

# Markdown & GitHub

---

CSCI 5828: Foundations of Software Engineering  
Lecture 3 — 09/01/2015

# Lecture Goals

---

- Present a brief introduction to Markdown and GitHub
- Present examples of the types of presentations I want you to create

# Purpose: Getting Ready for the Presentations

---

- I'm asking that all presentations this semester be uploaded to GitHub
  - That means you need to be comfortable with the following technologies
    - `git`
    - Markdown
    - GitHub
- Last Thursday and today, I presented an introduction to `git`
- Now, let's continue and learn about Markdown and GitHub
  - I will then bring it all together and show you some example presentations

# 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/>](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 it to this URL: <<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

# 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 backticks

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!](http://www.pluspets.net/wp-content/uploads/2011/01/Best-Cats-Photos8.jpg "So cute!")`

# 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 `<a href="http://daringfireball.net/projects/markdown/">Markdown</a>` 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 I present this information?
  - Because GitHub uses Markdown everywhere and even adds new features to Markdown
- What does GitHub add to 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** presentations 😊
  - Take a look at: <<https://github.com/features>> for more info
- Instead, I'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 presentations to the world

# Assumptions

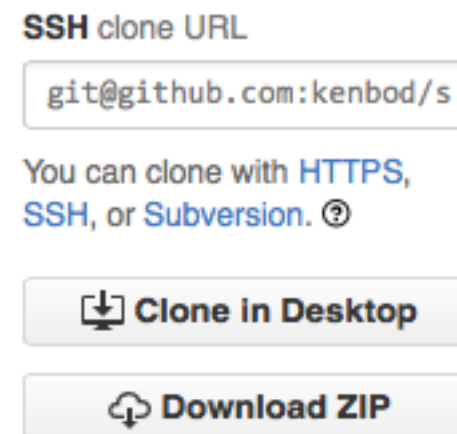
---

- 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

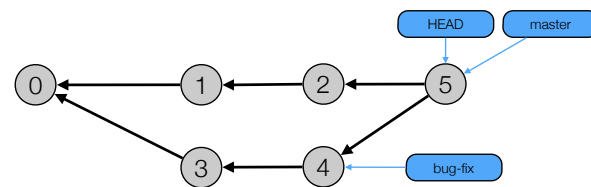
# Linking Repositories (I)

---

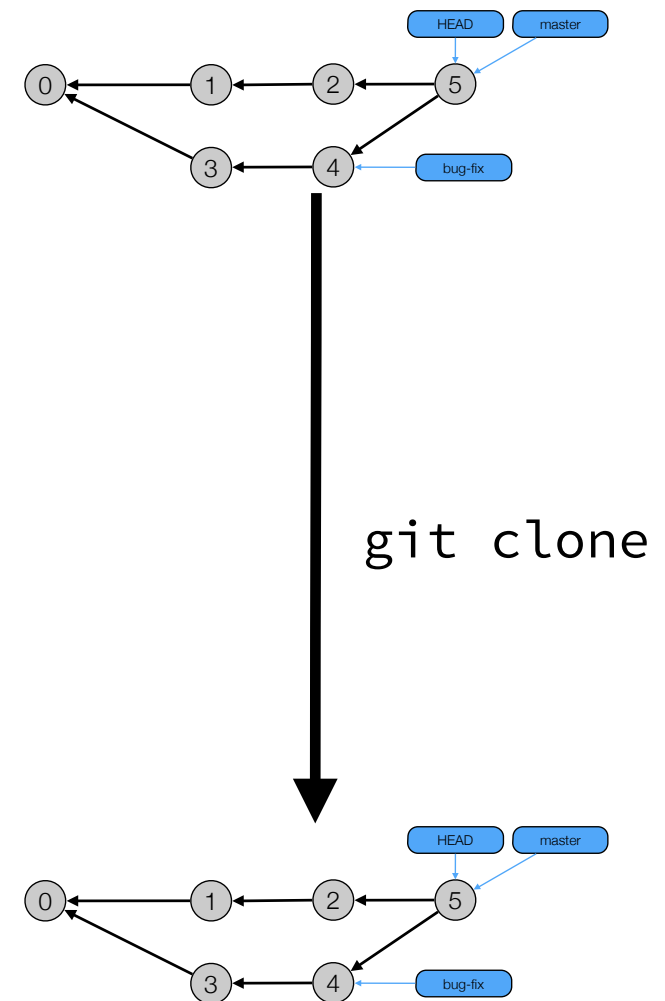
- The first thing that needs to happen to work with Github is linking a repository on your computer to one that exists on Github
- There are two primary ways to do this
  - **1.** If the repo exists on GitHub and not on your computer
    - `git clone <URI>`
    - GitHub make this easy by letting you 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**



# 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:kenbod/icse2016.git (fetch)`
      - `origin git@github.com:kenbod/icse2016.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)

Owner

Repository name



kenbod ▾

/



**Repo name goes here**

Great repository names are short and memorable. Need inspiration? How about **redolent-octo-rutabaga**.

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 let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▾

Add a license: **None** ▾




**Create repository**

# 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 “example\_project” to match what I called the repo locally.

**Quick setup — if you've done this kind of thing before**

 **Set up in Desktop** or **HTTPS** **SSH** `git@github.com:kenbod/example_project.git`

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

**...or create a new repository on the command line**

```
echo "# example_project" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin git@github.com:kenbod/example_project.git
git push -u origin master
```

**...or push an existing repository from the command line**

```
git remote add origin git@github.com:kenbod/example_project.git
git push -u origin master
```

**...or import code from another repository**

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

**Import code**

```
dlc75-223-dhcp:example_project $ ls
README.md
dlc75-223-dhcp:example_project $ git remote add origin git@github.com:kenbod/example_project.git
dlc75-223-dhcp:example_project $ git remote -v
origin  git@github.com:kenbod/example_project.git (fetch)
origin  git@github.com:kenbod/example_project.git (push)
dlc75-223-dhcp:example_project $ git push -u origin master
Counting objects: 21, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (14/14), done.
Writing objects: 100% (21/21), 1.60 KiB | 0 bytes/s, done.
Total 21 (delta 7), reused 0 (delta 0)
To git@github.com:kenbod/example_project.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
dlc75-223-dhcp:example_project $ git branch -a
  bug-fix
* master
remotes/origin/master
```

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
- As a result, 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 Presentations

---



# One Option: GitHub stores code and displays Markdown files for presentation

---

- If your presentation, involves a bunch of code
  - Say you are presenting on the topic of *Functional Programming in Scala*
- then, you can create a repo that has a **src** directory and put all of the source code that you need to demonstrate functional programming in the Scala programming language under that directory
  - You might have other top level directories that contain data, resources, and other information
- You can then create one or more Markdown documents in that repository to act as your “presentation”. Perhaps you have one long README.md document that contains the content of your presentation or you have a slides directory that contains markdown files titled 01.md, 02.md, etc. to indicate the order of your presentation.
- Create everything on your local computer and then upload to Github

# Second Option: Use GitHub's ability to serve a project website for the repo

---

- If your presentation has no code but lots of text and images, then you can create a repo that takes advantage of GitHub's ability to associate a website with a given repo
  - To take advantage of this functionality, follow these (high-level) steps:
    - create a new repo and associate it with GitHub
    - `git checkout --orphan gh-pages`
    - add HTML, CSS, and Javascript to your repo
    - `git push -u origin gh-pages`
  - If your repo is stored at `<https://github.com/<USER>/<REPO>>`
    - Your website is located at `<https://<USER>.github.io/<REPO>/>`
- I did this last year for one of my seminars:
  - Repo: `<https://github.com/cu-data-engineering-s15/lecture\_05>`
  - Website: `<https://cu-data-engineering-s15.github.io/lecture\_05/>`

# Third Option: Why Not Both?

---

- Have a topic that requires both text AND code
  - Then store code, data, resources, and instructions in the repo
  - And store HTML, CSS, and Javascript in the repo on the gh-pages branch
- At a high-level
  - create a new repo and associate it with GitHub
  - On the master branch, fill it with code and markdown files
  - `git checkout --orphan gh-pages`
  - `git rm -rf .; git commit`
    - This creates an empty working directory on the gh-pages branch
  - On the gh-pages branch, fill it with HTML, CSS, and Javascript
  - Switch between the two branches to work on code (master) and your website (gh-pages) until done;
  - `git push` frequently to view repo/website as it evolves

# Examples

---

- I have created two examples to view associated with my Github account
  - The first is this lecture presented with Markdown files on the master branch of this repository
    - [https://github.com/kenbod/markdown\\_github\\_01](https://github.com/kenbod/markdown_github_01)
  - The second is this lecture presented, again, as a website on the gh-pages branch of the same repository
    - [https://kenbod.github.io/markdown\\_github\\_01](https://kenbod.github.io/markdown_github_01)
- A “code heavy” example is left as an exercise for the reader
  - More details about THAT on Thursday

# Coming Up Next

---

- Lecture 4: Introduction to Software Engineering
- Homework 1 is due by the **start** of Lecture 4 (Thursday)
  - See class website for details