

Git/GitHub

An Introduction

Version control

Git is a version control tool — it allows individuals and teams to work on a project while retaining control over the evolution of the project's codebase over time.

Git is:

- open source
- written in C, making it high-performing and very fast
- distributed:

You, the developer, clone the entire repository — the whole history of your project's source code — instead of just a snapshot of the latest version. This means that if there is a crash on the main server, there are many copies that survive.

Unlike many other VCSs, Git allows you to work locally - your ability to work is not dependent on network access or a central server. You only need to communicate with your team's development server when you are ready to add your contribution to the source code or clone the source code to use locally (ie, “push”, “pull”, “fetch”, or “clone”).

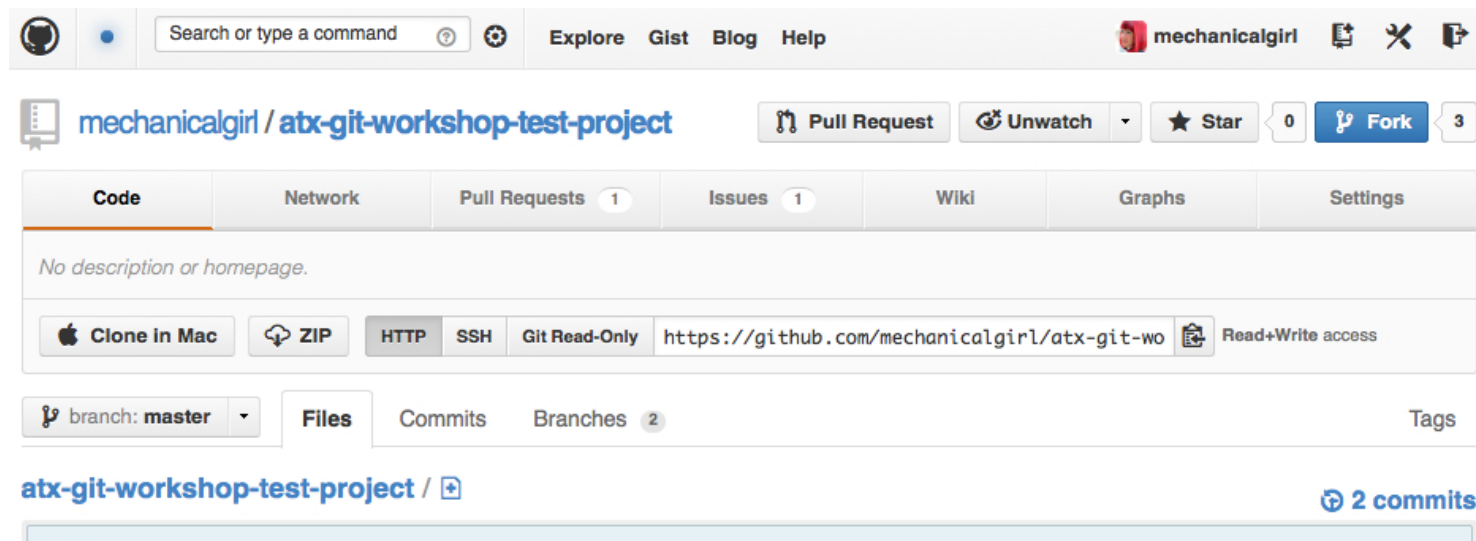
Typical workflow

1. Clone (copy) a repository to your local machine.
2. Make some changes to the code.
3. Commit those changes to your local copy of the repository.
4. Push those changes back up to the development server so they become part

Getting started

- Set up a Github account
- Fork this repository:

`github.com/mechanicalgirl/atx-git-workshop-test-project`



Getting started

Install git locally: <http://bit.ly/J6fdpy>

Mac:

```
$ sudo port  
install git-core +svn +doc  
+bash_completion +gitweb
```

Windows:

```
http://  
code.google.com/p/  
msysgit
```

Getting started

Open a terminal window,
create a folder where your Git projects will live,
then navigate to that folder:

```
$ cd ~
```

```
$ mkdir GitHub
```

```
$ cd GitHub
```

Cloning a repository

This brings a copy of the entire repository onto your local machine:

```
$ git clone  
https://github.com/{your username}/  
atx-git-workshop-test-project
```


Edit some code

Navigate into the working directory:

```
$ cd atx-git-workshop-test-project
```

Open the README file and add your name to the list:

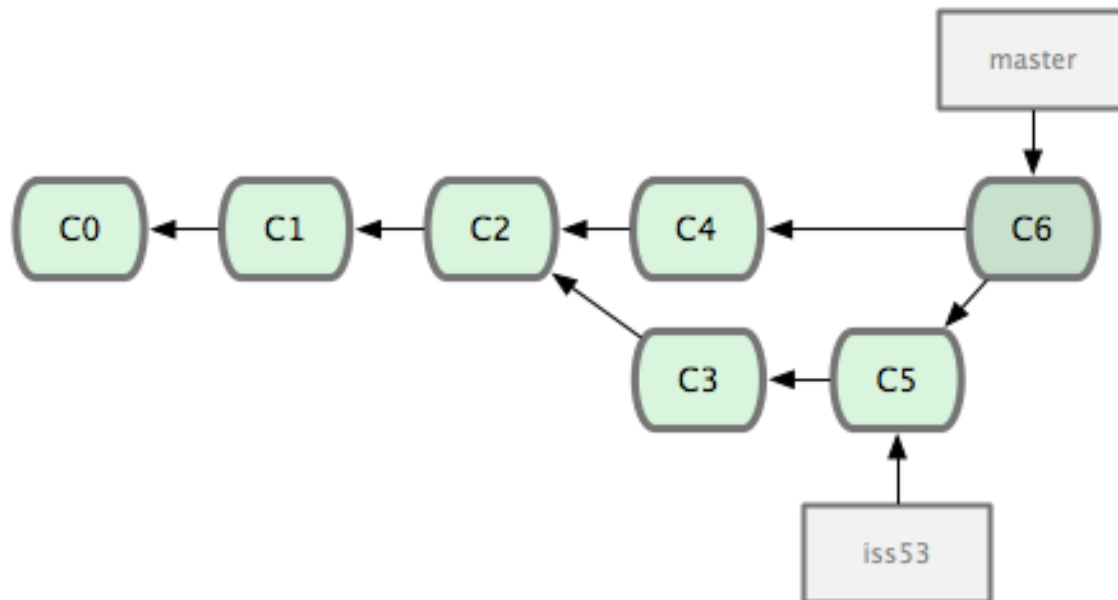
```
$ vi README.md
```

(you can also do this with the text editor of your choice)

Commit your changes

Before we actually commit that change,
let's take a look at a few other things you can do.

BRANCHES:



Commit your changes

In your terminal window, type this:

```
$ git branch
```

You should see something like this:

```
* master
```

This indicates that you're currently working out of the branch named "master".

Commit your changes

Create a new branch:

```
$ git checkout -b mynewbranch
```

The "-b" modifier on that command tells git that you're creating a brand new branch on this repository.

```
$ git branch
```

Now you should see that you've created and switched over to the new branch, but that the original "master" branch is still on your system:

```
master
* mynewbranch
```

Commit your changes

Go back to the master branch - to do that, you'll just run 'git checkout' again, but this time without that '-b' modifier:

```
$ git checkout master
```

Now that the new branch has been created, you can switch back and forth as much as you like:

```
$ git checkout mynewbranch  
$ git checkout master
```

Commit your changes

FILE STATES:

Changes you made to the README.md file still only exist locally and have not become a permanent part of the repository. Type:

```
$ git status
```

You should see something like this:

```
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   README.md
# no changes added to commit (use "git add" and/or "git commit -a")
```

Commit your changes

FILE STATES:

Git recognizes three different file states:

1. Modified: you have changed a file, but have not yet committed it to your db
2. Staged: you have marked a file in its current version to go into your next commit
3. Committed: the data is safely stored in your local database

Commit your changes

FILE STATES:

To take our modified "README.md" file and stage it, we're going to add the changed version to the repository:

```
$ git add README.md
```

Now run the status command again and you'll see a different output:

```
$ git status
```

```
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   README.md
#
```


Commit your changes

FILE STATES:

This changed file is now staged - let's go ahead and commit the change to our local repository:

```
$ git commit -m "test change" README.md
```

Now run the status command and you should see something like this:

```
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
# nothing to commit (working directory clean)
```

Commit your changes

FILE STATES:

A note about adds and commits - you can add/commit individual files if you need to, but you can also use wildcard characters and relative paths to add/commit multiple files at a time:

```
$ git add .  
# adds the current directory (.) and everything under it  
  
$ git add files/  
# add ./files, ./files/foo.txt, and ./files/foo/bar.txt  
  
$ git commit -m "my commit" *.html  
# commit all files in the current directory ending in .html
```

Examples:

```
$ git commit -m "replace all tables with divs" *.html  
$ git commit -m "change sort functionality" *.py
```

Push your changes

Inside your working folder, you should see a directory named ".git" - this folder and its contents were created when you checked out (cloned) the repository.

Let's take a look at the config file inside this folder:

```
cat .git/config      (or use the text editor of your choice)
```

```
[core]
  repositoryformatversion = 0
  filemode = true
  bare = false
  logallrefupdates = true
  ignorecase = true
[remote "origin"]
  fetch = +refs/heads/*:refs/remotes/origin/*
  url = https://github.com/mechanicalgirl/atx-git-workshop-test-project
[branch "master"]
  remote = origin
  merge = refs/heads/master
```

Push your changes

```
[core]
  repositoryformatversion = 0
  filemode = true
  bare = false
  logallrefupdates = true
  ignorecase = true
[remote "origin"]
  fetch = +refs/heads/*:refs/remotes/origin/*
  url = https://github.com/.../atx-git-workshop-test-project
[branch "master"]
  remote = origin
  merge = refs/heads/master
```

The [core] section is unrelated to git remotes, so we'll ignore that for now.

The [remote "origin"] and [branch "master"] sections are related to how git will interact with remote repositories.

The [remote "origin"] section refers to the address of the remote repository. The name "origin" is basically shorthand for the full repository address.

This is important because "origin" is the name you'll be using to refer to the repository whenever you pull/push (check changes in or out).

For example, if you changed that entry to '[remote "austin"]', you would use the name "austin" for your pull/push commands.

<http://www.gitguys.com/topics/the-configuration-file-remote-section/>
<http://www.gitguys.com/topics/the-configuration-file-branch-section/>

Push your changes

Now push those committed files to the central repository.:

```
$ git push origin master
```

As we mentioned earlier, "origin" is the shorthand name for the repository address. "master" refers to the branch you're pushing (in other words, if you had made changes in the new branch you created, that branch name is what you'd be using here).

Git will ask for your GitHub username and password.

Once your file change is successfully pushed, you'll see a response like this:

```
Counting objects: 5, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 302 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/.../atx-git-workshop-test-project
2cf3ffa..311ab22  master -> master
```

When you return to the repository on GitHub, you'll see your change reflected there:

```
https://github.com/.../atx-git-workshop-test-project
```

Extras: merge

Merge branches:

When you are finished working on a branch and are ready to merge it back into its parent (say the develop branch or the master branch), you must merge from within the parent branch:

```
$ git checkout master  
# Makes "master" the active branch
```

```
$ git merge branchName  
# Merges commits from branchName to master
```

Delete a branch if/when done with it:

```
$ git branch -d branchName  
# deletes branchName branch
```

More on basic branching and merging:

<http://git-scm.com/book/en/Git-Branching-Basic-Branching-and-Merging>

Extras: pull vs. fetch

There are two ways to get commits from a remote repo or branch — fetch and pull.

fetch

This command is useful if you need to keep your repo up to date but are working on something that might break if you update your files.

Fetch retrieves any changes that you do not have from the target remote repository, but does not merge them with your current branch.

To integrate the commits into your local branch, you use "git merge," which combines the specified branches and prompts you if there are any conflicts.

```
$ git fetch upstream
# pulls in any new changes
# not present in your local repo
# without modifying your files
```

pull

In "git pull", git tries to do your work for you.

It is context sensitive, so Git will merge any pulled commits into the branch you are currently working in.

Because "git pull" automatically merges the commits without letting you review them first, if you don't closely manage your branches you may run into conflicts.

```
$ git pull upstream
# pulls commits from 'upstream' and
# merges them in active branch
```

Extras: diff

Compare files:

Output a description of the changes that are staged or modified and unstaged with the `diff` command.

```
$ git diff
# Display code that has been changed since last commit,
# but that has not yet been staged.
# Adding '--cached' at the end will show you staged changes.
```

```
$ git checkout my_new_branch
$ git diff --name-status origin/master
# To preview file changes between branches before you merge
```

```
$ git diff master mynewbranch -- index.html
$ git diff mynewbranch master -- index.html
# To preview differences in a specific file between branches
```


Extras: log

See a log of commits:

There are several ways to see logs of activity on your repository - the most commonly used is:

```
$ git log
```

You should see something like:

```
commit 52191b83cf7856afd06879eec7dad67c0016f63d
Author: Barbara Shaurette <barbara.shaurette@gmail.com>
Date:   Wed Jan 16 14:59:54 2013 -0600
```

```
test
```

```
commit 2cf3ffa5d5114a6f5b21bdcdb04aa4d64901aa85
Author: Barbara Shaurette <barbara.shaurette@gmail.com>
Date:   Wed Jan 16 10:26:10 2013 -0800
```

```
Initial commit
```

The parts of this log are fairly self-explanatory - a SHA-1 commit number, the name of the user and date on the commit, and the commit message.

Extras: reflog

Another logging command, commonly used for resetting a repository and recovering lost commits, is:

```
$ git reflog
```

It returns something that looks like this:

```
52191b8 HEAD@{0}: commit: test
2cf3ffa HEAD@{1}: checkout: moving from master to mynewbranch
311ab22 HEAD@{2}: checkout: moving from mynewbranch to master
2cf3ffa HEAD@{3}: checkout: moving from 2cf3ffa5d5114a6f5b21bdcdb04aa4d64901aa85 to mynewbranch
2cf3ffa HEAD@{4}: clone: from https://github.com/.../atx-git-workshop-test-project
```

The first column is an unencrypted commit number.

The second column is the state of HEAD (a pointer to the top of the current branch).

The third column contains a description of the repo activity at that point.

To learn more about how git reflog can be used to recover lost commits, take a look at these blog posts:

<http://effectif.com/git/recovering-lost-git-commits>

More reading:

Branching Workflows

<http://git-scm.com/book/en/Git-Branching-Branching-Workflows>

Distributed Workflows

<http://git-scm.com/book/en/Distributed-Git-Distributed-Workflows>

Custom git configuration:

<http://git-scm.com/book/en/Customizing-Git-Git-Configuration>

Other resources:

<http://git-scm.com/book>

Using GitHub

GitHub is a web-based hosting service for projects that use the Git revision control system.

Basically, it's a collection of repositories, but they've thrown a nice visual interface on top of it to give you features like:

- shortcuts into basic git commands, like cloning and pull requests
- the ability to join teams, find projects, and follow users and projects easily
- a simple way to watch for changes on projects you follow

Using GitHub

- Fork a project - We've done that already
- Make a change locally - Done
- Commit that change to your own version of the project - Done

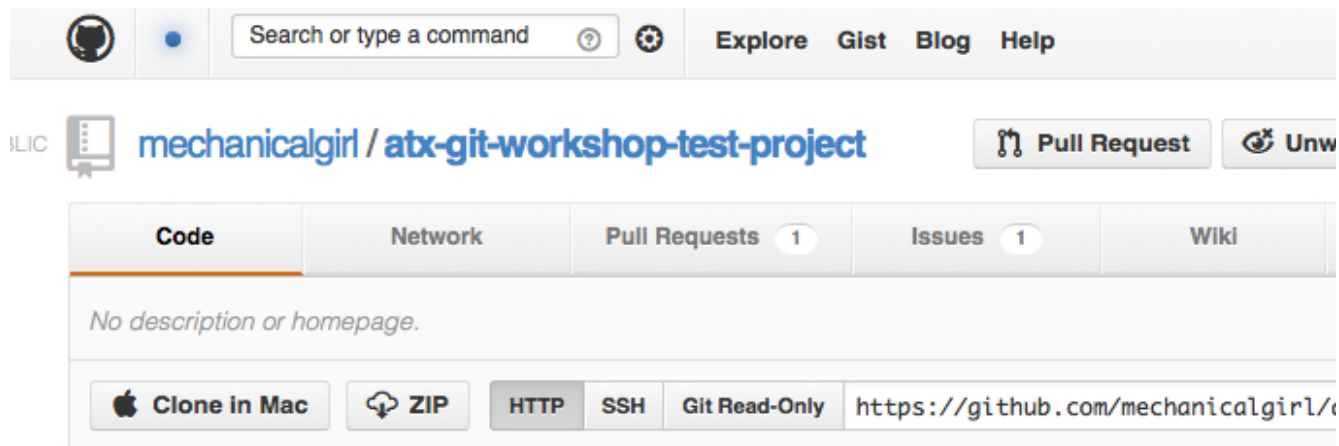
Pull requests

Make a request to the owner of a repository to merge your changes in:

1. Navigate to the main repository on the original account:

`https://github.com/mechanicalgirl/atx-git-workshop-test-project`

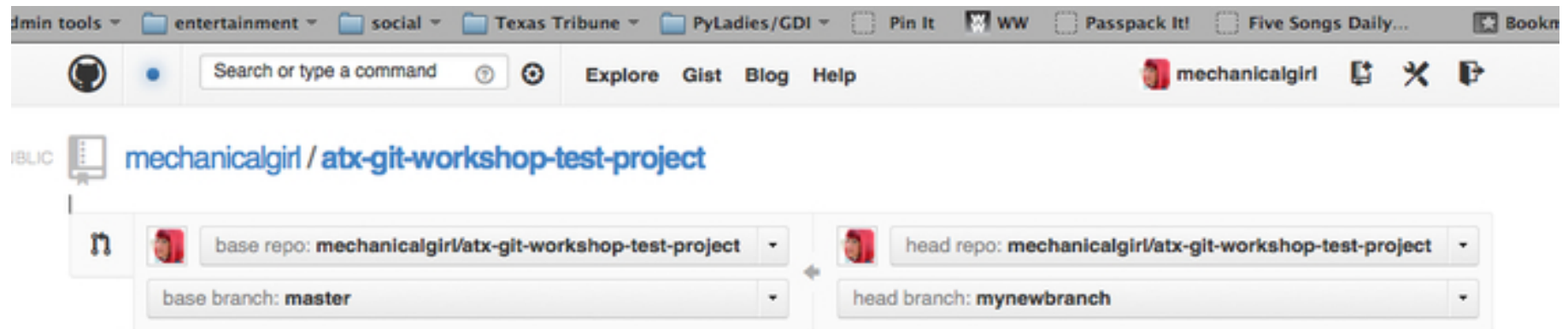
2. Click on the "Pull Request" button on the top nav bar



Pull requests

3. The request should show:

- target repository on the lefthand side
- source repository on the righthand side



Pull requests

4. Enter a brief description and submit the request - this creates a page that allows the manager of the repository to review the changes, accept/reject the code, and leave notes:

The screenshot displays a GitHub pull request page. At the top, a green 'Open' button is followed by the text 'mechanicalgirl wants to merge 1 commit into master from mynewbranch'. To the right, there is a progress bar with 4 green squares and a red square, and a '#1' label. Below this, there are tabs for 'Discussion', 'Commits 1', and 'Files Changed 1'. The 'Files Changed' tab is active, showing a summary: 'Showing 1 changed file with 3 additions and 1 deletion.' A 'Show Diff Stats' button is on the right. The main area shows a diff for 'README.md' with a 'View file @ 52191b8' button. The diff shows a deletion of a line and additions of three lines. A tip box at the bottom explains how to add notes to lines in a file.

Open mechanicalgirl wants to merge 1 commit into **master** from **mynewbranch** 4 #1

Discussion Commits 1 **Files Changed** 1

Showing 1 changed file with 3 additions and 1 deletion. **Show Diff Stats**

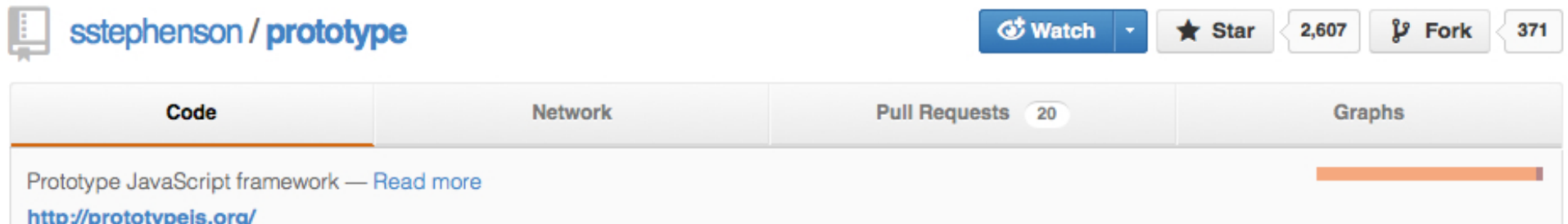
4 README.md **View file @ 52191b8**

```
...  ...  @@ -1,2 +1,4 @@
1  1  atx-git-workshop-test-project
2  2  -----
   \ No newline at end of file
2  2  +-----
3  3  +
4  4  +test test test
```

38 58 **Tip:** You can add notes to lines in a file.
39 59 Hover to the left of a line to make a note
40 60

Watch and contribute

If there's a project you use regularly (such as the Django framework) and you want to keep an eye on changes as they are committed, you can use the 'watch' feature on GitHub.




Watch and contribute




Once you've followed a few projects, you'll see a feed like this on your GitHub home page:


When you're ready to contribute to one of these projects, you'll probably go through the fork/pull request process we just learned about.




News Feed




Pull Requests



 [mlavin](#) starred [nvie/pip-tools](#) an hour ago

 2 hours ago
[twig](#) opened pull request [django/django#681](#)
 Users with unsalted MD5 passwords unable to log in with Django 1.4
 1 commit with 1 addition and 1 deletion

 [coderanger](#) starred [nvie/pip-tools](#) 2 hours ago

 2 hours ago
[erikrose](#) pushed to [testing](#) at [mozilla/dxr](#)
  [c989397](#) Fix indentation to conform to Mozilla's (and the greater communit...

 3 hours ago
[abbeyj](#) opened pull request [mozilla/dxr#81](#)
 Fix member: queries for classes/structs that only contain member variables or member functions.
 2 commits with 37 additions and 12 deletions

 3 hours ago
[charettes](#) commented on pull request [django/django#679](#)
 [@jonashaag](#) `min` and `max` values are only suitable for an input of type `"number"`. However, we can't use this input type when dealing with localized fi...

Gists

A 'gist' is a simple way to share code snippets with others.

All gists are git repositories, so they are automatically versioned and forkable.

The screenshot displays a GitHub Gist interface. At the top, there's a search bar and a 'Discover Gists' button. The user 'mechanicalgirl' is logged in. The gist is titled 'SECRET' and is a public gist. It was created 6 hours ago. The gist is named 'gistfile1.txt' and contains Python code for a Django form. The code defines a class 'ImageAdminForm' that inherits from 'UniqueSEOSlugForm'. It has a 'Meta' class with 'model = Image'. The '.__init__' method checks for an 'initial' value in 'kwargs' and sets defaults for 'pub_date', 'publication_status', 'authors', and 'tags'. The code also includes a comment about adding a 'multiple' attribute to the input field.

Gist Detail

Revisions 1

Download Gist

Clone this gist
<https://gist.github.com/mechanicalgirl/320b48b142ba2060c286>

Embed this gist
`<script src="https://gist.github.com/mechanicalgirl/320b48b142ba2060c286.js"></script>`

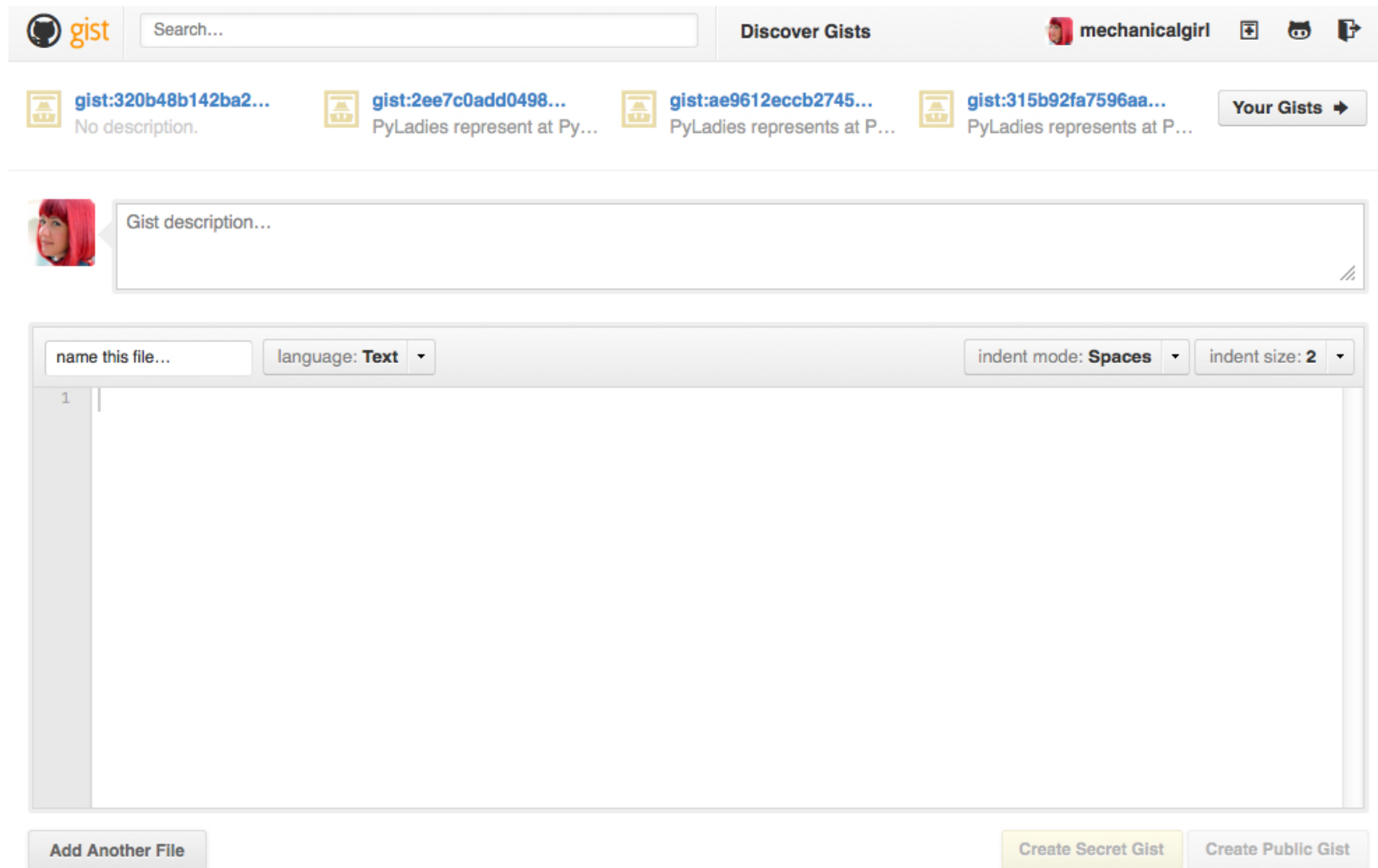
Link to this gist
<https://gist.github.com/mechanicalgirl/320b48b142ba2060c286>

```
1 # forms.py
2
3 class ImageAdminForm(UniqueSEOSlugForm):
4     class Meta:
5         model = Image
6
7     def __init__(self, *args, **kwargs):
8         if 'initial' in kwargs and kwargs['initial'] is not None:
9             defaults = {
10                 'pub_date': datetime.datetime.now(),
11                 'publication_status': 'P',
12                 'authors': (1,), # Admin
13                 'tags': (2414,), # The Texas Tribune | Company
14             }
15             defaults.update(kwargs['initial'])
16             kwargs['initial'] = defaults
17             super(ImageAdminForm, self).__init__(*args, **kwargs)
18
19 # This is the only thing I've added to the form - the 'multiple' attribute on the input,
20 # so that I can get multiple files into the request to test with
21 self.fields['image'].widget = forms.FileInput(attrs={'size': '30', 'multiple': ''})
```

Gists

A gist can be:

- public: Anyone can see it.
- secret: Only someone with the link can get to it.



The screenshot shows the GitHub Gist creation page. At the top, there's a header with the GitHub logo, a search bar, and a 'Discover Gists' button. Below the header, there's a row of four gist cards, each with a file icon, a gist ID, and a description. To the right of these cards is a 'Your Gists' button. Below the cards, there's a section for the new gist. It starts with a profile picture of a person with red hair and a text input field labeled 'Gist description...'. Below this is a large text area for the code. Above the text area, there are three dropdown menus: 'name this file...', 'language: Text', and 'indent mode: Spaces'. To the right of these is another dropdown menu for 'indent size: 2'. At the bottom of the page, there are three buttons: 'Add Another File', 'Create Secret Gist', and 'Create Public Gist'.

gist Search... Discover Gists mechanicalgirl

gist:320b48b142ba2... No description. gist:2ee7c0add0498... PyLadies represent at Py... gist:ae9612eccb2745... PyLadies represents at P... gist:315b92fa7596aa... PyLadies represents at P... Your Gists

Gist description...

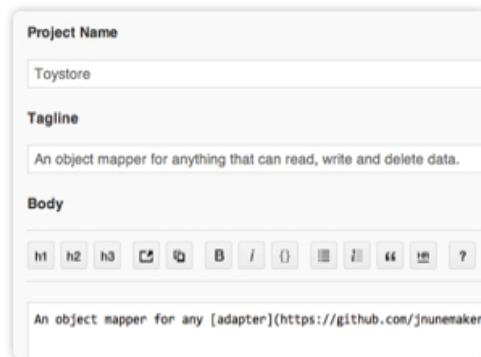
name this file... language: Text indent mode: Spaces indent size: 2

1

Add Another File Create Secret Gist Create Public Gist

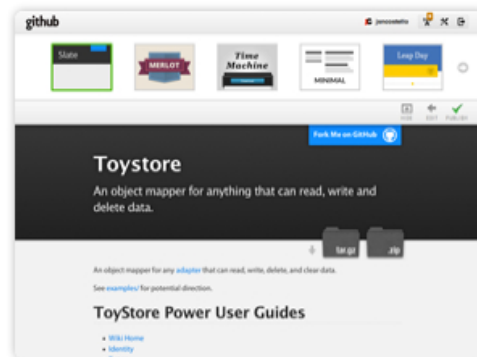
Pages

Create Project Pages In 3 Steps

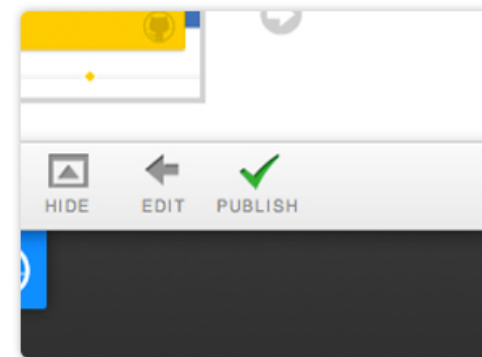


The screenshot shows the 'Project Name' form in the GitHub interface. It has three main sections: 'Project Name' with a text input containing 'Toystore', 'Tagline' with a text input containing 'An object mapper for anything that can read, write and delete data.', and 'Body' with a rich text editor. The rich text editor has a toolbar with icons for bold, italic, link, and other formatting options. The body text is 'An object mapper for any [adapter](https://github.com/jnunemaker'.

Author



Theme



Publish



Free Hosting

GitHub Pages are hosted free and easily published through our site, the GitHub for Mac app, or from the command line. Manage your site's content from GitHub using the tools and workflow that you're familiar with, so you won't skip a beat. More about publishing with [GitHub Pages](#).

Page Generation with Themes

If you're creating a page for your project, check out our automatic Project Page generator. It lets you to author your page content in Markdown and toggle through our selection of designer themes. Many of our themes are responsive and include layouts optimized for mobile, so your page looks great on any device.

Manual Pages and Jekyll

There are a few ways to create GitHub Pages, including manually pushing html or a Jekyll site. You can easily redirect your page to a custom url. To read more about creating Pages manually with Jekyll, read the [documentation here](#).

<http://pages.github.com>