HW 7

Homework 7

Student name: Jama Brookes

This homework is based on the advanced Git lectures.

Question 1

What is the holy trinity of Git? (Just name the parts.)

```
Branch, Staging Area, and Working Copy
```

Question 2

Explain how patches relate to the idea of committing a change to the history of your code?

```
Patches provide line by line information on what is needed to transform a file
into a new version, which can be applied to a file. When it is applied to a file,
it changes the file in the working directory but these changes are not a part of
the projects history until they are committed. However, we can commit these changes
to alter the main branch if we wanted to. The history of the repository can be
thought of as a bunch of committed patches.
```

Question 3

What is the difference between a branch and a fork?

```
A branch points to a commit and is the history of your code.
A fork is permanent copy of a repository and is separate from the original.
```

Question 4

How are diff and patch related? What's the difference?

```
‘Diff‘ provides human-readable and machine readable files outlining
the differences between two files in commits, branches, or working directories.
The output of ‘diff‘ is used by ‘patch‘ to organize the differences line by line.
This information can then allow you to transform one version into another version.
The difference between ‘diff‘ and ‘patch‘ is ‘diff‘ just provides all differences
between two files while ‘patch‘ records all needed information needed to transform
the file.
```

Question 5

See the diagram below.

```
alice           *-*-*-*-d-e
bob             *-*-*-*-a-b-c
origin/main     *-*-*-*-a-b-c
```

a) What part of the Git trininty does each line represent?

```
Branch
```

b) What would the result be if Alice performed a rebase? Edit the diagram. What would she type in the terminal to rebase, then push?

```
alice           *-*-*-*-a-b-c-d'-e'
bob             *-*-*-*-a-b-c
origin/main     *-*-*-*-a-b-c-d'-e'
```

Code:

```
git rebase
git push
```

c) What would the result be if Alice performed a merge? Edit the diagram. What would she type in the terminal to merge, then push?

```
alice           *-*-*-*-d-e-a-b-c
bob             *-*-*-*-a-b-c
origin/main     *-*-*-*-d-e-a-b-c
```

Code:

```
git merge
git push
```

d) Which would be the better option (rebase vs. merge)? Why?

```
Rebase is the better option because rebase hides Alice's current
changes/working draft and adds the most recent commits from the
origin/main branch with Alice's changes on top.
This creates a more linear history.
```

Question 6

Match the command/vocab word to the description. You can edit the table!

Command / Vocab	Answer	Description
git stash	B	Takes any changes that haven't been committed and puts them in a dust bin
git push	J	Pushes the new commits to the main branch
git clone	N	Makes a copy of an existing repo at in a new directory at another location
git commit	M	Saves file changes to the main branch
git log	A	Gives project history
git add -i	K	Interactive staging!
git rebase	P	Combines branches by moving commits onto the tip of another branch, creating a linear history
git init	Q	Creates a new git repository
git checkout	E	Moves something from the branch to the working copy
git status	F	Lists which files are staged, unstaged, and untracked
git diff	O	Compares current, unsaved changes to the main branch
git merge	H	Combines two branches together in a way that is not ideal for your collaborators
git add	I	Adds changes from the working directory to the staging area
git pull	D	Retrieves any commits on the remote branch that you don't yet have locally and integrates them into your current branch
git show	G	Displays information about a specific commit
git blame	C	Shows who last modified each line of a file and in which commit
.gitignore	L	Text file that contains names of files Git should not track

Question 7

Walk me through how you would do interactive staging.

a) Firstly, what situation would interactive staging be useful in?

```
It allows you to organize your edits and commit them separately.
This may be useful if you are working on different sections of code
or files that may be more useful to be in separate commits for later
organization and documentation.
```

b) What command(s) help you prepare for what you'll see while interactive staging? *Hint*: Check the TA notes.

```
git status
git diff
```

b) What git command would you use to start interactive staging?

```
git add -i
```

c) On the text-based interactive menu, what option do you use?

```
p
```

d) After pressing that option, how do you select the file you want to stage?

```
Type the number of the file you want to stage.
Press Enter.
Then press ‘y‘ to stage the hunk.
```

e) What option do you type if you do not want to stage a hunk?

```
n
```

f) Once you get to a hunk you want to stage, what do you do?

```
Control D
git status
git commit -m "Your message here"
```

g) What if we have more hunks we want to stage?

```
If you do not exit interactive mode, we could use option ‘a‘
to stage current hunks and all remaining hunks or go hunk by hunk
by choosing different numbers to stage. You can commit these changes
by exiting and then committing.
Or, if interactive staging was closed, you can start over
the interactive section with: ‘git add -i‘ if you exited out.
```