Problem 1

Consider the following string of ASCII characters that were captured by *Wireshark* when the browser sent an HTTP GET message (i.e., this is the actual content of an HTTP GET message). The characters <CR><LF> are carriage-return and line-feed characters. Answer the following questions, indicating where in the HTTP GET message below you find the answer.

```
GET /classes/spring17/cs118/project-1.html HTTP/1.1<CR><LF>
Host:  web.cs.ucla.edu<CR><LF>
Connection:  keep-alive<CR><LF>
Upgrade-Insecure-Requests:  1<CR><LF>
User-Agent:  Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/56.0.2924.87 Safari/537.36<CR><LF>
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8<CR><LF>
Referer:  http://web.cs.ucla.edu/classes/spring17/cs118/homeworks.html<CR><LF>
Accept-Encoding:  gzip, deflate, sdch<CR><LF>
Accept-Language:  en-US,en;q=0.8,lv;q=0.6,ru;q=0.4<CR><LF>
If-None-Match:  "5a17-54c4847c4f640-gzip"<CR><LF>
If-Modified-Since:  Mon, 03 Apr 2017 19:36:49 GMT<CR><LF>
```

1. What is the **full** URL of the document requested by the browser?
2. What version of HTTP is the browser running?
3. What type of browser initiates this message? Why is the browser type needed in an HTTP request message?
4. What is the IP Address of the host on which the browser is running?

1. http://web.cs.ucla.edu/classes/spring17/cs118/project-1.html
2. From the end of the first line: `HTTP/1.1`
3. From the `User-Agent` header, with a little research: `Chrome 56`.
   This message is useful to the application developer if the developer wishes to support multiple browsers that may not adhere to the standard. The server can parse this User-Agent to identify the type of browser and serve a modified version of the app which was tested on that browser.

   This link was helpful in answering this question:
   `https://developer.mozilla.org/en-US/docs/Web/HTTP/Browser_detection_using_the_user_agent`
4. These HTTP headers do not provide us with that information; in fact, it's a rather odd question to ask at this level because the source/destination IP address is what the IP layer uses in the IP header to deliver the individual packets that compose this HTTP request.

Problem 2

Alice and Bob are working remotely on a course project and are using `git` as the version control software.

1. Is it true that one must have GitHub/GitLab account to use git?
2. What is(are) the command(s) to initialize a local git repository?
3. Do Alice and Bob both must initialize local git repository? If no, what are the alternative?
4. Let's consider that Alice modified the file `server.cpp`:
   (a) What git commands Alice needs to save modifications in the local git repository
   (b) What git commands Alice needs to upload saved modifications to GitHub
   (c) What git commands Bob needs to get Alice's changes and apply them to the local repository
5. Let's consider that both Alice and Bob modified the file `server.cpp` and Alice was first to successfully upload saved modifications (commit) to GitHub
   (a) Can Bob upload his changes without any additional actions? If no, why?
   (b) If actions needed, list git commands that Bob will need to use to share his modifications with Alice.

1. It is absolutely false that you must have a GitHub/GitLab account to use git. They are different tools; git is used for version control and GitHub/GitLab are hosting solutions for git repositories.
2. The command to initialize a local git repository in your current working directory is `git init`
3. Alice and Bob probably shouldn't both initialize a local git repository unless they want to create a lot more work for themselves. It's much easier for one of them (e.g. Alice) to set up the respository on GitHub (e.g. at the URL `http://github.com/Alice/repository.git`) and for the other (e.g. Bob) to run the command `git clone http://github.com/Alice/respository.git`
4. (a) To save modifications in the local repository, Alice should run the commands:
   ```
   git add server.txt # adds server.txt to commit staging
   git commit # creates a new commit which stores the changes
   ```
   (b) To upload modifications to GitHub, Alice needs to create a repository on GitHub first. Let's say the URL is `http://github.com/Alice/repository.git`
   ```
   git remote add origin http://github.com/Alice/repository.git
   git push origin master
   ```
   (c) To get Alice's changes, Bob must run the `git pull` command which will require him to merge his local commits with the commits on GitHub's servers (if he's made any to the current branch).
5. (a) Bob can't upload his changes to GitHub without additional action because his changes are in conflict with Alice's changes on GitHub's server – git has two versions of what `server.cpp` should look like and we need to specify which changes will make it to the "final" version of `server.cpp` (this process is known as merging).
   (b) To share his changes, Bob must run `git pull` which will require him to merge his changes with Alice's (creating a new merge commit that references two parents – Alice's last commit and Bob's last commit). After the merge is complete, Bob will be able to push his merged changes onto the GitHub server.

Problem 3

You will learn some basic usages of `Vagrant` and `Docker` in your projects.

1. What is Vagrant mainly used for?
2. What is Docker mainly used for?
3. List at least one difference between Vagrant's "box" and Docker's "container"?
4. List at least five commands you can use with Vagrant.
5. List at least five commands you can use with Docker.

1. Vagrant is used to create development environments running on virtual machines that are identical to the environment used in production for ease of testing, development, and potentially deployment. The VM is actually simulated by a provider like VMWare, Virtualbox, etc.
2. Docker is different from Vagrant; it's used to wrap an application in a filesystem/execution environment that has everything the app needs to run correctly. The idea is that this "container" is standardized and can be "shipped" and executed on different machines, where the application will run in the same environment regardless of physical machine.
3. The difference between Vagrant's box and Docker's container is how they abstract the application environment: Vagrant relies on virtualization, whereas Docker relies on containerization. Virtualization requires the emulation of an operating system on simulated hardware and is in a computational sense more expensive to do than containerization. With containerization, the application runs directly on the host OS and hardware, but in an execution environment that is isolated from the rest of the OS through specially designed software tools.
4. (a) `vagrant init # creates Vagrantfile`
   (b) `vagrant up # starts and sets up VM environment`
   (c) `vagrant ssh # access the VM`
   (d) `vagrant halt # stop the VM`
   (e) `vagrant reload # restart the VM with new configuration`
5. (a) `docker ps # list running containers`
   (b) `docker build # build Docker image from Dockerfile`
   (c) `docker run # run a container`
   (d) `docker attach # enter a running container`
   (e) `docker kill # stop a running container`