Lab 1

Using UNIX command line utilities

In this lab you will use UNIX command line utilities to do some data cleaning and basic analysis. At the end of the exercise, remember to fill out the response form.

For all the utilities mentioned in this lab, you can read its manual by running `man command-name` from a terminal.

Section 1: File System Analytics

In this exercise, we'll use some of the basic aggregation facilities provided by UNIX tools to do some analysis of the file system on your virtual machine.

Navigating the file system

If you've used the terminal on a Linux or UNIX system before, you're probably very familiar with `ls` and `cd`. `ls` has some great extended options, but accessing the results of `ls` programmatically can be difficult, and tasks like getting access to filenames recursively can be tricky.

find

`find` provides a much more flexible interface to listing files that match a particular criteria than ls. In particular, you can easily restrict yourself to a particular type of file (e.g. directories or plain files), recurse indefinitely or to some maximum depth, dispatch an action on file names that match your search, or pair the results of find with a command like `xargs` to do something further with files that match your filters.

For example, the command `find /usr/bin -type f -name 'py*'` will tell find to list the names of all ordinary files (not directories) in `/usr/bin` whose names start with "py".

du

Understanding what files/directories are using space on the file system is a problem that people have been dealing with since UNIX was invented. According to the man page for `du`, "A `du` command appeared in Version 1 AT&T UNIX."

`du` allows you to report usage statistics of files and directories in your system. By default, it reports its usage statistics at the block level, but the `-h` option provides this information in a human-readable format. It's worth noting that the results of `du` are not exact - instead sizes are reported as multiples of block size.

Try running `du -hs *` in your home directory to see where the space is going.

Now, think about how you could output the results of `find` into `du` to do reporting on a subset of files.

Sorting

The `sort` command is pretty powerful. In most systems, it is implemented via an external merge sort - this makes it possible to sort really huge files, even ones that don't fit into memory, in a reasonable amount of time.

By default, `sort` will sort items lexicographically, but with the `-n` flag it will sort in numeric order. This means that 10100 will come after 2 in numeric sort order, even though it precedes 2 lexicographically. In recent versions of GNU `sort`, the command also takes a `-h` parameter, which allows you to sort human-readable numbers (like file sizes produced by du -h).

Additionally, `sort` will let you select exactly which fields of a file you want to sort on (by default, it's the whole line), and in which order these fields should be sorted.

Exercises

1.1. What are the 10 biggest directories at depth 1 in /usr/lib on your virtual machine?

1.2. What are the 5 biggest directories in /home/datascience, including hidden folders?

Section 2: Log processing with command line tools

In the next exercise we will look at tools which you can use to quickly analyze and explore text files.

Downloading data files

The first step in data analyis is typically downloading data files that you need to process. While typically you might be used to downloading files using your web browser, this gets tedious when you want to download many files or if you want to automate the process. There are UNIX command line tools which you can use in such cases to download files

curl

`curl` is a commonly used tool for downloading files. It is typically available on most UNIX machines and should be installed in your VM. To download a file using curl you need to run `curl <url> -o <filename>`. For example to download the course webpage in html you can run something like

```
curl http://biddata.github.io/datascience-fa14/index.html -o index.html
```

This will download the course webpage and save it to a file named `index.html` in your current directory. This is of course a simple example of how you can use `curl`. You can find other options using and features in the curl manpage using `man curl`.

By default `curl` does not follow redirects when downloading a file. To make it follow redirects, add `-L` to the curl command-line.

wget

Another popular command line tool used for downloading data is `wget`. You might have seen this used in other examples or used it before. For simple use cases you can use either tool and we will use `curl` for this exercise. A more detailed comparison of the two tools can be seen at curl vs wget .

HTTP Logs Dataset

For this exercise we will be using HTTP access logs from the 1998 Soccer WorldCup website. The complete dataset contains around 1.3 billion requests, and we will use a subset of it for this exercise. As a first step download the sample dataset using `curl` from `https://raw.github.com/biddata/datascience/master/F14/lab1/data/wc_day6_1_log.tar.bz2` . (You will need `-L` since this redirects to a non-github.com domain for security reasons .)

The dataset has been compressed to make the download finish faster. To get the raw data unzip the downloaded file by running `tar -xf <filename>`. (Note: `tar` is also a very frequently used command line tool and you can learn more about it with `man tar`).

Having extracted the file, take a look at how the file looks by running `less wc_day6_1.log`. This will show you the first few lines of the file and you can page through the file using the arrow keys. You will notice that each hit or access to the website is logged as in a new line in the log file. The format of each line is in the Common Log File Format and this format is supported by most HTTP servers. In this case the data has been annonymized and lets take a look at one line from the file to explain each field

```
57 - - [30/Apr/1998:22:00:48 +0000] "GET /english/images/lateb_new.gif HTTP/1.0" 200 1431
```

In the above line `57` refers to a `clientID`, a unique integer identifier for the client that issued the request. While this field typically contains the IP address, for privacy reasons it has been replaced with an integer. The next two fields are `-` and say that the fields are missing from the log entry. Again these correspond to `user-identifier`, `userid` and have been removed for privacy reasons.

The next field is the time at which the request was made and this is followed by the HTTP request that was made. In this example a GET request was made for `lateb_new.gif`. The next field is the HTTP return code and in this example it is 200. You can find a list of codes and their meanings from w3 . The last field is the size of the object returned to the client, measured in bytes.

Exploring the dataset

Before we get to the exercises, lets explore the dataset and try out some basic commands.

First up lets count how many visits the website got. To do this we just count the number of lines in the file by running `wc -l wc_day6_1.log`. Your output should look like

```
1193353 wc_day6_1.log
```

We can do something more interesting by finding out how many times the ticketing webpage was visited. To do this you could run

```
grep tickets wc_day6_1.log | wc -l
29818
```

However the above line counts images and other elements which have the word `tickets` in their path. (Note: You can verify this using `less`). To restrict it to just `html` pages, you can use a regular expression

```
grep "tickets.*html" wc_day6_1.log | wc -l
2776
```

We can also prune the dataset to only look at interesting parts of it. For example we can just look at the first 50 URIs and their sizes using the `head` and `cut` command.

```
head -50 wc_day6_1.log | cut -d ' ' -f 7,10
```

In the above command the `-d` flag denotes what delimiter to use and `-f` stats what fields should be selected from the line. Try out different delimiter and field values to see how `cut` works.

Finally we can see how many unique URIs are there in the first 50 visits. To do this we could run something like

```
head -50 wc_day6_1.log | cut -d ' ' -f 7 | sort | uniq | wc -l
```

Here we use the tool `uniq` to only count unique URIs. Note that the input to `uniq` should be sorted, so we use `sort` before calling `uniq`. The `uniq` tool can also be used to count how many times an item occurs by passing it the `-c` flag. For example if we run the same command as above but with `uniq -c` we'll get

```
head -50 wc_day6_1.log  | cut -d ' ' -f 7 | sort | uniq -c | tail -10
  1 /images/home_fr_phrase.gif
  2 /images/home_intro.anim.gif
  1 /images/home_logo.gif
  1 /images/home_sponsor.gif
  1 /images/home_tool.gif
  1 /images/logo_cfo.gif
  1 /images/nav_bg_top.gif
  1 /images/team_hm_afc.gif
  1 /images/team_hm_caf.gif
  1 /images/team_hm_concacaf.gif
```

This shows that `/images/home_intro.anim.gif` occured twice in the first 50 URIs.

Exercises

Now use the above tools to answer some analysis questions

2.1 What are the 5 most frequently visited URIs?

2.2 Print the top 10 URIs which did not have return code 200.

2.3 Print the number of requests that had HTTP return code 404. Next break down number of 404 requests by date (i.e how many on 30th April and how many on 1st May).

Section 3: Data transformation with sed

While the data in the log file is probvided in a standardized format, most tools we used are not intended specifically for analyzing web page logs. In this exercise, we will translate the log file into CSV (comma-separated values) format with fields:

1. Client ID
2. Date (YYYY-MM-DD format)
3. Time (HH:MM:SS format)
4. URI
5. Response code
6. Response size
7. Request method (GET, POST, etc.)

Regular expressions

To do this, we will take advantage of *regular expressions*. We have prepared a short introduction to regular expressions that might be useful if you want a quick overview. More detail is easy to find online, for example, in the GNU sed Manual .

Regular expression substituion in esd

`sed` (short for "Stream EDitor") is a tool for modifying and extracting data from text files. Many sed commands were discussed in this week's reading, but, by far, the most common sed command is regular expression substitution. You can run a substitute command as follows:

`cat in.txt | sed 's/regexPattern/replacementString/flags' > out.txt'`

The flags are described in `man sed`, but the most notable flag is `g`, which causes sed to find and replace all instances of the pattern rather than the first one.

For example,

`echo "The quick brown fox jumps over the lazy dog." | sed 's/[tT]he [a-z]*/The yellow/'`

matches only *The quick*, resulting in `The yellow brown fox jumps over the lazy dog.`, while

`echo "The quick brown fox jumps over the lazy dog." | sed 's/[tT]he [a-z]*/The yellow/g'`

matches both *The quick* and *the lazy* and results in `The yellow brown fox jumps over The yellow dog.`

Removing cruft in sed

As a first step toward converting it to CSV format, let's start by removing cruft from dates in the log file. Each date is surrounded by a `[` and `+0000]`. We can remove the `+0000]`s using `sed 's/ +0000]//'`. To try this on the first ten lines of the log file, use:

`head wc_day6_1.log | sed 's/ +0000]//'`

Similarly, we can remove `[` using `s/\[//`. (We need the backslash before the `[` to prevent sed from treating it as a special character in its regular expression syntax.) We can combine this our previous cleaning using:

`head wc_day6_1.log | sed 's/ +0000]//' | sed 's/\[//'`

or, only invoking `sed` once:

`head wc_day6_1.log | sed 's/ +0000]//; s/\[//'`

Backreferences in sed

`sed` supports *backreferences* in substitutions, allowing you to include part of the matched text in the replacement string. To use backreferences, make a *capturing group* by surrounding part of the matched text with `\(` and `\)`; then in the replacement text, use `\1` to place the text of the first group, `\2` for the second and so on. For example:

`echo "The quick brown fox jumps over the lazy dog." | sed 's/\([Tt]he\) \([a-z]*\)/\1 "\2"/g'`

results in `The "quick" brown fox jumps over the "lazy" dog.`

Task

3.2 Use `sed` to convert the log file into the CSV format mentioned above. Note the transformation of dates.

3.3 Check (e.g. with `cut` and `grep` or with a `sed` script) that each line of your converted log file actually contains 7 fields, each of which is never empty.

Challenge Exercises (Optional)

4.1. Print the number of HTTP requests that had return code 404 in each hour of the day.

4.2. Break down the number of HTTP requests that did not have a 200 return code by date (i.e. how many responses were 304, 404, etc. on each day).

Response form

Fill out the questions here.