In this document you'll be given some tips that could help you implementing the project.

# 1 Command Line Interface

You're free to build the different sections in different scripts or just use the same script. However, when it comes to me, I'd use a command line interface for that project, where each command runs a different part of the project. One way of doing it would be by having a first argument representing the action and the second argument is the command that the project is going to work on.

```
$ ./PROJECT_1.sh generate lspci
# generates a manual document for lspci command

$ ./PROJECT_1.sh verify lspci
# verify the generated document for lspci command

$ ./PROJECT_1.sh search devices
# returns the commands that mention the word 'devices' in their manuals

$ ./PROJECT_1.sh suggest
# suggests 5 commands for the user
```

*But remember, you still have the choice to them separatly in different shell scripts.*

# 2 Manual Generation

In this section, you have to generate the document itself. The project gives some tips to generate the file, but here are more things that you can take a look at.
First, is the `column` command. Which allows you to format the result in a tabular format.

```
$ cat test
first 13 12456
second 1728 23

$ column -t test
first   13    12456
second  1728  23
```

Suppose that I have the description and the version stored in separate variables. Where the description is a long paragraph (without new lines). You can use another separator, like tabs, instead of spaces, using option `-t`. Also, you can enable cell wrapping to fit the long paragraph in the cell using option `-W <column>`, where `<column>` is the number of the column you want to enable wrapping for.

```
$ printf "Description\t$description\nVersion\t$version" | column -W 2 -s $'\t' -t
```

And this is how the output would look like:

```
Description  lspci is a utility for displaying information about PCI buses in
              the system and devices connected to them. By default, it shows
             a brief list of devices. Use the options described below to requ
             est either a more verbose output or output intended for parsing
             by other programs. If you are going to report bugs in PCI device
              drivers or in lspci itself, please include output of "lspci -vv
             x" or even better "lspci -vvxxx" (however, see below for possibl
             e caveats). Some parts of the output, especially in the highly v
             erbose modes, are probably intelligible only to experienced PCI
             hackers. For exact definitions of the fields, please consult eit
             her the PCI specifications or the header.h and /usr/include/linu
             x/pci.h include files. Access to some parts of the PCI configura
             tion space is restricted to root on many operating systems, so t
             he features of lspci available to normal users are limited. Howe
             ver, lspci tries its best to display as much as available and ma
             rk all other information with <access denied> text.
Version      lspci version 3.7.0
```

# 3 Manual Verification

In this section, you need to verify if there's anything in the generated manual that has changed. One command that you might find useful would be `diff`. Which finds the difference between two files. Suppose that the new file has the version updated to `3.8.0`, then the `diff` command would return the following output:

```
$ diff old_man new_man
17c17
< Version      lspci version 3.7.0
---
> Version      lspci version 3.8.0
```

This output might be useful to use for the verification process.

# 4 Command Search

In this part, you should allow the user to search for specific topic, like the word `devices`. If the user searched for that word, it would return the list of commands with the word `devices` in their manuals.

```
$ ./PROJECT_1.sh search devices
lspci
lsblk
```

# 5 Command Recommendation/Suggestion

In this part, you have to suggest few commands for the user when they ask for it. In order to do so, you need some sort of command history to rely on, and based on that history you need a policy to rank these commands based on and return the top few of them (let's say top 5).
For the history, you can simply use the `history` command to get the history of the latest commands that the user has run. Based on that, you'd use some policy to rank them, for example, you might get the most frequently used commands. For example if that was the user's command history:

```
$ history
1 ls
2 cd
3 ls
4 cd
5 ls
6 ls
7 pwd
```

Then the command `ls` is the most frequently used one, then `cd`, and finally `pwd`. There are other policies that you might use. For example, one can argue that the *least* frequently used command should be ranked higher, because it's more likely that the user would forget what it does or how it works. It's up to you to use whatever policy that'd make sense or would be easy to implement.

## 6 Random Sentences That I Could Not Find a Suitable Section Name For

1. **What's the difference between Related Commands and Recommended Commands?**

   Related commands are related to the command that you're generating the manual for, so for `ls`, this could be `cd` and `pwd`. While recommended commands are suggestions for the user to look at, it should be related to what the user is interested in, not related to a specific command.

2. **How does the user verify the command manual if the manual hasn't been generated in the first place?**

   You should assume that the manual has been generated, otherwise, just notify the user that there isn't any manual generated for that command and exit.

3. **Those are just tips. You are NOT REQUIRED to follow them. The project requirements are specified in the description message sent by Dr. Khader.**

4. **For any unspecified requirement. You have the freedom to do it the way you want.**

5. **DO NOT COMPLICATE THINGS IF THEY ARE NOT COMPLICATED. JUST DO THE SIMPLEST USEFUL THING.**

# Good Luck!