Assignment 1 – Organizing Files from a Web Crawler

You were hired to work as part of a team that is building a new web search tool. Without much experience on the subject, and counting only with a small team, your team lead decided to take small steps. Another teammate is building the crawler, which produces a file with web links. On the other hand, you will be responsible for analyzing such text files with web links, downloading the files, and organizing them in a local Unix filesystem.

Description of the crawling file

The results of the web crawling are a file with a list of web links, one link per line, such as in the example.

List 1: Example of contents from the crawling file

```
https://getbootstrap.com/
https://getbootstrap.com/docs/5.3/getting-started/introduction/
https://getbootstrap.com/docs/5.3/assets/img/favicons/apple-touch-icon.png
https://www.uvic.ca/ecs/computerscience/people/faculty/index.php
https://www.uvic.ca/ecs/computerscience/people/faculty/profiles/bittencourt-roberto.php
https://www.uvic.ca/ecs/computerscience/people/staff/index.php
http://www.taco.com/
http://www.taco.com/sysadmin.html
http://www.taco.com/webhost.html
http://www.taco.com/assets/images/autogen/Systems___and_Networks_Ntaco2big.gif
http://www.taco.com/assets/images/autogen/Web___Hosting_Htaco2biga.gif
```

As you may see from the example, each URL describes the protocol (**`https`**), the web server (**`getbootstrap.com`**), and may have a hierarchy of subdirectories (e.g., **`docs`**, **`5.3`**, **`getting-started`** and **`introduction`**) and sometimes also a filename (**`apple-touch-icon.png`**). When there is no filename, the web server usually adds a default **`.html`** file name to the URL, such as **`index.html`** or **`default.htm`**.

Preprocessing the crawling file

First of all, you need to get rid of the files that are not text files. Images, audio or any file types that do not make sense to be indexed in the future web search should be discarded. Thus, produce a second file from the first one, eliminating files with extensions associated to non-text files (e.g., **`.jpg`**, **`.gif`**, **`.png`**, **`.mp3`**, **`.mp4`**). You may host a list of known extensions in another file to make it simpler filter them out or you may add them as strings. You may use **`grep -e`**, **`grep -f`**, and **`grep -v`**, if you wish. Use the man pages (**`man grep`**) to learn about those options. If still in doubt, you may also google those terms with double quotes to find additional explanations about them.

Creating a directory for each web server

First, you need to recover the web server address from the full URL. An easy way to do that is by tokenizing the URL line. Normally, bash tokenizes strings using either spaces, tabs or newlines as separators. You may temporarily change the shell separator by changing the IFS variable. But do not forget to return the IFS variable to its original value so you do not affect your shell operation.

An example of tokenizing may be seen in the following script. You may run it to check what it does. That may inspire you to find how to assign the web server address to a variable. It uses the **`set`** command to add a string's tokens to the script's arguments (e.g., **`$0`**, **`$1`**, **`$2`**, …). Notice that **`$#`** gives you the number of arguments to the script, and **`${!i}`** gives you the value of one of those arguments indexed by the **`i`** counter variable.

```
#!/bin/bash
string="don't//worry/be/happy"
OLDIFS="$IFS"
IFS="/"
set $string
total=$#
for (( i=1; i<=$total; i++))
do
    echo "Position" $i "value: "${!i}
done
IFS="$OLDIFS"
```

There is an important catch here if you decide to use this solution. If the code above that changes the separator to **/** is inside a loop that processes text lines (which is the case here, since there is one URL per line), you need to return your separator to the original separator every time you finish an iteration. Otherwise, the loop that processes lines will not be able itself to separate lines. So, do not forget to use the command **IFS="$OLDIFS"** when you finish your loop iteration.

After tokenizing the URL and isolating the web server address, you will use the web server address to name a directory that will keep the directories and files from the given web server.

Then choose a directory for your data and create it. Inside it, you may create the directories for each web server, using the web server address as the directory name.

Creating a hierarchy of directories for the directories in the URL path

Now it is time to create a hierarchy of directories for each link inside the list. The idea here is to produce a directory path as a string to allow for later directory creation. Use what you have learned with tokenizers to remove the protocol and web server address from the string. And then you may additionally check whether your URL finishes with a file or with a directory (hint: if the URL ends with **/**, it is a directory; and here it is way simpler to check the string end, no tokenizers needed, just a simple comparison using **if** and a wildcard, such as in **if [[$string == */]]**). If it ends with a file, you will have to remove the file from your new directory path.

The URL **https://www.uvic.ca/ecs/computerscience/people/faculty/index.php**, for instance, tells us that there should be some subdirectories inside the directory for **www.uvic.ca** web server that you created in the previous step. In this case, there should be a hierarchy of subdirectories (**ecs**, **computerscience**, **people** and **faculty**) for hosting the default HTML file that is received by requesting the **index.php** file. You may use **mkdir -p** to create the whole hierarchy with only one command. You may want to use **man mkdir** or **mkdir --help** to understand what **-p** does.

Downloading a file to the appropriate path

Use the **curl** tool to download an individual file from a web server (just use the full URL as the parameter). This tool sends the download file to standard output. Remember to redirect the output to the appropriate file inside the correct directory in the hierarchy of directories you have created.

Use a convention for your downloaded files. There are at least three different situations you have to deal with:

1. The URL ends with a directory – then name the output file as **index.html**;
2. The URL has a named text file (e.g., **.html** or **.txt** extensions) – then name the output file with the same filename as the one in the web server;
3. The URL produces a dynamic web page (e.g., by requesting files with **.php**, **.py** or **.perl** extensions) – then use a convention to name them. For instance, **[…]/faculty/index.php** will be named **index.php.html** inside the **faculty** subdirectory. And **[…]/profiles/bittencourt-roberto.php** will be named **bittencourt-roberto.php.html** inside the **profiles** subdirectory. To compare a list of file extensions, you may, for instance, use an array of dynamic extensions and loop over them, such as in the code excerpt below:

```
extensions=(".php" ".py" ".perl")
for extension in extensions
do
    if [[ $line == *"$extension"* ]]
        echo "Found it!"
        break
    fi
done
```

Of course, there are other possibilities of finding a list of substrings in a string, which might use other ideas such as regular expressions or even the **`grep`** command. The code above is just one of them.

When you're done

Test all your scripts before sending them. To do so, you may use the examples provided in the description of the crawling file to help build your tests. They are real files and you can compare what you download with what you may see in a web browser (especially if you choose seeing the page source from inside your web browser).

Also, after you test each script, go over the full sequence of scripts to make sure all of them work well together in sequence.

Please, name your files following a numerical sequence in order to facilitate grading (for instance: **`script1`**, …, **`script4`**). Be sure they have execution permissions to all. Send your script files was well as the additional data file(s) essential to run the scripts (e.g., the data file with the list of irrelevant file extensions) through *Brightspace* in the *Assignment 1* page. Remember to click *submit* afterward. You should receive a notification that your assignment was successfully submitted.