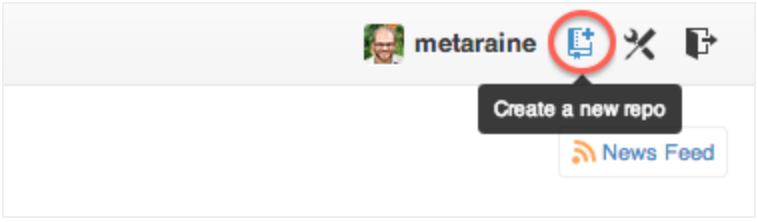
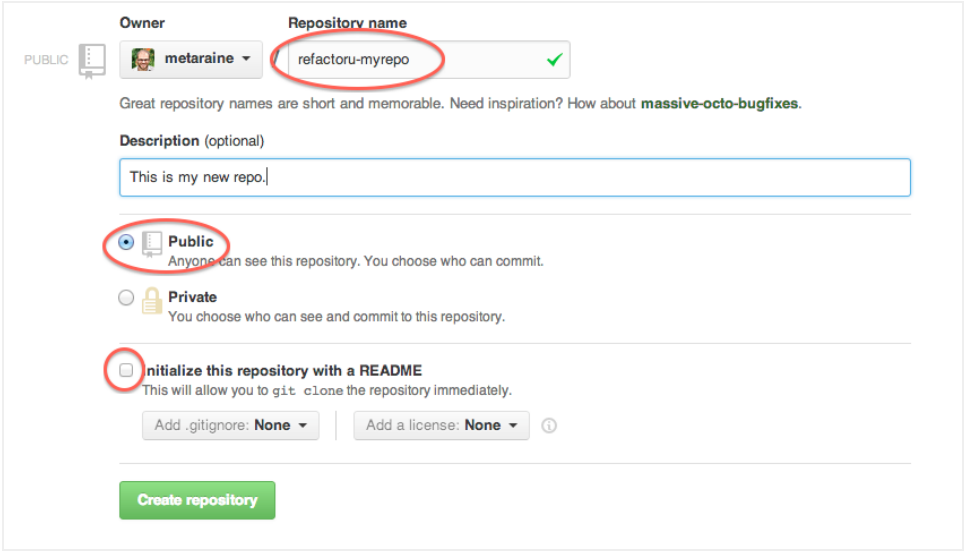
# How To Git

## Creating a new repository

Use this process to create a new repository that is stored locally and sync'd on github. A new repo should be created for each exercise.

### Steps

1. Log into github.com
2. Create a new repo for your exercise. Prefix the name with 'refactoru-'.
3. 

4. 

5. Open terminal and navigate to the directory where your exercise solution sits. e.g. `~/projects/js1-basics`.

6. Warning: Make sure you are located in the specific directory for your exercise before executing the next command, not your root ~/projects folder. Otherwise you will initialize the entire projects directory as a git repo! You can use the pwd command to print the current working directory in terminal.
7. Initialize the folder as a local git repo. (Behind the scenes this is simply creating a hidden .git folder storing the revision history and settings of your repo.)
8. `> git init`
9. Add all changes (including new files and deleted files) to the staging area (ready to be committed to the revision history)
10. `> git add -A`
11. Commit the staged changes to the revision history of your local repo.
12. `> git commit -m "first commit"`
13. Add a reference to the remote repo named 'origin'. This url should be clearly visible on github.com on the repo page you just created, so feel free to copy and paste.
14. `> git remote add origin https://github.com/YOUR_GITHUB_ID/REPO_NAME.git`
15. Push the commit history of your local repo to the remote repo. Adding `-u` adds 'upstream tracking' which means that future pushes or pulls in this repo require only the short commands `git push` or `git pull`.
16. `> git push -u origin master`
17. Note: If you get an error that says `! [rejected] master -> master (non-fast-forward)` then you need to pull in the remote changes before you push your local changes. Run the following command then resume step 8:
18. `> git pull --commit origin master`
19. Navigate to your newly created repo on github.com (or refresh it if it's already open from a previous step) and verify that it contains the newly added files.

## Making changes

If you want to make changes to an existing repo, the steps are much quicker!

1. After editing your files, add them to the staging area:

2. `> git add -A`
3. Commit the changes to the project history:
4. `> git commit -m "I added feature X"`
5. Push the changes to the remote repo:
6. `> git push`

# Status Commands

Ever wonder whether you've added files successfully? Whether they've been committed? Whether you have unpushed commits? The following commands are extremely useful for understanding the state of your local repository. Use these commands liberally in between other commands to know the state of your repo at all times.

| | |
|---|---|
| `git status` | Tells you what branch you are on, if there are uncommited changes (i.e y have modified local files but the changes have not been saved into yo revision history). All changes must be added to the staging area befor committing them using `git add -A`, so `git status` also tells you if you changes have been staged or not. |
| `git log` | Displays a list of all commits in the local version history |
| `git remote -v` | Displays a (verbose) list of all remotes attached to your local repo. thi indicates where you can push to or pull from. If you have a remote call origin you can push to origin with the command`git push origi master`. |
| `git branch` | Displays a list of all branches in the repo, with a star next to the curren checked out branch. |

# Exclude .DS_Store

These two steps will exclude .DS_Store from all future repos by setting a global exclude list. It will not remove .DS_Store files that have already been added to repos.

1. Specify a global exclusion list
2. `> git config --global core.excludesfile ~/.gitignore`
3. Add .DS_Store to the .gitignore file
4. `> echo .DS_Store >> ~/.gitignore`