Homework 7

Collin Stewart

https://github.com/collings512/BIOS512_Collin_Stewart

This homework is based on the advanced Git lectures.

Question 1

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

The Head, the Tree, and the Stage.

Question 2

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

Patches are applied to show any differences between 2 files. When using a patch, R takes the unified diff (showing the differences between two files) and then creates a new singular file with the changes incorporated. You can then move the changes applied in a patch to the staging area, and commit them to the history of your code. Commits essentially save the patched changes to your main branch.

Question 3

What is the difference between a branch and a fork?

A branch is a history or version of your code that can be followed chronologically with commits. Branches can diverge off other branches, presenting alternate versions of the code that continue as individual branches/chains. However, all branches can be followed back to the moment or individual commit that caused them to diverge from another branch. All branches are a history of the code within the same repository and are temporary.

A fork is an entirely different repository that is permanently separate from the original copy.

Question 4

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

Git diff compares any two files, and shows the difference between two files, in both human-readable and machine-readable formats.

Applying a patch will consider these differences and record all changes needed to transform one file into another, then save this to a new, separate file.

Using diff is what allows us to tell patch what we want to change/reconcile between two files, and patch carries it out.

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?

All three lines represent branches, showing the shared history of commits (*) as well as the different heads (abc or de).

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?

Rebase will temporarily remove Alice's changes (d and e), then add Bob's changes (a, b, and c) in front, then adding d and e back on the end. If Alice performed a rebase, the diagram would now look like:

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

In her terminal, she would type:

```
git checkout alice
git fetch origin
```

```
git rebase origin
git push origin main
```

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?

A merge would grab Bob's changes (abc) and put it on top of her Alice's changes (de), then commit this to the main branch. When Bob returns to his code, his copy will not match the history of the main branch.

If Alice performed a merge, the diagram would now look like:

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

In her terminal, she would type:

```
git checkout alice
git fetch origin
git merge origin
git push origin main
```

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

You would use git rebase because it takes the changes made by Bob and reorders it to incorporates Alice's changes to the main branch, without unsyncing Bob from the main branch. This now leaves Bob's changes in the history of commits so he can resume where he left off.

Question 6

Match the command/vocab word to the description.

You can edit the table!

Command / Vocab	Answer	Description
git stash	B.	A. Gives project history
git push	J.	B. Takes any changes that haven't been committed and puts them in a dust bin
git clone	N.	C. Shows who last modified each line of a file and in which commit
git commit	M.	D. Retrieves any commits on the remote branch that you don’t yet have locally and integrat
git log	A.	E. Moves something from the branch to the working copy
git add -i	K.	F. Lists which files are staged, unstaged, and untracked

Command / Vocab	Answer	Description
git rebase	P.	G. Displays information about a specific commit
git init	Q.	H. Combines two branches together in a way that is not ideal for your collaborators
git checkout	E.	I. Adds changes from the working directory to the staging area
git status	F.	J. Pushes the new commits to the main branch
git diff	O.	K. Interactive staging!
git merge	H.	L. Text file that contains names of files Git should not track
git add	I.	M. Saves file changes to the main branch
git pull	D.	N. Makes a copy of an existing repo at in a new directory at another location
git show	G.	O. Compares current, unsaved changes to the main branch
git blame	C.	P. Combines branches by moving commits onto the tip of another branch, creating a linear
.gitignore	L.	Q. Creates a new git repository

Question 7

Walk me through how you would do interactive staging.

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

Interactive staging would be useful when you have a lot of commits you want to do in a row, but you don't want to stop every few lines to do another commit each time. Interactive staging allows you to do them in a series.

b) What command(s) help you prepare for what you'll see while interactive staging?

Hint: Check the TA notes.

Using git status and git diff will allow you to see any file changes or differences before making your commits.

```
cd ~/BIOS512/BIOS512_Collin_Stewart
git status
git diff
```

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

Use git add -i

```
git add -i
```

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

You would use option 5: [p]atch, so you would type p.

```
p
```

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

After entering p, it will give you a list of files. Select the desired file by typing the number of the file and pressing enter.

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

Option n (do not stage this hunk).

```
n
```

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

When you get to a hunk you want to stage, enter option y.

Then, press Enter and Control D to exit the interactive staging. Enter git status to ensure that the change was added to the stage to commit.

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

If there are more hunks to stage you can reenter the interactive menu using git add -i and review the rest of the chunks. Either go through and repeat the process with "y" each time, or choose option "a" (stage this hunk and all remaining hunks).