

Markdown and GitHub

CSCI 5828: Foundations of Software Engineering
Lecture 04

Lecture Goals

Present an introduction to Markdown and GitHub

Purpose: Getting Ready for the Essays

We're asking that all essays this semester be uploaded to GitHub

That means you need to be comfortable with the following technologies

- Git
- Markdown
- GitHub

In Lecture 3, we presented an introduction to git

Now, let's continue and learn about Markdown and GitHub

Markdown

Markdown is a mark-up language created by John Gruber in 2004.

The spec has been available from his website, Daring Fireball, ever since

<<http://daringfireball.net/projects/markdown/>>

He describes it like this -

Markdown is a text-to-HTML conversion tool for web writers. Markdown allows you to write using an easy-to-read, easy-to-write plain text format, then convert it to structurally valid XHTML (or HTML).

Thus, “Markdown” is two things: (1) a plain text formatting syntax; and ...

The overriding design goal for Markdown’s formatting syntax is to make it as readable as possible. The idea is that a Markdown-formatted document should be publishable as-is, as plain text, without looking like it’s been marked up with tags or formatting instructions. While Markdown’s syntax has been influenced by several existing text-to-HTML filters, the single biggest source of inspiration for Markdown’s syntax is the format of plain text email.

Markdown is Everywhere

Markdown proved to be very popular

- Support for it has popped up in all sorts of places
 - There are dedicated markdown editors; GitHub uses it extensively
 - There are many blogging systems that will accept markdown-formatted articles and convert them to HTML automatically
 - Indeed, all of the Daring Fireball website is written in Markdown • For instance, go to this URL: <<http://daringfireball.net/linked/2015/08/28/panzerino-twotter>>
 - Then change the URL to this:
<<http://daringfireball.net/linked/2015/08/28/panzerino-twotter.text>>
- Markdown is thus a project that “scratched an itch” for John Gruber. It is an example of “eating your own dog food”: using your own software daily

Trying out the Examples

Note: You can try out any of the examples shown in this lecture

Head to <<http://daringfireball.net/projects/markdown/dingus>>

You can also go to GitHub -

- Create a new repo and ask that it have a README.md file created
- Edit the README.md file to contain any Markdown you want
 - and then flip over to the “Preview changes” tab
- Each time you switch to that tab, Github converts the Markdown that you wrote to HTML and displays the results

The Basics (I)

Paragraphs

- A paragraph is one or more consecutive lines of text
- To create a new paragraph, separate it from the previous paragraph with a blank line
- `<p>`When converting to HTML, the paragraph is wrapped in a p tag`</p>` •

Headings

- You can specify headings in one of two ways; I'll cover one of them
- To specify that a line is a heading, start it with a hash mark (#)
- A heading line can start with 1 to 6 hash marks, corresponding to HTML's h1, h2, h3, h4, h5, and h6 tags.

The Basics (II)

Blockquotes

- If you need to indicate a quotation in your document, use the notation that has been used in plain-text e-mail since the dawn of time
 - that is, start each line with a “greater than” (>) symbol
- Blockquotes can be nested and can contain other markdown elements

> # This is a heading inside of a blockquote.

>

> Here's the text of paragraph 1 in the blockquote

>

> > Here's a quote within a quote

>

> Here's the text of paragraph 2 in the blockquote

The Basics (III)

Emphasis

- To emphasize a phrase, surround it with asterisks or underscores
 - I **really** want you to be there
 - No, I really do.
- If you strongly want to emphasize something, then double the symbols
 - Listen, I ****really**** want you to be there
 - I'm not kidding.

In HTML, the former will be translated as an em tag; the latter as a strong tag

- By convention, em tags appear in italics while strong tags appear in bold

The Basics (IV)

Lists

- You can create two types of lists (just like in HTML)
- Unordered lists: use *, +, or - followed by a space
- Ordered lists: use numbers followed by periods and a space.

* This is a list

* Items can have

multiple paragraphs
if you need them
just indent by four spaces

and have blank lines between items

* Here's the end of the list

Here's the start of a new paragraph

1. Here's an ordered list that follows
2. with multiple items
3. as many as you need

The Basics (V)

Embedded Lists

- You can embed one list inside another, as many levels deep as you need (within reason). Just prefix each level with four spaces in front of the list marker. Level 1 is a normal list. Level 2 lists have four spaces at the start of their lines. Level 3 lists have eight spaces at the start of their lines, etc.

* This is a list

* This is an embedded list

1. that is indicated by adding
2. four spaces in front of the list marker

* Go as deep as you need.

* This is the second item of the outer list

* And this is the third item

This is a paragraph between lists.

1. For ordered lists, you can help yourself remember how to embed lists by putting TWO spaces after your list marker. This only works for SHORT lists!

1. This starts a new embedded list by putting the list marker under the text of the previous bullet which just happens to then have four spaces in front of it!

2. Here's a second element

5. Here's yet another embedded list.

2. And here's the second element of the outer list.

The Basics (VI)

Code Elements and Code Blocks

- You can have “code” elements either in-line or in a stand-alone block
- In-line code is indicated by surrounding the word or phrase with back ticks

Be sure to call the ``init()`` method before you call ``doMyWorkForMe()``

- Stand-alone code blocks are indicated by four spaces at the start of the line

```
def check(dir: Path) = {  
  if (!exists(dir)) {  
    error(s"Directory: <$dir> does not exist.")  
  }  
}
```

The Basics (VII)

Links

- Markdown supports two styles of links. I'll show you the most common one
- To create a link, you indicate the text of the link, and then the link itself • The text goes in square brackets, the link goes in parentheses

Check out [GitUp](http://gitup.co) for a cool way to view git repos.

- You can add a link title if you want

Check out [GitUp](http://gitup.co "It's cool!") for a cool way to view git repos.

Link titles appear as a tool-tip in web browsers when you hover over a link.

They also increase the accessibility of your page as they can be processed by screen readers and other similar tools.

The Basics (VIII)

Images

- Images have a similar syntax to links.
 - `![alt text](href "title")`
- For example
 - `![Kitties!](https://kenbod.github.io/5828_S18/images/pounce.jpg "So cute!")`

Just FYI

To make the example work on the previous page, I used a feature of GitHub that allows you to publish a website from a repository on GitHub by creating a branch called gh-pages.

HTML Allowed (I)

Markdown's reason for existence is to take text and convert it to HTML

- As a result, if there is something that HTML allows but Markdown doesn't support, just insert the HTML directly into the Markdown

This applies to block level and in-line elements of HTML

- See <http://daringfireball.net/projects/markdown/syntax> for details

For instance, the official version of Markdown does not support tables

- Note: GitHub's version (or "flavor") of Markdown does

So, in the official version of Markdown, if you need a table just add it

- See example next slide

HTML Allowed (II)

This is a `Markdown` paragraph

```
<table>
<tbody>
  <tr>
    <td>Row 1, Column 1</td>
    <td>Row 1, Column 2</td>
  </tr>
</tbody>
</table>
```

This is a second Markdown paragraph

Why?

Why did we present this information?

- Because GitHub uses Markdown everywhere and even adds new features to Markdown

What does GitHub add to Markdown? <https://help.github.com/articles/github-flavored-markdown/>

- URL auto-linking
- Strikethrough text
- Fenced code blocks and syntax highlighting
- Tables

Click on the link above for more information

GitHub



GitHub

GitHub is a Web-based repository hosting service for git

- You can upload your repositories to it and then access/manipulate them with a nice Web-based interface for many of the most common commands
 - (as many public repositories as you want; you have to pay to keep your repositories private)
- It then adds new services on top of git that are designed for collaboration
 - access control
 - issue tracking
 - notifications
 - pull requests

Brief Introduction to GitHub

I will not attempt to present a comprehensive introduction to GitHub

- Features that I don't cover can become the topic of YOUR essays 😊
- Take a look at: <<https://github.com/features>> for more info

Instead, we'll cover

- how you can use GitHub as a remote copy of a local repository
 - This will allow us to explore the pull and push commands
- how you can use GitHub to serve your essays to the world

Assumptions

I assume that you

- have a GitHub account
 - you will need one for this class (so please create one, if needed)
- have followed the “[Set Up Git](#)” instructions at GitHub [Bootcamp](#)

When you issue “git clone”, “git fetch”, “git pull”, and “git push” requests, GitHub will ask you for your password or passphrase

- If you don't want to type that each time, you will need to
 - a) [create/upload your public key file to your GitHub profile](#), or
 - b) store your [GitHub password in a credential helper](#)

Linking Repositories (I)

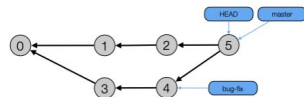
The first thing that needs to happen to work with Github is linking a repository on your computer to one that exists on Github

There are two primary ways to do this

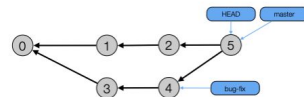
- 1. If the repo exists on GitHub and not on your computer
 - `git clone <URI>`
 - GitHub makes this easy by letting you cut-and-paste the URI from the repo's home page

Big Picture View (I)

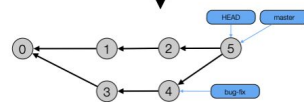
Use “git clone” to copy a repo to your machine



**Use “git clone”
to copy a repo
to your machine**



git clone



The Link

When you do this, git clone configures the repository to remember where it came from

- You can verify this with the git remote command
 - For one of my GitHub-based repos, the git remote -v command lists
 - origin git@github.com:coloradocollective/5828_S21.git (fetch)
 - origin git@github.com:coloradocollective/5828_S21.git (push)

This says that my local repository is associated with a remote copy called “origin” (a standard convention) that is located on GitHub.

- My repository can be associated with **any number** of other repositories
- On a local-only repo, git remote -v produces no output

Linking Repositories (II)

Going back to linking repositories... the other primary way of doing this is:

- 2. If the repo exists on your computer but not on GitHub then
 - Create an empty repo on Github:
 - Fill out the resulting dialog (see next slide)

LecturLecturesHomekenbo86-020.pCSCiGitHuCre

https://github.com/

Search


Search or jump to...Pull requestsIssuesMarketplaceExplore

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Owner

Repository name *


 coloradocollective

 /


5828_S21

Great repository names are short and memorable. Need inspiration? How about **ideal-eureka**?

Description (optional)

☐  **Public**

Anyone can see this repository. You choose who can commit.

☒  **Private**

You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer.

Add .gitignore: **None**

Add a license: **None**

Create repository

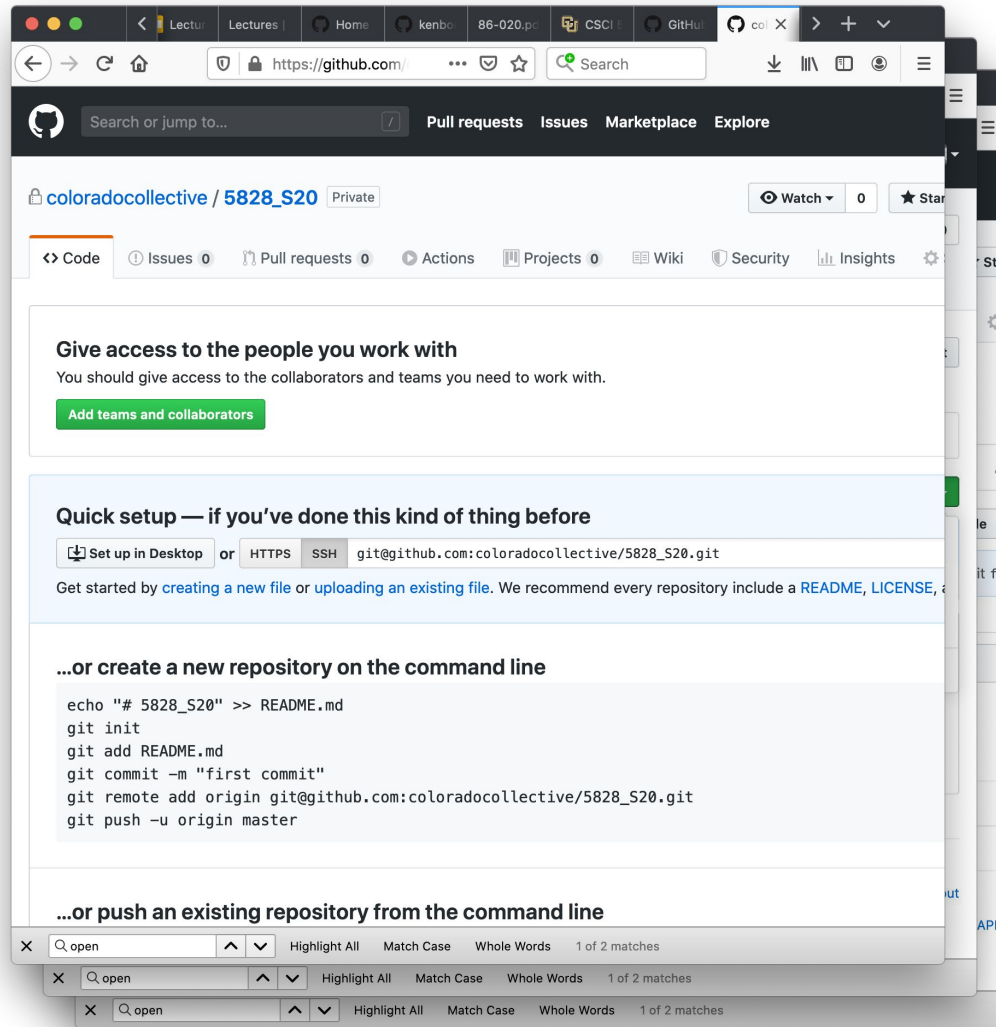
open

Highlight AllMatch CaseWhole Words1 of 2 matches

Linking Repositories (III)

After the empty repository has been created, GitHub provides helpful instructions on how to establish the link (see next slide)

- As an example, I took the (very) simple repo I created in the Git lecture and uploaded it to GitHub
 - I named my new empty repo on GitHub “5828_S21” to match what I called the repo locally.



Linking Repositories (III)

Things to note:

`git remote` command first used to add a remote repository, then used to list our local repository's remotes.

`git push` is used to push the contents of our master branch to the remote repo and to configure the local branch to “track” the remote

`git branch -a` is used to list all branches including the ones on our remote

We have a local branch called “bug-fix” that did NOT get copied to the remote

Establishing the link

`git clone`

- When you clone a repository, it automatically creates a local branch for each remote branch and sets up a “tracking relationship” between them
 - A local branch that “tracks” a remote branch will allow “git pull” commands to copy commits from the remote branch that were added in some way (typically pushed to that branch by one of your collaborators)

`git push -u <repo> <refspec>`

- This is the other way to establish a link between a local branch and a remote one; in this case, you’re simultaneously pushing the contents of a local branch (thus creating the branch on the remote repository) and setting up the tracking relationship

git push (I)

We are just scraping the surface of the “git push” command

The generic form of the command is

- `git push <options> <repo> <refspec>`

The `<repo>` argument ALWAYS refers to a remote repository

- If you don't specify it, then 'origin' is assumed

`<refspec>` is a placeholder for this beast: `(+)<src_ref>:<dst_ref>`

- `src_ref` is a refspec that references your local repo; `dst_ref` is for the remote
- If you just list a single branch name (like we did), it is short for
 - `<src_branch>:<dst_branch>` or, in our case, `master:master`
- If you don't specify the `<refspec>`, your current branch is assumed

git push (II)

Putting it all together

- `git push -u origin master`

means: push the contents of the master branch of MY repository to the master branch of the REMOTE repository (creating the branch, if needed) AND set up a tracking relationship between them

After we set-up the tracking relationship, we can push new commits to master to the remote copy of master using ANY of these commands

- `git push origin master:master`
- `git push origin master`
- `git push` (assuming that master is your current branch)

Pull before you Push (I)

If you have new commits that you want to push to the remote but someone else already pushed THEIR new commits to the remote, you will need to pull them in first using ANY of these commands

- `git pull origin master`
- `git pull`

`git pull` is a “short cut” in that it does the following thing

- `git fetch <args>`: pulling down the latest info about the remote
- `git merge <remote-branch>`: this merges the changes of the remote branch into your current local branch

Note: you can just do “`git fetch`” and then do the merge yourself if you want

Pull before you Push (II)

Regardless, once you have pulled, you can resolve any conflicts that might have popped up and then you can push your own commits to the remote branch

Finally, going back to our example, if we want our “bug-fix” branch from our local repository to be stored on the remote, then we can just push it using the same command that we did before

- Here's how
- `git checkout bug-fix`
- `git push <==` fails, because remote branch doesn't exist yet
- `git push -u origin bug-fix <==` works!

Using GitHub for our Essays



Using GitHub's Wiki Feature

Starting week 6, we are going to use GitHub's Wiki Feature to submit our essays

https://github.com/coloradocollective/5828_S21/wiki

Overview

Each team should select software engineering topics and techniques related to their project “stack” - and then write a short essay on that topic (2 to 4 paragraphs)

Go to the wiki, create a team page, then create a team page (named after your topic), and use Markdown to enter information, links, code, images, etc.

Essays are due on Wednesday by 11:59 PM

On Thursday, find three essays from other students and add a review

Reviews are due on Sunday by 11:59 PM

Essay topics

Topics

“Name of stack”

Backlog

Language

Build tool

Testing tools

Back end framework

Front end framework

Data persistence

Queues/Messaging

Production environment

Continuous integration

Production monitoring

Continuous delivery

Examples

(Rails, Spring, Kotlin)

(Tracker, Trello, etc)

(Java, Kotlin, Ruby, Python, etc)

(Gradle, maven, etc.)

(JUnit, Mockito, etc)

(Spring, Ktor, Dropwizard, Rails, etc)

(Angular, React, etc)

(Postgresql, Cassandra, etc)

(Redis, Kafka, etc.)

(K8s, Istio, Heroku, GCP, AWS, etc)

(Jenkins, Travis, TeamCity, etc)

(New Relic, Graphite, etc)

(Spinnaker, Jenkins, etc.)

Essay topics

Topics

“Name of stack”

Backlog

Language

Build tool

Testing tools

Back end framework

Front end framework

Data persistence

Queues/Messaging

Production environment

Continuous integration

Production monitoring

Continuous delivery

Examples

“Rails stack”

Tracker

Ruby

Bundler, rbenv

Rspec

Rails

Angular

Postgresql

Redis

Heroku

Travis

New Relic

Travis

Questions?