1. Write a Ruby or Bash script that will print usernames of all users on a Linux system together with their home directories. Here's some example output:

```
gitlab:/home/gitlab
nobody:/nonexistent
```

As you can see, each line is a concatenation of a username, the colon character (:), and the home directory path for that username. Your script should output such a line for each user on the system.
Next, write a crontab entry that accomplishes the following:

- Runs once every hour.
- Takes the output of your above script and converts it to an MD5 hash.
- Stores the MD5 hash into the /var/log/current_users file.
- On subsequent runs, if the MD5 sum changes, it should log this change in the /var/log/user_changes file with the message, DATE TIME changes occurred, replacing DATE and TIME with appropriate values. Make sure to replace the old MD5 hash in /var/log/current_users file with the new MD5 hash.

Both the script and crontab entry should be provided for the answer to be complete.

# CRONTAB ENTRY

```
0     *     *     *     *     ~/monitor_user_list_changes.sh      # every hour
```

# The bash script: save below in a file called ~/monitor_user_list_changes.sh

```
#!/bin/bash

# +--------------------------------------------------------------------------------+
# | GitLab initial interview question/response                                     |
# +--------------------------------------------------------------------------------+
# | Benson Oluga                                                                   |
# | mcoluga@gmail.com                                                              |
# | 9th Feb, 2021                                                                  |
# +--------------------------------------------------------------------------------+

current_users=/var/log/current_users
user_changes=/var/log/user_changes
cur_md5=$(cut -d ":" -f 1,6 /etc/passwd | md5sum)

# file exists and has data in it
if [[ -f "$current_users" ]] && [[ -s "$current_users" ]]; then
      old_md5=$(cat $current_users)
      echo $cur_md5 >$current_users
      cur_md5=$(cat $current_users)
      echo "If executed: old_md5=${old_md5}"
      echo "If executed: cur_md5=${cur_md5}"
      if [[ $cur_md5 != $old_md5 ]]; then
            date +'%D %T changes occurred' >>$user_changes
      fi
else
      # file doesn't exist, create an empty file
      echo $cur_md5 >$current_users
      echo "else executed ${cur_md5}"
fi
```

2. A user is complaining that it's taking a long time to load a page on our web application. In your own words, write down and discuss the possible cause(s) of the slowness. Also describe how you would begin to troubleshoot this issue?

Keep the following information about the environment in mind:

- The web application is written in a modern MVC web framework.
- Application data is stored in a relational database.
- All components (web application, web server, database) are running on a single Linux box with 8GB RAM, 2 CPU cores, and SSD storage with ample free space.
- You have root access to this Linux box.

We are interested in learning about your experience with modern web applications, and your ability to reason about system design and architectural trade-offs. There are no right and wrong answers to this question. Feel free to write as much or as little as you feel is necessary.

Troubleshooting Steps

1. Collect as much information as possible relevant to the reported incident, whether from the user, or others sources (like performing own tests):
   a. What page is being loaded?
   b. Is the slowness happening on all pages?
   c. Are you submitting a form (such as a login page)?
   d. What browser is being used?
2. Try to reproduce the issue by accessing the web application yourself
3. If you establish that only one user is affected, then it can be an issue caused by the user’s system (client side).
4. If you establish that multiple users are affected, then it’s an issue affecting a common resource like the server (DB, Web server, etc.) or network (in case where the users share a LAN)
5. Check if there has been any changes immediately prior to the incident and see if there’s any relationship that might require a rollback
6. Read application logs and database logs to:
   a. see if any errors are being logged
   b. see timestamps of what components are being called and how long it takes to get a response from those components
   c. retrieve exact query being executed by the application and run it directly on the DB.
   d. In the DB, using developer tool like SQL developer, check for SQL execution plans and note if the query needs to be modified
7. Once an issue is identified, work towards resolving and test then communicate resolution.
8. Develop an RCA and implement a monitoring for a recurrence.

Possible cause(s) of the slowness

Client Side Issues:

1. The user’s machine is running multiple programmes and its resources are over utilized.
2. High latency in the user’s network. Slow site speeds can result from network congestion, bandwidth throttling and restrictions, data discrimination and filtering, or content filtering from user’s Internet Service Provider (ISP).
3. Malwares, viruses and other unwanted programmes in the user’s machine
4. Caching issues preventing optimized page from loading. Then when users access your site, their browsers can display the cached data instead of having to reload it.
5. Browser plugins slowing down the user’s browser

Troubleshooting Client Side Issues:

1. Check system utilization on the client and close some/all open programmes currently not being used
2. Perform network latency tests

- with apps or websites like fast.com
- ping command from the user's machine to your server to show packet loss and round trip times
- tracert/traceroute commands to determine hops, find which hops are taking longest, and eliminate where possible
- identify the actual request duration using tools like browser's developer tools (Chrome's Network profiler) or Fiddler

3. Run anti-virus/antimalware scans
4. Engage user's ISP
5. Uninstall/disable user's plugins on the browser

Server Side Issues:

1. Traffic spikes. This could be a result of:

- a marketing campaign/promo resulting in higher users
- DOS attack

2. Over-utilization of server resources (CPU, memory, ~~disk space (not a factor in this context)~~, disk I/O, network bandwidth).

- could be a result of traffic spike or running multiple programmes and zombie processes
- all applications, including DBs will be impacted if these resources are over utilized

3. Database is slow. This can be caused by:

- missing indexes that may be needed as DB tables grow
- non-optimized queries that result in whole table scans instead of using indexed columns in where clauses
- table locks that may cause long waits in the DB as application waits for the locks to be released
- Poorly designed database schema
- Inadequate storage I/O subsystem ~~(currently not a factor since we've been informed there's ample space)~~
- When the DB buffer pool too small compared to the request. Data retrieval and sorting does not happen well resulting in a lot of disk I/O
- Stored procedures for complex computations on database
- When DB connections are not being closed and returned to pool in cases of errors occurring
- When DB connections are not being pooled resulting in many connections being opened and impacting the DB performance

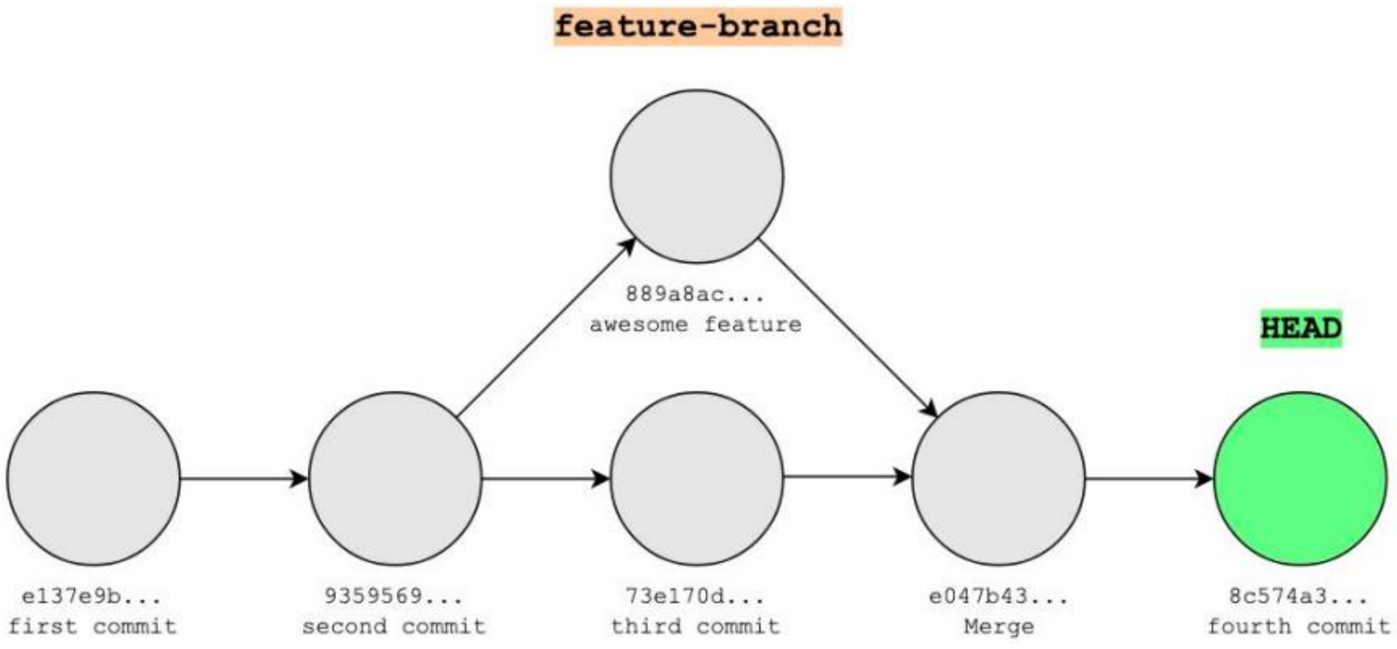
4. DNS issues. When DNS server takes long to resolve

Troubleshooting Server Side Issues:

1. Block unwanted traffic in case of DOS attack
2. Site optimization
- Unnecessarily large images can be reduced to smaller images while maintaining a relatively high resolution
- Limit linked elements
- Update, replace, or remove third-party plugins, especially when they're no longer being updated to support lates hardware and technology trends
- Remove unnecessary transitions/transforms, interactions and animations that may not improve user experience
3. Check server resource utilization:
- Disk utilization: `df -hP`
- Memory utilization: `free -mto`
- CPU and memory: `top`
- Disk I/O: `iostat -dx 5` (d – display disk utilization, x – display extended stats, at 5 second intervals)
4. On the basis of the finding:
- Kill zombie processes
- Clear disk by deleting/moving unused files like old log files
- Reduce log application log level to reduce disk I/O

- Add swap/RAM to the server

3. Study the Git commit graph shown below. What sequence of Git commands could have resulted in this commit graph?



```
git add *

git commit -m “first commit”

git add *

git commit -m “second commit”

git checkout -b feature-branch master

git add *

git commit -m “awesome feature”

git checkout master

git add *

git commit -m “third commit”

git merge -m “Merge” feature-branch

git add *

git commit -m “fourth commit”
```

4. GitLab has hired you to write a Git tutorial for beginners on: Using Git to implement a new feature/change without affecting the master branch

In your own words, write a tutorial/blog explaining things in a beginner-friendly way. Make sure to address both the "why" and "how" for each Git command you use. Assume the audience are readers of a well-known blog.

Using Git to implement a new feature/change without affecting the master branch

i. What is Git?

According to https://git-scm.com/, git is ***"a free and open source distributed version control system (VCS) designed to handle everything from small to very large projects with speed and efficiency".***

VCSs are software tools that help software teams manage changes to source code over time. They keep track of every modification to the code in a special kind of database. They are also known as Source Control Management (SCM)

ii. What is a git branch?

Because git is distributed, it allows individual users to work on and change the files and source code of various parts of a project without impacting everybody else working in the project.

This is achievable through branches in git.

A branch represents an independent line of development. Branches serve as an abstraction for the edit/stage/commit process.

iii. Types of Branches

The name of a branch is specified by the creator at the time of creation, but this can be renamed.

Despite the name, branches could be categorised into work flows of different roles:

- ***Master branch:*** Upon making the first commit in a repository, Git will automatically create a master branch by default. Subsequent commits will go under the master branch until you decide to create and switch over to another branch.
- ***Feature/Topic branch:*** When you start working on a new feature/bug fix, you should create a feature/topic branch. A feature/topic branch is normally created off a develop/integration branch.
- **Release branch:** When you roll out a new release, you create a release branch. A release branch helps you to ensure that the new features are running correctly. By convention, release branch names normally start with the prefix "release-".
- ***Hotfix branch:*** When you need to add an important fix to your production codebase quickly, you can create a Hotfix branch off the master branch. By convention, hotfix branch names normally start with the prefix "hotfix-".
- ***Develop/Integration branch:*** A develop/integration branch should be kept stable at all times. This is important because new branches are created off of this branch, and this branch could eventually go out live on production.

iv. Actual Steps

1. Start by cloning the copying the original repository with all the files and source code of the complete project
   ```
   git clone remote_url
   ```
   The above command creates a local copu=y of all the remote files, by default into a master branch
2. Create a new branch off the master. The switch `-b` creates the branch if it doesn't exist. The new branch will be called `feature-branch` and is created off `master`.
   ```
   git checkout -b feature-branch master
   ```
3. Edit your source code to implement the new feature
4. Once the feature is implemented, commit the changes to git using below commands:
   ```
   git status  # list all files that have been changed since last commit
   ```

```
git add *    # stage all the files that have been changed in readiness to commit
git commit –m “Using Git to implement a new feature/change without affecting the
master branch”    # this will commit all the changes by taking the current
snapshot
```

5. What is a technical book/blog you read recently that you enjoyed? Please include a brief review of what you especially liked or didn’t like about it.

I read *“The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win” by Gene Kim, Kevin Behr, and George Spafford* for the first and only time recently and could not help but notice how technical concepts were introduced and presented in an easy and non-technical way that makes the book quite enjoyable to read and easy to follow.

The books presents Parts Unlimited, a fictional company which has a myriad of problems, from technical issues to company politics, with only one employee, Brent Geller, appearing to know the operations of the applications. There’s little to no transfer of knowledge, employees work in silos, teams have walls that separate them and impedes work flow, there’s no proper work flow and handover process from the dev team to the support teams. Many projects get started without being brought to closure within time periods, resulting high project overheads without return on investment. There’s no prioritization of work and whoever is politically right with the powers that be tend to get the resource allocation. With legacy systems, poor and uncoordinated change processes, frequent system downtimes that catch people unawares and take long to resolve, the company is rapidly falling below its competitors in the market.

The business team has unrealistic expectations from the IT team but no one is able/ready to communicate clearly what is achievable within what time frame.

Even internal systems fail, with the payroll system failure being the top most priority for Bill Palmer to ensure employees are paid on time. As a result, employees have to work late nights, weekends, forgo leave days and forsaking any work-life balance.

The novel introduces DevOps as a solution to the problem. That DevOps requires management buy-in is very important and the presentation of the book in a non-technical manner means that even non-techies can read and follow the book.

The novel shows that DevOps can bring order into a company, resulting in IT being seen as a department that adds value to the whole company and not just a consumer of Capex & Opex. In addition, it means the organization can react quickly to market dynamics and keep afloat without heavy overheads.

But there’s no quick fix. It requires mind-set change, involvement of all, starting with the management, and most importantly, it’ll take a long time for initial fruits to be seen. There is no guarantee of all-time success. There will be failures, those who can’t adapt will leave. However, when the initial fruits are seen, more people will tend to join in.

Bill Palmer has an advisor called Eric. Through Eric, additional concepts like theory of constraints, lean thinking and agile methodologies are introduced. Also, the *three way (principles)* are introduced. These principles encourage flow, feedback and culture of continuous learning.

The authors elaborate on 4 types of work that exist in a typical work environment:

- Planned Work — exists in the cases where a business carries out projects or when new features are being added to existing systems/applications.
- Internal Projects — server migrations, software updates and so on.
- Changes — Occur when feedback from an already completed work requires new functionality.
- Unplanned Work — these crop up inform of downtimes, escalations.

Although Parts Unlimited is a manufacturing company, the lessons derived from it can be applied in all industries

The lessons and the suspense makes it a must read.

References:

- https://www.itprotoday.com/sql-server/examining-scalar-function-performance-sql-server
- https://dzone.com/articles/reasons-slow-database
- https://university.webflow.com/lesson/troubleshoot-website-performance-issues
- https://backlog.com/git-tutorial/branching-workflows/
- https://git-scm.com/