matches the predicate passes the control to the next predicate (`-print` in previous example)
- (1) find and display all java files (i.e. `–name ‘*.java’`) under your account [if it takes too long, press `CTRL-c` to cancel]
- The following command finds all files in folder **JDBC** (and descending from there) containing either string **JDBC** or **jDBC** in their names. Names of matching files are then passed to the `–exec` predicate which executes the next Linux command (i.e., `ls –l`) on them.
  - `find ./JDBC -name '*[jJ]DBC*.java' -exec ls -l '{}' \;`
  - PS: `'{}'` are replaced with the names of the matching files and `\;` indicates the end of the command (with no space between \ and **;**)
- (2) change the above command and run it to display permissions on all files with names ending in `.java` under your account
- Other useful predicates
  - `-type d` ➔ true if the file is a directory
  - `-type f` ➔ true if the file is a plain file
  - `-mtime -5` ➔ file is less than 5 days old (i.e. modified within the last 5 days … +5 means older than five days … a 5 with no sign means exactly five days)
  - `-atime -5`➔ file was accessed less than 5 days ago
  - `-newer myEx.class` ➔ file is newer than `myEx.class`
  - `-size +24k` ➔ file is greater than 24k
- (3) Without running them, can you figure out what the following commands do?
  - `find . -name '*.java' -mtime -10 -atime +5 -print`
  - `find . -name '*.java' -mtime -10 -atime +5 -exec rm '{}' \;`