

Lab 14 Check-Off

Student Name: _____

Partner Name: _____

Date of Lab: _____

Check-Offs

_____ The project has integrated a pull request into their master branch that was provided by another student.

_____ The student submitted a pull request to add comment functionality that was accepted by another student.

TA (signed): _____

TA (print): _____

Lab 14 Instructions

Lab Objectives: The primary objective of this lab to help students learn how to work with others using pull requests. Secondary objective is to learn more about git, including how to recover files from older git commits.

Directions:

1. We are going to begin by getting a copy of the project "my_world" from the profh directory on github. Get a copy of that file by typing the following on the command line:

```
git clone https://profh@github.com/profh/my_world.git
```

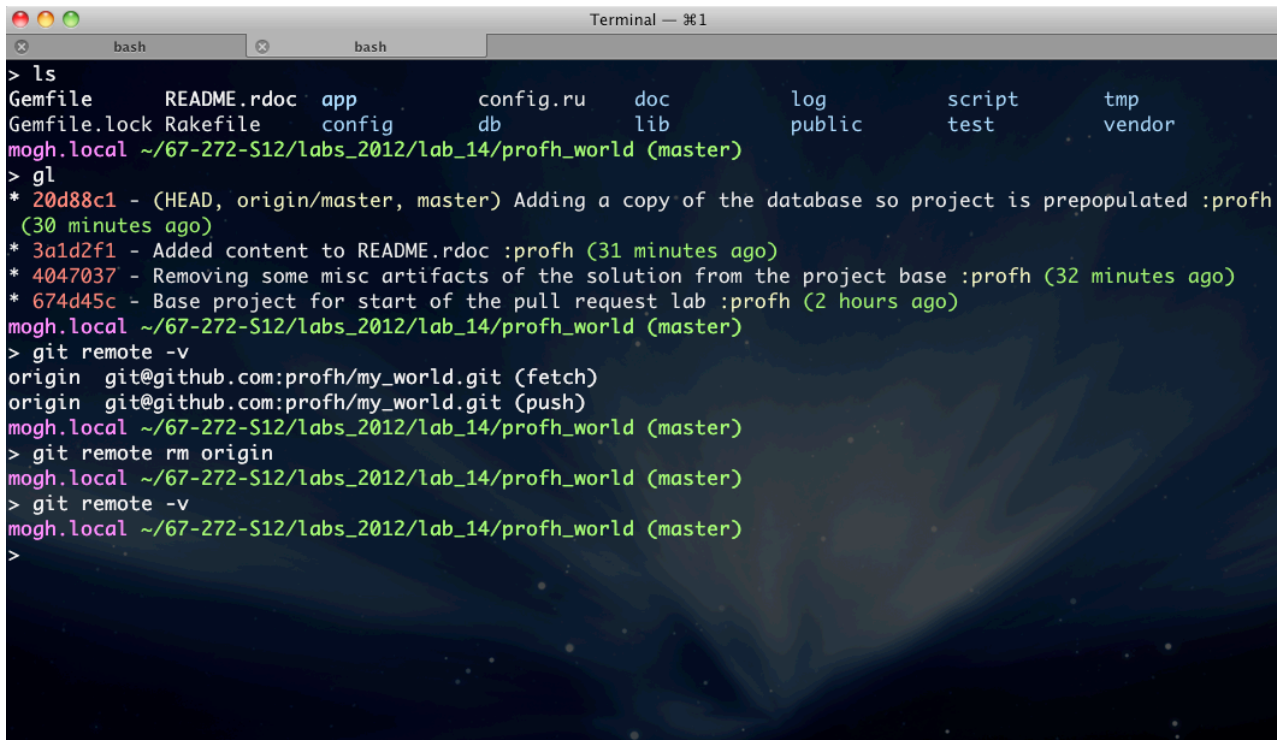
Once you have a copy of the project on your machine, do the following:

- a) change the name of the project directory to `<your_github_id>_world` (without the `<>`, and where `your_github_id` is your actual github id) and then switch into that directory. (If you don't have an account, go get one right now...)
- b) Look at the git log with the command `git log --oneline --graph` (NOTE: if you have Prof. H's dotfiles installed, just type 'gl' and you will get all this and some extra) and you see the git history for this project.
- c) Type the command `git remote -v` to see information about any remote repositories.

A "remote" is just another git repository. In this case, it's github, but it could also be a peer's computer, shared server (like darkknight), or even another directory on the same machine.

For more information on remotes, and git knowledge in general, visit: <http://gitref.org>.

- d) Remove the remote named `origin` (it should be the only one). Confirm its deletion by re-running the command `git remote -v`.
- e) If in doubt, your terminal window should look something like this:




A terminal window titled 'Terminal — #1' with two tabs labeled 'bash'. The terminal shows a directory listing with files like Gemfile, README.rdoc, app, config.ru, doc, log, script, tmp, and vendor. It then shows a series of git commands: 'gl' (which is an alias for 'git log'), 'git remote -v' (showing origin as git@github.com:profh/my_world.git), and 'git remote rm origin'. The commit history shows three commits: 20d88c1 (adding a database), 3a1d2f1 (adding content to README.rdoc), and 4047037 (removing misc artifacts). The prompt is 'mogh.local ~/67-272-S12/labs_2012/lab_14/profh_world (master)'.

```
> ls
Gemfile      README.rdoc  app          config.ru    doc          log          script       tmp
Gemfile.lock Rakefile     config       db           lib          public       test         vendor
mogh.local ~/67-272-S12/labs_2012/lab_14/profh_world (master)
> gl
* 20d88c1 - (HEAD, origin/master, master) Adding a copy of the database so project is prepopulated :profh (30 minutes ago)
* 3a1d2f1 - Added content to README.rdoc :profh (31 minutes ago)
* 4047037 - Removing some misc artifacts of the solution from the project base :profh (32 minutes ago)
* 674d45c - Base project for start of the pull request lab :profh (2 hours ago)
mogh.local ~/67-272-S12/labs_2012/lab_14/profh_world (master)
> git remote -v
origin  git@github.com:profh/my_world.git (fetch)
origin  git@github.com:profh/my_world.git (push)
mogh.local ~/67-272-S12/labs_2012/lab_14/profh_world (master)
> git remote rm origin
mogh.local ~/67-272-S12/labs_2012/lab_14/profh_world (master)
> git remote -v
mogh.local ~/67-272-S12/labs_2012/lab_14/profh_world (master)
>
```

2. To make the project operational, first run the `bundle` command to install any missing gems. To speed things along, the project already has a prepopulated `development.sqlite3` database to give you some idea of what this project is about. (If for some reason there is a problem with the db, the sql code to recreate it is in the `doc/` directory.) Once you feel comfortable with the application, customize it as you wish, although you don't have to do anything if you don't wish to.¹
3. Now we are going to create a repository for the revised project on github. Log into github (if you haven't already) and create an empty repository called `<your_github_id>_world`. The screenshot below will show you how it would look before you press the green 'Create repository' button:


¹ If you really want to fully customize the app, you can do a global find/replace for 'KcwWorld' and


Owner **Repository name**

 **profh**  


Great repository names are short and memorable. Need inspiration? How about **freezing-sansa**.

Description (optional)

☒ **Public** 
Anyone can see this repository. You choose who can commit.

☐ **Private** 
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will allow you to `git clone` the repository immediately.

Add .gitignore: **None** 

Create repository

After the repo² is created, github gives you the following instructions:

```
git remote add origin git@github.com:<your_github_id>/<your_github_id>_world.git
git push origin master
```

Why are they telling us this? The point is that now that we have created the remote repository on github, we need to add that remote to our local repo so the two are connected. The general "formula" for adding a remote repository is:

```
git remote add <REMOTE> <URL>:<REPO>
```

The `REMOTE` can be named whatever you want, but convention usually dictates that you name it `origin`. For github, the `URL` is `git@github.com`, and your `REPO` is namespaced under your `github_id`. Now that you know this, add the remote repository to your repo.

Push your code to github (`git push origin master`), and verify that it now shows up on github.com.

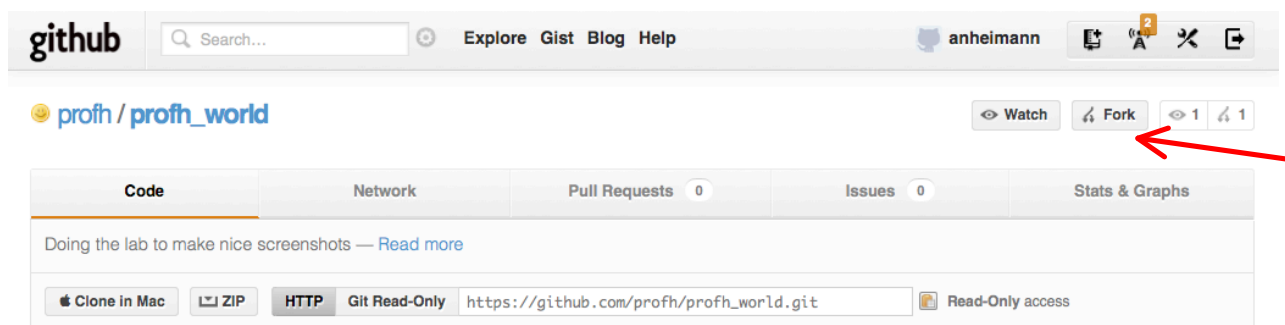
Your screen should look something like this:

² "repo" is just short for repository and can be used interchangeably

```
mogh.local ~/67-272-S12/labs_2012/lab_14/profh_world (master)
> git remote add origin git@github.com:profh/profh_world.git
mogh.local ~/67-272-S12/labs_2012/lab_14/profh_world (master)
> git push origin master
Counting objects: 195, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (173/173), done.
Writing objects: 100% (195/195), 467.62 KiB, done.
Total 195 (delta 33), reused 0 (delta 0)
To git@github.com:profh/profh_world.git
 * [new branch]      master -> master
mogh.local ~/67-272-S12/labs_2012/lab_14/profh_world (master)
>
```

Once this is done and your project is on github, notify a TA and tell him/her your github id so that you can be paired with another person.

4. The TA will assign you another student/team to work with -- write your partner's name(s) on the front of this lab now. You will be doing two roles at roughly the same time. In the first role, you will be the 'maintainer' of your project on github. In the later part of the lab, there are instructions for you to follow as the maintainer. It will require you to check in periodically to see if there have been any pull requests posted that you will need to review and possibly accept. In the second role you will be the 'requester' who will be adding additional functionality to the project and requesting that the maintainer pull your changes into the original project. I will put the name of the role in the beginning of the line so you know what role you are acting in.
5. **(Requester)** Go to the github repo of the person you are assigned to. Start by forking the project you've been paired with by clicking the 'Fork' button towards the upper right corner as seen below:



Now clone this forked project from your github account so you have a copy on your local machine. Create a branch in the forked project called "articles" and switch to that branch now. In this branch, create a new model called Article using the command `rails generate model Article` ... with the following fields:

- title:string
- content:text
- category_id:integer
- active:boolean

Now run the migration to add this table to the database. Once complete, go into the model and set the relationship to `category` (and also go into the `Category` model and set the relationship with `Article`). With relationships set, make sure that `title` and `content` are required fields and set up two scopes: `alphabetical` (which orders the records alphabetically by title) and `active` (which only returns active articles). Have a TA look over your model code before proceeding so you know the model is correct. Once you get the thumbs-up, commit this code to the 'articles' branch. Rerun `git log` to verify it was committed.

6. **(Requester)** Now that you are sure you have a decent working version for articles, it is time to open up a pull request. A pull request is a formal way of asking the maintainer to merge your code into his/her project.

- Push your changes back to your forked repository on github with the command `git push origin articles`
- On the forked project's github page (*i.e., in your github account*), switch to the branch 'articles' (*upper left side*)
- Press the 'Pull Request' button on the upper right side of the page. You will get a form asking for a title and description of the pull request as seen below – fill this out with information that will be useful to the maintainer and help him/her decide how to handle the request. (*Just like `git log` loses value quickly with meaningless comments like "made changes", a pull request with meaningful comments help a lot and make it more likely to be accepted by the maintainer.*) Be sure that the top line says "You're asking <user> to pull <x> commits into <original_project>:master from <your_fork>:articles" so that we are requesting the right material to be pulled into the project.

If everything looks good, submit the request by clicking the green 'Send pull request' button on the bottom right of the page.

The screenshot shows the GitHub interface for creating a pull request. At the top, it says 'anheimann / profh_world' and 'forked from profh/profh_world'. There's a 'Back to Source' button and a view count of 1. Below this is a bar with 'profh_world' and 'Send a pull request'. A summary line states: 'You're asking **profh** to pull 1 commit into **profh:master** from **anheimann:master**'. There's a 'Change Commits' button. Below this is a navigation bar with 'Preview Discussion', 'Commits 1', and 'Files Changed 1'. The main area has a 'Write' tab and a 'Preview' tab. The 'Write' tab is active, showing a text area with the title 'Adding articles to the profh_world project' and a description: 'We really want to have enough information here for the project maintainer to have enough information to make a good decision on whether to accept the request. First, why are you adding this functionality? (In many cases an issue would be opened first and you and the maintainer would already agree it's a good idea, but don't assume that here.) Second, a brief description of what you've done would be in order. Add all this into your request now...'. There's a 'Send pull request' button at the bottom right. On the right side, there's a 'Change base to send to another repository' link and a 'PEOPLE TO BE NOTIFIED' section with a dropdown menu showing 'profh'.

Example of screen for pull request (in your case it should be from a different branch than master)

7. (**Maintainer**) Go to your project page and see if you have any pull requests yet. *(People work at different speeds so just because you are done doesn't mean your partner is; you may have to check back periodically.)* If you look at the project bar at the top, you see that the third tab over is for pull requests; if the number next to it is one, then you have a pending request. If so, click on the tab and review the pull request. You will get something similar to what is below:

The screenshot shows the GitHub project page for 'profh / profh_world'. At the top, there's a navigation bar with 'Admin', 'Unwatch', 'Fork', 'Pull Request', and a view count of 2. Below this is a bar with tabs: 'Code', 'Network', 'Pull Requests 1', 'Issues 1', 'Wiki 0', and 'Stats & Graphs'. The 'Pull Requests 1' tab is selected. On the left side, there's a sidebar with 'All Requests 1' and 'Yours'. Below this is a search bar 'Find a user...' and a list of users with 'anheimann' selected. On the right side, there's a section titled '1 open request'. It shows a table with columns 'Read', 'Open', 'Closed', 'Submitted', 'Updated', and 'Popularity'. The first row is for the pull request 'Adding articles to the profh_world project'. It shows that 'anheimann' submitted it to 'profh/profh_world' 2 minutes ago, updated 2 minutes ago, and has 0 comments. A red arrow points to the title of the pull request. At the bottom, it says '1 open request and 0 closed requests'.

Clicking on the pull request title link (*in the screenshot above, it is 'Adding articles to the profh_world project' link in the middle of the page*) will take you to the details as seen in the example on the next page.

profh / profh_world

Admin Unwatch Fork Pull Request 2 2

Code Network Pull Requests 1 Issues 1 Wiki 0 Stats & Graphs

Open anheilmann wants someone to merge 1 commit into profh:master from anheilmann:master #1

Discussion Commits 1 Diff 1

anheilmann opened this pull request 4 minutes ago

Adding articles to the profh_world project

No one is assigned No milestone

We really want to have enough information here for the project maintainer to have enough information to make a good decision on whether to accept the request. First, why are you adding this functionality? (In many cases an issue would be opened first and you and the maintainer would already agree it's a good idea, but don't assume that here.) Second, a brief description of what you've done would be in order. Add all this into your request now...

1 participant Add a comment

anheilmann added some commits 13 minutes ago

a9a5396 Update README.rdoc

This pull request can be automatically merged. Merge pull request **X**

not yet... do this later

Comment on this pull request (Help) Close pull request

Write Preview Comments are parsed with GitHub Flavored Markdown

Tip: You can also add notes to lines changed in a file under Diff

Close & comment Comment on this pull request

WARNING: Do **NOT** click on 'Merge pull request' until explicitly instructed towards the end of the lab! You can review the code changes by clicking on any of the commits as seen in the screenshot below (and if while browsing the commit you click on the 'Browse code' button in the blue bar, you will go to the forked repo; you can even clone it and run it locally to see if you like it).

What you see after clicking on the commit link 'a9a5396' in the previous screenshot:

The screenshot shows a GitHub repository page for 'anheimann / profh_world', which is a fork of 'profh/profh_world'. The repository has 0 pull requests and 1 branch. The 'Commits' tab is selected, showing a commit titled 'Update README.rdoc' by 'anheimann' 19 minutes ago. The commit hash is 'a9a5396ced6d26ccae93fd5ca084e138e04a8dd'. The commit message is: '-NOTE: This project does require the use of the will_paginate, carrierwave, and simple_form gems to work properly. The twitter-b\n\\ No newline at end of file\n+NOTE: This project does require the use of the will_paginate, carrierwave, and simple_form gems to work properly. The twitter-b\n10 +\n11 +Minor edit so I can get a screenshot...\n\\ No newline at end of file'. The file 'README.rdoc' is shown with a diff view. Below the commit, there are 0 notes on the commit. A comment box is visible with a 'Write' tab and a 'Preview' tab. A tip suggests adding notes to lines in a file. A green button 'Comment on this commit' is at the bottom right.

However, as nice as this is, you should **NOT** merge this change into your project yet. On the pull request details page there is a link for 'Add a comment' in the lower right corner – click on that and add the following comment:

"I really like the idea of adding articles, but I think it needs more than just the models before I can make a decision. After all, Rails is all about MVC working together, not just the M portion in isolation. If the views and controller are in working order, then I'll pull this into my project."

Submit this comment and see that it was added to the pull request. Then go back to your work as a requestor.

8. **(Requester)** Check on your pull request and see if there has been any response yet. Once you see the response, clench your fists and say (quietly) "Oh shoot, I totally forgot about the views and controller!" You know you need to make controllers and views, but now start to wonder if there is a short cut. Then you remember seeing something unusual in the git log when you ran it earlier. Re-run the command `git log --oneline --graph` (or just `gl`) and notice the second commit with the comment "Removing some misc artifacts of the solution from the project base". You think to yourself that there might be a shortcut there if only I could access the project contents *before* that second commit; after all if Prof. H deleted files that were part of the solution, then perhaps the controller and views are still there in the repo. 'How could I see that?' you start to wonder...
9. **(Requester)** There is a way with git to get those files back. Remember in class we could checkout an individual file from the repo if we didn't like the changes made to it. We can also checkout entire commits and put them on a branch to investigate and possibly recover. To do this here we will take the following steps (*see screenshot on next page if uncertain*):
 - a) Move a copy of that first commit to a branch called 'recovery' with the following command, substituting the seven (or more) character sha value of the first commit for the item in angle brackets:

```
git checkout <sha value of the first commit> -b recovery
```
 - b) Look at the README file in your working directory – it should be empty now (*remember from git log comments that content wasn't added until the third commit*). Then look in the `doc/` directory; you notice two things of interest. First, you see a directory called `article_views/` and second, there's a file called `article_controller.sample.rb` – jackpot.
 - c) Now the trick is to transfer these items to their right places. Start with the controller by running the following command³:

```
cp doc/articles_controller.sample.rb app/controllers/articles_controller.rb
```
 - d) Next, create a directory for the views with the command⁴:

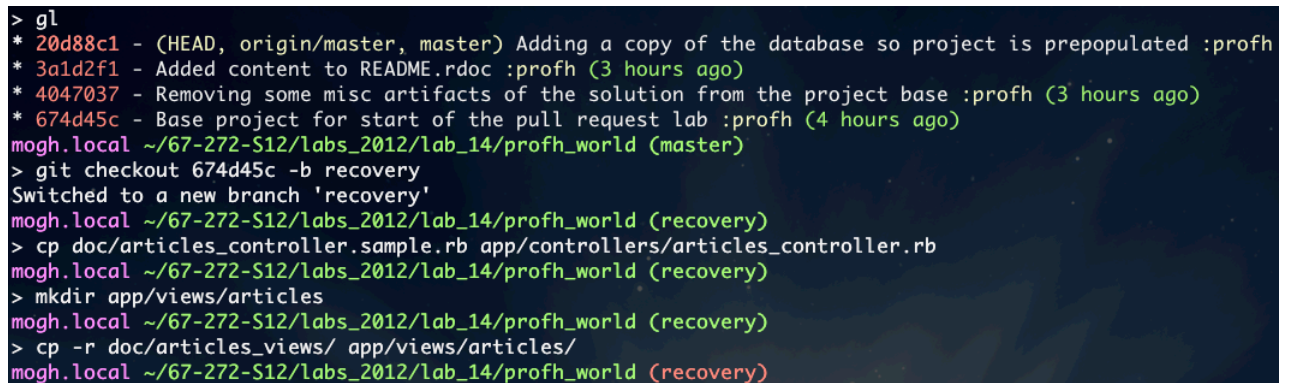
³ cp is the *nix command short for "copy". It takes in two arguments - the origin file, and the destination file. There are many flags, including the -r flag (used later), which means "recursively" copy.

⁴ mkdir is the *nix command short for "make directory"

`mkdir app/views/articles`. Now transfer the views into their proper directory with the command

```
cp -r doc/articles_views/ app/views/articles/
```

- e) Verify that this has been done correctly and then commit these changes to the recovery branch on git. Below is a screenshot of what we just did on the command line if you are unsure:



```
> gl
* 20d88c1 - (HEAD, origin/master, master) Adding a copy of the database so project is prepopulated :profh
* 3a1d2f1 - Added content to README.rdoc :profh (3 hours ago)
* 4047037 - Removing some misc artifacts of the solution from the project base :profh (3 hours ago)
* 674d45c - Base project for start of the pull request lab :profh (4 hours ago)
mogh.local ~/67-272-S12/labs_2012/lab_14/profh_world (master)
> git checkout 674d45c -b recovery
Switched to a new branch 'recovery'
mogh.local ~/67-272-S12/labs_2012/lab_14/profh_world (recovery)
> cp doc/articles_controller.sample.rb app/controllers/articles_controller.rb
mogh.local ~/67-272-S12/labs_2012/lab_14/profh_world (recovery)
> mkdir app/views/articles
mogh.local ~/67-272-S12/labs_2012/lab_14/profh_world (recovery)
> cp -r doc/articles_views/ app/views/articles/
mogh.local ~/67-272-S12/labs_2012/lab_14/profh_world (recovery)
```

10. **(Requester)** Now we need to merge these changes back into the articles branch, which is the branch we are using for our pull request. Do so now (*remember you have to go back to 'articles' branch first to make the merge*) and then fire up rails server and try out the new articles functionality. Going to `http://localhost:3000/articles` immediately runs into trouble. What is the error message telling you? This is an easy fix, but if you are stuck, you can look in the footnotes for help.⁵ Make the revision, verify it works, and then commit the change to the articles branch (which you should be on).
11. **(Requester)** Now that the article functionality is running properly it is time to resubmit the pull request. Back on github, go to your pull request on the original project (*not your page, the original project's page*) and see that your new commits are there and part of the pull request. Leave a comment for the maintainer explaining what you've done and how you've complied with his/her request and submit.

⁵ We remember that controllers and routes often go hand-in-hand. A controller action without a route will rarely get invoked while a route with a missing controller action will raise an exception. The easy fix is to add `resources :articles` to our `config/routes.rb` file to automatically generate our seven RESTful routes we need in this case.

12. (**Maintainer**) Go back to see if changes have been made to the pull request. Seeing that there are, review the changes and *verify that they work as advertised*. (If not, send a message explaining the errors encountered.) Once you have verified the article functionality has complete CRUD functionality, you can now merge these changes into your master branch by pushing the green **Merge pull request** button.
13. Have the TA check off that you have a project in your github account with article functionality and a record of the dialog back and forth between you and the requester AND that you have been a requester of another student who also has a revised repository – if so, you're done!

pltlh