

# Git and GitHub Workshop

## Table of Contents

<b>Git Workshop .....</b>	<b>2</b>
<b>1. Installation and Setup .....</b>	<b>2</b>
1. Installation.....	2
2. Git Configuration .....	2
<b>2. Introduce the big picture before we move on.....</b>	<b>3</b>
<b>3. Create a Git Repo from scratch .....</b>	<b>3</b>
<b>4. Add commits to a repository .....</b>	<b>4</b>
Good practice for commit and commit messages .....	9
<b>5. Review a Repo's History .....</b>	<b>11</b>
<b>6. Tagging, Branching, and Merging .....</b>	<b>15</b>
git tag .....	15
git branch, git checkout, git merge .....	15
<b>7. Undoing changes .....</b>	<b>20</b>
 <b>GitHub Workshop .....</b>	 <b>22</b>
<b>1. Push your local project to GitHub .....</b>	<b>22</b>
<b>2. Add a README.....</b>	<b>24</b>
<b>3. push changes to GitHub.....</b>	<b>26</b>
<b>4. Clone existing project to local machine .....</b>	<b>27</b>
<b>5. Create your training portfolio .....</b>	<b>27</b>

In this workshop, we will work on a demo project together to try some commonly used git commands, such as git add, git commit, git branch, git checkout, git merge, etc. Then push it to your GitHub to share with others, and practice GitHub related commands like git push, git pull, git clone, etc.

## Git Workshop

Please refer to this file during the workshop for step-by-step commands.

### 1. Installation and Setup

#### 1. Installation

To download Git:

- 1) go to <https://git-scm.com/downloads>
- 2) download the software for Windows/Mac/Linux
- 3) install Git choosing all of the default options

Once everything is installed, you should be able to run `git` on the command line. If it displays the usage information, then you're good to go!

FYI, Git is actually installed on MacOS, but we'll be reinstalling it so that we'll have the newest version.

#### 2. Git Configuration

```
# sets up Git with your name
git config --global user.name "<Your-Full-Name>"

# sets up Git with your email
git config --global user.email "<your-email-address>"

# makes sure that Git output is colored
git config --global color.ui auto

# displays the original state in a conflict
git config --global merge.conflictstyle diff3

# displays your configurations
git config --list
```

```
(base) dongli ~ $ git config --global user.name "dongli"
(base) dongli ~ $ git config --global user.email "dongli.pzh@gmail.com"
(base) dongli ~ $ git config --global color.ui auto
(base) dongli ~ $ git config --global merge.conflictstyle diff3
(base) dongli ~ $ git config --list
credential.helper=osxkeychain
filter.lfs.required=true
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
user.name=dongli
user.email=dongli.pzh@gmail.com
core.editor=subl -n -w
push.default=upstream
merge.conflictstyle=diff3
color.ui=auto
```

FYI, git can be configured to use your preferred editor as well. Please feel free to google for more information if you are interested.

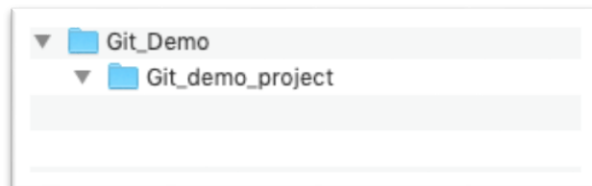
## 2. Introduce the big picture before we move on

( Explain the directory – staging index – repository structure in class )

## 3. Create a Git Repo from scratch

If you want to follow along with me, please do the following:

- 1) Create a directory/folder called ***Git\_Demo***
- 2) Inside that, create another directory called ***Git\_demo\_project***



- 3) Open your terminal/command window. Use the cd command to move into the ***Git\_demo\_project*** directory

```
(base) dongli ~ $ cd /Users/dongli/Desktop/Digi_Safari/Content/Git_Demo/Git_demo_project
(base) dongli Git_demo_project $
```

FYI, the previous three steps can be done with one command:

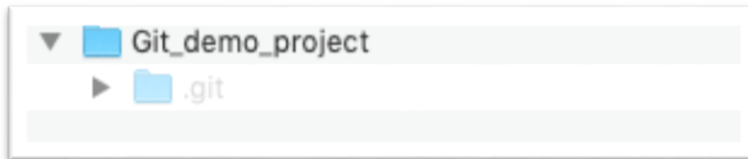
```
mkdir -p Git_Demo/Git_demo_project && cd $_
```

```
(base) dongli ~ $ cd /Users/dongli/Desktop/Digi_Safari/Content
(base) dongli Content $ mkdir -p Git_Demo/Git_demo_project && cd $_
(base) dongli Git_demo_project $
```

- 4) Now suppose we want to use version control on this project. All we need to do is run `git init`

```
(base) dongli ~ $ cd /Users/dongli/Desktop/Digi_Safari/Content
(base) dongli Content $ mkdir -p Git_Demo/Git_demo_project && cd $_
(base) dongli Git_demo_project $ git init
Initialized empty Git repository in /Users/dongli/Desktop/Digi_Safari/Content/Git_Demo/Git_demo_project/.git/
(base) dongli (master #) Git_demo_project $
```

“git init” command initializes an empty Git repository in the current directory. Now if you turn on the hidden files, you should be able to see a .git hidden file inside the Git\_demo\_project folder.



Please DON'T mess around this folder unless you know what you are doing... to be safe, just let it remain hidden.

- 5) Before we move to the next step, let me introduce the command you will be use a lot:

```
git status
```

```
(base) dongli ~ $ cd /Users/dongli/Desktop/Digi_Safari/Content
(base) dongli Content $ mkdir -p Git_Demo/Git_demo_project && cd $_
(base) dongli Git_demo_project $ git init
Initialized empty Git repository in /Users/dongli/Desktop/Digi_Safari/Content/Git_Demo/Git_demo_project/.git/
(base) dongli (master #) Git_demo_project $ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
(base) dongli (master #) Git_demo_project $
```

“git status” will tell us what Git is thinking and the state of our repository as Git sees it. Before you become fluent in Git, try to use it after any other command. This would help you understand how Git works and would help you from making (possibly) incorrect assumptions about the state of your files/repository.

## 4. Add commits to a repository

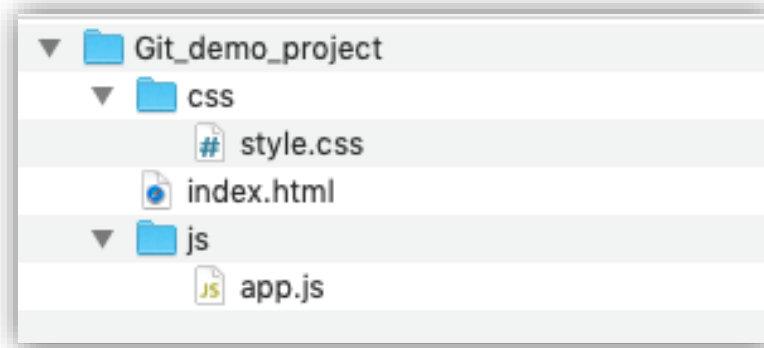
Now we already have an empty repository, we can start to work on our project content:

- 6) Go to your Git\_demo\_project folder, create an index.html file and copy the following starter code to index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <script src="js/app.js"></script>
  <meta charset="utf-8">
  <title>Git Workshop</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta name="description" content="">
  <link rel="stylesheet" href="css/app.css">
</head>
<body>

</body>
</html>
```

- 7) create a folder called “css”, and another folder called “js”
  - 8) Inside the “css” folder, create an empty css file called style.css
  - 9) Inside the “js” folder, create an empty js file called app.js
- Now your folder structure should look like:



- 10) before we move to the next step, use git status to check the status:

```

[(base) dongli ~ $ cd /Users/dongli/Desktop/Digi_Safari/Content
[(base) dongli Content $ mkdir -p Git_Demo/Git_demo_project && cd $_
[(base) dongli Git_demo_project $ git init
Initialized empty Git repository in /Users/dongli/Desktop/Digi_Safari/Content/Git_Demo/Git_demo_project/.git/
[(base) dongli (master #) Git_demo_project $ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
[(base) dongli (master #) Git_demo_project $ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        css/
        index.html
        js/

nothing added to commit but untracked files present (use "git add" to track)
(base) dongli (master #) Git_demo_project $ 

```

As you can see, even though we haven't done anything specific with Git, but it's watching this directory, and it knows we have created a couple of new folders/files! In addition, it also showing that all these changes are currently "untracked" by Git.

Seems like a good time to add a save point to our project. To do so, we first need to add these untracked files into the staging index using `git add`.

11) First let's try add one file at a time:

```

$ git add index.html
$ git status

```

```

(base) dongli (master #) Git_demo_project $ git add index.html
(base) dongli (master +) Git_demo_project $ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   index.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        css/
        js/

(base) dongli (master +) Git_demo_project $ 

```

12) Or we can add all the remaining files to staging index by:

```
$ git add .
```

```
[(base) dongli (master +) Git_demo_project $ git add .
[(base) dongli (master +) Git_demo_project $ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   css/style.css
        new file:   index.html
        new file:   js/app.js

(base) dongli (master +) Git_demo_project $
```

FYI, you can add multiple files at a time by listing their directories with a space in between. For example:

```
$ git add css/app.css js/app.js
```

13) After all the files are staged, we are ready to do our first commit.

```
$ git commit -m "Initial commit"
```

Note that, -m is followed by the message you provided for this commit. Commit message usually describes what is commit is about. For example, “add page footer”, “increase body font from 12 to 14”.

Now do a “git status”:

```
[(base) dongli (master +) Git_demo_project $ git commit -m "Initial commit"
[master (root-commit) 9c3cb35] Initial commit
 3 files changed, 17 insertions(+)
 create mode 100644 css/style.css
 create mode 100644 index.html
 create mode 100644 js/app.js
[(base) dongli (master) Git_demo_project $ git status
On branch master
nothing to commit, working tree clean
(base) dongli (master) Git_demo_project $
```

14) Now let's make a small change to the index.html: add a header inside the body tag.

```
<body>
  <header>
    <h1>Git Workshop</h1>
  </header>
</body>
```

After save the change, check the status:

```
[(base) dongli (master) Git_demo_project $ git status
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:   index.html

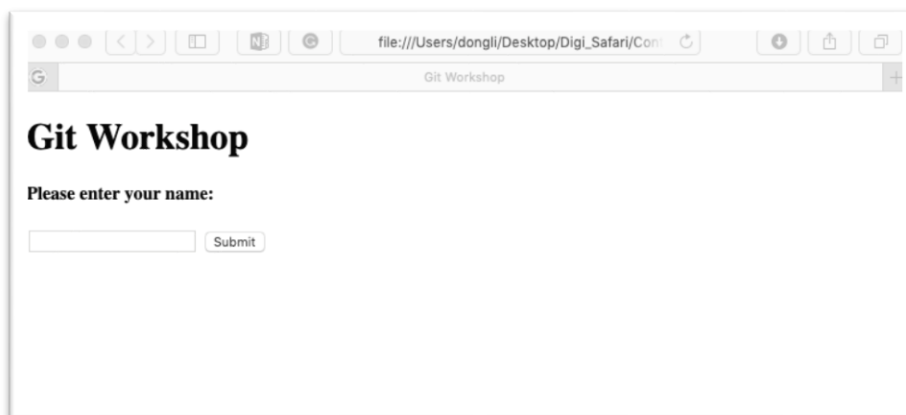
no changes added to commit (use "git add" and/or "git commit -a")
(base) dongli (master *) Git_demo_project $
```

15) Stage it and commit

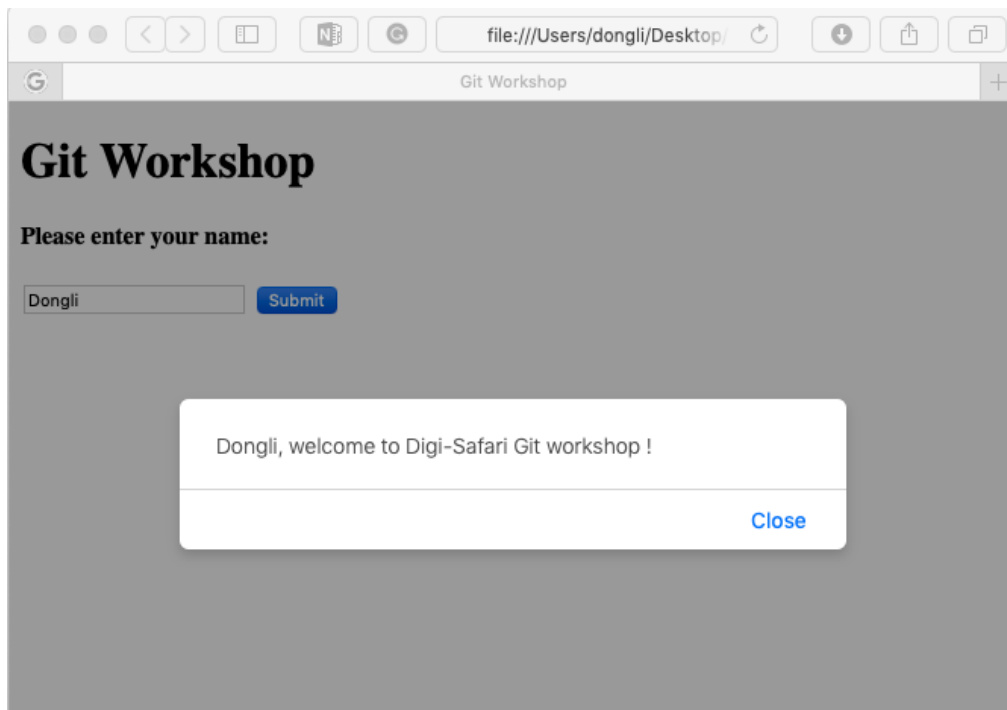
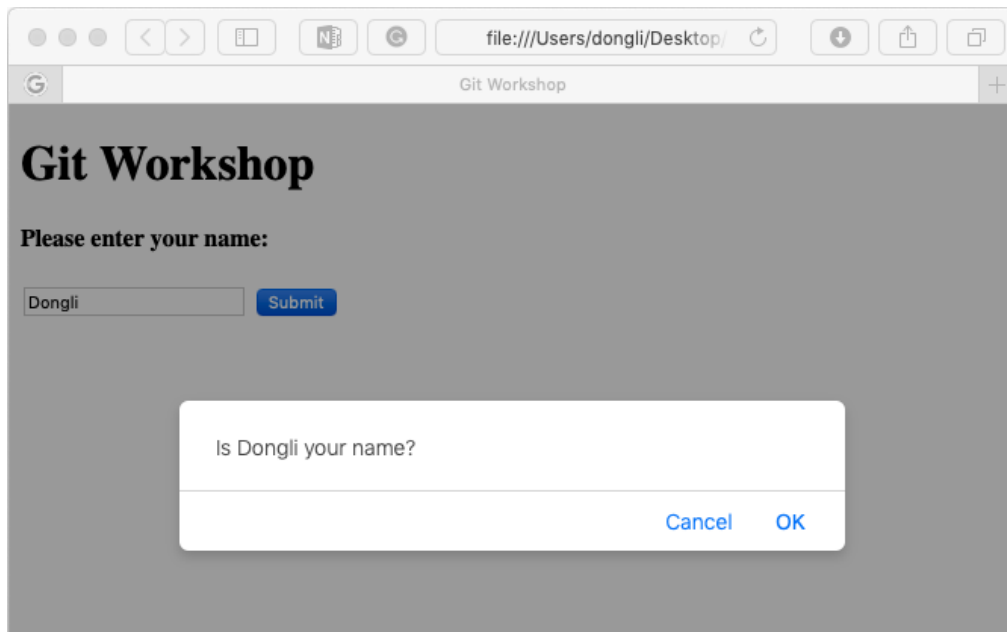
```
(base) dongli (master *) Git_demo_project $ git add index.html
[(base) dongli (master +) Git_demo_project $ git commit -m "add header to body"
[[master 3dcd916] add header to body
 1 file changed, 3 insertions(+)
(base) dongli (master) Git_demo_project $
```

Please do the following practice on you own:

- Edit your index.html and app.js so that you can have the users to input their name, and then say welcome to them. Just as what you did in your previous JavaScript assignment.
- Here are some sample screen shots:







- Now stage these changes and commit. Don't forget to add commit message.
- Check git status. You should still have working tree clean after this practice.

Good practice for commit and commit messages

**What to include in a commit?**

The goal is that each commit has a single focus.

Each commit should record a single-unit change. Now this can be a bit subjective (which is totally fine), but each commit should make a change to just one aspect of the project.

This isn't limiting the number of lines of code that are added/removed, or the number of files that are added/removed/modified. For example, to get user name and say hello to them, we edited the index.html and the app.js. A commit that records both of these changes would be totally fine.

Conversely, a commit shouldn't include unrelated changes – if we want to say welcome to the user in this commit, then we should randomly add a unrelated footer or something in the index.html.

### **What to include in a commit message?**

Do

- do keep the message short (less than 60-ish characters)
- do explain what the commit does (not how or why!)

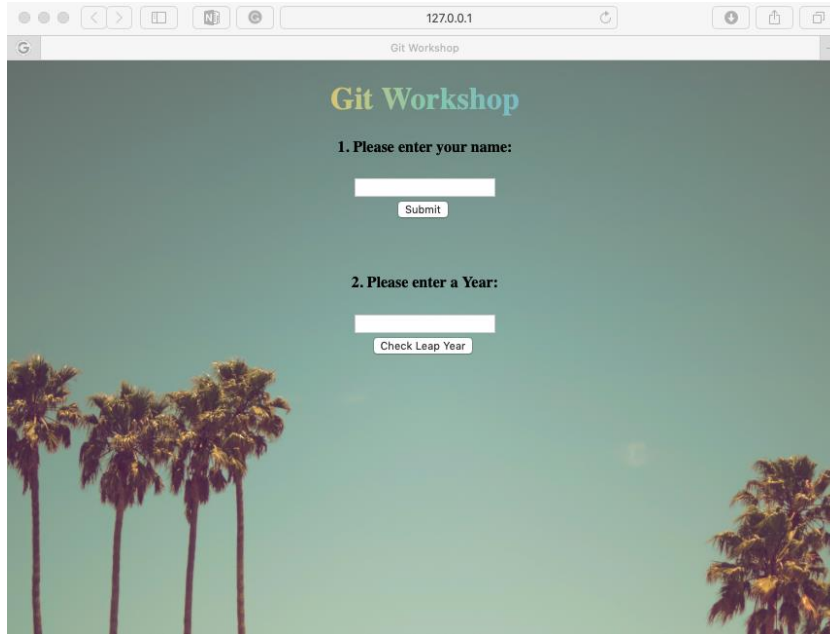
Do not

- do not explain why the changes are made (more on this below)
- do not explain how the changes are made (that's what git log -p is for!)
- be careful with the word "and"
  - if you have to use "and", your commit message is probably doing too many changes - break the changes into separate commits
  - e.g. "make the background color pink and increase the size of the sidebar"

Above all, be consistent in how you write your commit messages! If you have your git text editor setup, you can follow some more complicated rules on commit messages. Especially when you join a project team, it's always a good habit to see or ask if there is a define message rule for the team.

Practice makes perfect. Here is another short practice before we move on:

- Edit necessary files to add a leap year check to the page, which takes in user input year and print the check result on the page. Just like what you did in previous homework
- Also, add some style to the page, such as background color, font size, font color, ect.
- Keep in mind that “each commit should have a single focus”
- Here is a sample page screenshot:



## 5. Review a Repo's History

Take a look at this output from running `git log`:

```

((base) dongli (master) Git_demo_project $ git log
commit 2ec15056118ab5a1be0972bf34836d50bb22b68c (HEAD -> master)
Author: dongli <dongli.pzh@gmail.com>
Date: Sun Sep 8 19:43:08 2019 -0400

    add page style

commit 0e3b73fbcfd48af3e47a92ec842cbd106ba46d
Author: dongli <dongli.pzh@gmail.com>
Date: Sun Sep 8 19:41:01 2019 -0400

    add leap year check section

commit 8d4b836bf193a7ffcf5da9e88283346a426e11f
Author: dongli <dongli.pzh@gmail.com>
Date: Sun Sep 8 19:39:20 2019 -0400

    clean welcome message section

commit 7c5a76561137f43451423123be4934a980724247
Author: dongli <dongli.pzh@gmail.com>
Date: Sun Sep 8 18:33:01 2019 -0400

    add userInput name and welcome message

commit 3dcd916acecec65de6a629b56001b9d100188cd9
Author: dongli <dongli.pzh@gmail.com>
Date: Sun Sep 8 13:35:45 2019 -0400

    add header to body

commit c80253a9f3b493692a0044046ddfd7e321abd6a9
Author: dongli <dongli.pzh@gmail.com>
Date: Sun Sep 8 13:34:29 2019 -0400

    initial commit
((base) dongli (master) Git_demo_project $
```

Note that, for each commit, we can see its SHA (can be considered as commit id), author, date, and commit message. If you don't need all these detailed information, try `git log --oneline`:

```
[(base) dongli (master) Git_demo_project $ git log --oneline
2ec1505 (HEAD -> master) add page style
0e3b73f add leap year check section
8d4b836 clean welcome message section
7c5a765 add userInput name and welcome message
3dcd916 add header to body
c80253a initial commit
(base) dongli (master) Git_demo_project $ ]
```

On the other hand, if you need more information for each commit, such as which files are changed during that commit, use `git log --stat`:

```
[(base) dongli (master) Git_demo_project $ git log --stat
commit 2ec15056118ab5a1be0972bf34836d50bb22b68c (HEAD -> master)
Author: dongli <dongli.pzh@gmail.com>
Date: Sun Sep 8 19:43:08 2019 -0400

    add page style

css/style.css | 21 ++++++
index.html    | 2 +-
2 files changed, 22 insertions(+), 1 deletion(-)

commit 0e3b73fbcfddf48af3e47a92ec842cbd106ba46d
Author: dongli <dongli.pzh@gmail.com>
Date: Sun Sep 8 19:41:01 2019 -0400

    add leap year check section

index.html | 8 ++++++
js/app.js  | 25 ++++++
2 files changed, 33 insertions(+)

commit 8d4b836bf193a7fffcf5da9e88283346a426e11f
Author: dongli <dongli.pzh@gmail.com>
Date: Sun Sep 8 19:39:20 2019 -0400

    clean welcome message section

assets/sky_bg.jpg | Bin 0 -> 487586 bytes
index.html        | 13 ++++++
js/app.js         | 8 ++++++
3 files changed, 14 insertions(+), 7 deletions(-)

commit 7c5a76561137f43451423123be4934a980724247
Author: dongli <dongli.pzh@gmail.com>
Date: Sun Sep 8 18:33:01 2019 -0400

    add userInput name and welcome message

index.html | 8 ++++++
js/app.js  | 10 ++++++
2 files changed, 16 insertions(+), 2 deletions(-)
```

If you need to know exactly changes you made in each commit, use `git log -p`:

```
(base) dongli (master) Git_demo_project $ git log -p
[commit 2ec15056118ab5a1be0972bf34836d50bb22b68c (HEAD -> master)]
Author: dongli <dongli.pzh@gmail.com>
Date: Sun Sep 8 19:43:08 2019 -0400

    add page style

diff --git a/css/style.css b/css/style.css
index e69de29..078617f 100644
--- a/css/style.css
+++ b/css/style.css
@@ -0,0 +1,21 @@
+html {
+  background: url(../assets/sky_bg.jpg) no-repeat center center fixed;
+  -webkit-background-size: cover;
+  -moz-background-size: cover;
+  -o-background-size: cover;
+  background-size: cover;
+}
+
+.rainbow-text {
+  background-image: linear-gradient(to left, violet, #b0aac0, #87bdd8, #96ceb4, #ffcc5c, orange, #d96459);
+  -webkit-background-clip: text;
+  -webkit-text-fill-color: transparent;
+}
+
+header {
+text-align: center;
+}
+
+section {
+text-align: center;
+}
\ No newline at end of file
diff --git a/index.html b/index.html
index ae169c4..9670adf 100644
--- a/index.html
+++ b/index.html
@@ -10,7 +10,7 @@
</head>
<body>
  <header>
-    <h1>Git Workshop</h1>
+    <h1 class="rainbow-text">Git Workshop</h1>
  </header>
  <section>
    <label for="year"><h4>1. Please enter your name: </h4></label>
  </section>
</body>
</html>
[commit 0e3b73fbcfddf48af3e47a92ec842cbd106ba46d]
```

All the previous git log commands can be combined with the commit SHA to display commit information up to a specific commit. Eg “`git log -p <SHA>`”

Or, if you only need the details for one commit, you can use “`git show <SHA>`” For example, if I want to check the detailed changes made during the “add leap year check section” commit. All I need to run is “`git show 0e3b7`”:

```

(base) dongli (master) Git_demo_project $ git show 0e3b7
commit 0e3b73fbcfddf48af3e47a92ec842cbd106ba46d
Author: dongli <dongli.pzh@gmail.com>
Date: Sun Sep 8 19:41:01 2019 -0400

    add leap year check section

diff --git a/index.html b/index.html
index 4341b7a..ae169c4 100644
--- a/index.html
+++ b/index.html
@@ -19,5 +19,13 @@
     <br>
     <p id="welcome_msg"></p>
   </section>
+  <br>
+  <section>
+    <label for="year"><h4>2. Please enter a Year: </h4></label>
+    <input type="text" id="year"><br>
+    <button onclick="checkLeapYear()">Check Leap Year</button>
+    <br>
+    <p id="result"></p>
+  </section>
</body>
</html>
\ No newline at end of file
diff --git a/js/app.js b/js/app.js
index 05dc8f2..1338beb 100644
--- a/js/app.js
+++ b/js/app.js
@@ -11,4 +11,29 @@ function nameCheck(){
   alert("Please enter full name.");
   document.getElementById("welcome_msg").innerHTML = "Welcome!";
}
+
+function checkLeapYear() {
+  //clear for resubmits
+  document.getElementById("result").innerHTML = "";
+
+  var year = document.getElementById("year").value;
+  var answer = "Not a leap year"
+
+  if (year % 4 == 0) {
+    answer = "It's a leap year!";
+  }
+  if (year % 100 == 0) {
+    answer = "Not a leap year";
+  }
+  if (year % 100 == 0 && year % 400 == 0) {
+    answer = "It's a leap year!";
+  }
+
+  document.getElementById("result").innerHTML = answer;
+
+  // If the year can be evenly divided by 100, it is NOT a leap year, unless;
+  // The year is also evenly divisible by 400. Then it is a leap year.
+}
\ No newline at end of file
(base) dongli (master) Git_demo_project $

```

Before we finish up this section, let me introduce one more command: `git diff`. `git diff` can be used to check the changes you made even before you stage/commit them. Try change something in your index.html, and run `git diff`:

```

(base) dongli (master) Git_demo_project $ git diff
diff --git a/index.html b/index.html
index 9670adf..a6a7552 100644
--- a/index.html
+++ b/index.html
@@ -10,7 +10,7 @@
</head>
<body>
  <header>
-    <h1 class="rainbow-text">Git Workshop</h1>
+    <h1 class="rainbow-text">* Git Workshop *</h1>
  </header>
  <section>
    <label for="year"><h4>1. Please enter your name: </h4></label>

```

## 6. Tagging, Branching, and Merging

Overview:

- `git tag` : add tags to specific commits
- `git branch` : allows multiple lines of development in parallel
- `git checkout` : switch between different branches and tags
- `git merge` : combines changes on different branches automatically

### git tag

After a couple of commits, your app might already become mature enough to call itself version 1.0. That's when we tag that commit with a "v1.0" tag. Let's try it with our Git workshop web application: (note `git log --decorate` is now equivalent to `git log`)

```
((base) dongli (master) Git_demo_project $ git tag -a v1.0 -m "Ready for the first release"
((base) dongli (master) Git_demo_project $ git log --decorate
commit 8edfea83fc337ece24ebc3b6d871d181a5b3b251 (HEAD -> master, tag: v1.0)
Author: dongli <dongli.pzh@gmail.com>
Date: Sun Sep 8 20:25:10 2019 -0400

    add stars to page title

commit 2ec15056118ab5a1be0972bf34836d50bb22b68c
Author: dongli <dongli.pzh@gmail.com>
Date: Sun Sep 8 19:43:08 2019 -0400

    add page style

commit 0e3b73fbcfd48af3e47a92ec842cbd106ba46d
Author: dongli <dongli.pzh@gmail.com>
Date: Sun Sep 8 19:41:01 2019 -0400

    add leap year check section

commit 8d4b836bf193a7fffcf5da9e88283346a426e11f
Author: dongli <dongli.pzh@gmail.com>
Date: Sun Sep 8 19:39:20 2019 -0400

    clean welcome message section

commit 7c5a76561137f43451423123be4934a980724247
Author: dongli <dongli.pzh@gmail.com>
Date: Sun Sep 8 18:33:01 2019 -0400

    add userInput name and welcome message

commit 3dcd916acecec65de6a629b56001b9d100188cd9
Author: dongli <dongli.pzh@gmail.com>
Date: Sun Sep 8 13:35:45 2019 -0400

    add header to body

commit c80253a9f3b493692a0044046ddfd7e321abd6a9
Author: dongli <dongli.pzh@gmail.com>
Date: Sun Sep 8 13:34:29 2019 -0400

    initial commit
(base) dongli (master) Git_demo_project $
```

If you want to delete the tag, do `git tag -d v1.0`

### git branch, git checkout, git merge

( Explain git branch concept in class. Introducing the HEAD.)

`git branch` can be used to :

- list all branch names in the repository

- create new branches
- delete branches

If we do git branch now, all we can see is our only branch, the master branch:

```
[(base) dongli (master) Git_demo_project $ git branch
* master
```

Now, let's create a new branch called sidebar: git branch sidebar

```
[(base) dongli (master) Git_demo_project $ git branch sidebar
[(base) dongli (master) Git_demo_project $ git branch
* master
  sidebar
[(base) dongli (master) Git_demo_project $ git status
On branch master
nothing to commit, working tree clean
(base) dongli (master) Git_demo_project $
```

As you can see, a new branch sidebar was created, but our HEAD, denoted with “\*” is still on master branch. So if you do git status, it still say “on branch master”.

To switch to the new sidebar branch, we need to checkout it `git checkout sidebar`:

```
[(base) dongli (master) Git_demo_project $ git checkout sidebar
Switched to branch 'sidebar'
[(base) dongli (sidebar) Git_demo_project $ git branch
  master
* sidebar
[(base) dongli (sidebar) Git_demo_project $ git status
On branch sidebar
nothing to commit, working tree clean
(base) dongli (sidebar) Git_demo_project $
```

Note: git checkout will:

- **remove** all files and directories from the Working Directory that Git is tracking (files that Git tracks are stored in the repository, so nothing is lost)
- go into the repository and pull out all of the files and directories of the commit that the branch points to

Now let's make some commits on the branches.

16) First make sure you are on the sidebar branch. Then add the following code to your index.html body tag:

```
<body>
<!-- Side navigation -->
<div class="sidenav">
  <a href="#">About</a>
  <a href="#">Services</a>
  <a href="#">Contact</a>
```



```

</div>

<!-- Page content -->
<div class="main">
  <header>
    <h1 class="rainbow-text">* Git Workshop *</h1>
  </header>

  <section>
    <label for="year"><h4>1. Please enter your name: </h4></label>
    <input type="text" id="Name"><br>
    <button onclick="nameCheck()">Submit</button>
    <br>
    <p id="welcome_msg"></p>
  </section>
  <br>
  <section>
    <label for="year"><h4>2. Please enter a Year: </h4></label>
    <input type="text" id="year"><br>
    <button onclick="checkLeapYear()">Check Leap Year</button>
    <br>
    <p id="result"></p>
  </section>
</div>
</body>

```

17) Add to stage index and commit:

```

[(base) dongli (sidebar *) Git_demo_project $ git add .
[(base) dongli (sidebar +) Git_demo_project $ git status
On branch sidebar
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   index.html

[(base) dongli (sidebar +) Git_demo_project $ git commit -m "add sidebar html"
[sidebar 16f54a6] add sidebar html
1 file changed, 42 insertions(+), 31 deletions(-)
rewrite index.html (63%)
(base) dongli (sidebar) Git_demo_project $ 

```

18) Let's add sidebar style by creating a sidebar\_style.css inside the css folder. Copy the following code to sidebar\_style.css:

```

/* The sidebar menu */
.sidenav {
  height: 100%; /* Full-height: remove this if you want "auto" height */
  width: 160px; /* Set the width of the sidebar */
  position: fixed; /* Fixed Sidebar (stay in place on scroll) */
  z-index: 1; /* Stay on top */
  top: 0; /* Stay at the top */

```

```

left: 0;
background-color: rgba(200,200,200,0.7); Black
overflow-x: hidden; /* Disable horizontal scroll */
padding-top: 20px;
}

/* The navigation menu links */
.sidenav a {
padding: 6px 8px 6px 16px;
text-decoration: none;
font-size: 25px;
display: block;
}

/* When you mouse over the navigation links, change their color */
.sidenav a:hover {
color: #f1f1f1;
}

/* Style page content */
.main {
margin-left: 160px; /* Same as the width of the sidebar */
padding: 0px 10px;
}

/* On smaller screens, where height is less than 450px, change the style of the sidebar (less padding and a smaller font size) */
@media screen and (max-height: 450px) {
.sidenav {padding-top: 15px;}
.sidenav a {font-size: 18px;}
}

```

Git status:

```

(base) dongli (sidebar *) Git_demo_project $ git status
On branch sidebar
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:   index.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        css/sidebar_style.css

no changes added to commit (use "git add" and/or "git commit -a")
(base) dongli (sidebar *) Git_demo_project $ 

```

Stage and commit the changes:

- 19) Let's move back to master branch by git checkout master
- 20) Make some changes to the style.css file. For me, I changed the background image to some new image picture, and adjusted the title color accordingly:
- 21) Stage and commit the changes
- 22) Run `git log --oneline --decorate --graph -all`

```
(base) dongli (master) Git_demo_project $ git log --oneline --decorate --graph --all
* e7c8188 (HEAD -> master) restyle page
| * e12e3fa (sidebar) add sidebar style
| * 16f54a6 add sidebar html
|/
* 8edfea8 (tag: v1.0) add stars to page title
* 2ec1505 add page style
* 0e3b73f add leap year check section
* 8d4b836 clean welcome message section
* 7c5a765 add userinput name and welcome message
* 3dcd916 add header to body
* c80253a initial commit
(base) dongli (master) Git_demo_project $
```

We can clearly see that after v1.0, we have two parallel branches going on. One is the sidebar with two commits, one is the master with one commit.

- 23) Now let's merge the sidebar to the newly styled main page. Before you merge, make sure you are on the master branch, then do:

```
git merge sidebar -m "merge sidebar to newly styled main page"
```

- 24) after merging, git log to visualize it:

```
(base) dongli (master) Git_demo_project $ git merge sidebar -m "merge sidebar to newly styled main page"
Merge made by the 'recursive' strategy.
 css/sidebar_style.css | 38 ++++++++++++++++++++++++++++++++++++++
 index.html             | 46 ++++++++++++++++++++++++++++++++++++++
 2 files changed, 67 insertions(+), 17 deletions(-)
 create mode 100644 css/sidebar_style.css
(base) dongli (master) Git_demo_project $ git log --oneline --decorate --graph --all
* b3c12fb (HEAD -> master) merge sidebar to newly styled main page
| \
| * e12e3fa (sidebar) add sidebar style
| * 16f54a6 add sidebar html
| * | e7c8188 restyle page
|/
* 8edfea8 (tag: v1.0) add stars to page title
* 2ec1505 add page style
* 0e3b73f add leap year check section
* 8d4b836 clean welcome message section
* 7c5a765 add userinput name and welcome message
* 3dcd916 add header to body
* c80253a initial commit
(base) dongli (master) Git_demo_project $
```

Attention: sometimes merges fail, it's called merge conflict. Merge conflict will occur when the exact same line(s) are changed in separate branches. So it's important to have branches works independently. This is why the modern frameworks are so into loosely coupling design.

Beside jumping between branches, git checkout can also be used to detach the HEAD and go back to a specific commit. Please try it and read the output note.

```
[(base) dongli (master) Git_demo_project $ git checkout 8d4b836
Note: checking out '8d4b836'.
```

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using `-b` with the checkout command again. Example:

```
git checkout -b <new-branch-name>
```

HEAD is now at 8d4b836 clean welcome message section

```
[(base) dongli ((8d4b836...)) Git_demo_project $ git checkout -
Previous HEAD position was 8d4b836 clean welcome message section
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
(base) dongli (master) Git_demo_project $
```

## 7. Undoing changes

- `git commit --amend`
  - if you saw a typo in your commit message right after you made a commit. Just do `git commit -amend -m "new message"`
  - For example:

```
[(base) dongli (master +) Git_demo_project $ git commit -m "give p a rainball"
[master 172f881] give p a rainball
1 file changed, 2 insertions(+), 2 deletions(-)
[(base) dongli (master) Git_demo_project $ git commit --amend -m "give p a rainbow"
[master e4acad3] give p a rainbow
Date: Sun Sep 8 22:10:33 2019 -0400
1 file changed, 2 insertions(+), 2 deletions(-)
[(base) dongli (master) Git_demo_project $ git log --oneline --decorate --graph --all
* e4acad3 (HEAD -> master) give p a rainbow
* b3c12fb merge sidebar to newly styled main page
| \
| * e12e3fa (sidebar) add sidebar style
| * 16f54a6 add sidebar html
* | e7c8188 restyle page
|/
* 8edfea8 (tag: v1.0) add stars to page title
* 2ec1505 add page style
* 0e3b73f add leap year check section
* 8d4b836 clean welcome message section
* 7c5a765 add userinput name and welcome message
* 3dcd916 add header to body
* c80253a initial commit
(base) dongli (master) Git_demo_project $
```

- Also, if you forget to add/update some files for the last commit, do:
  - Edit the file(s)
  - Save the file(s)
  - Stage the file(s)
  - And run `git commit --amend`

- git revert
- git reset

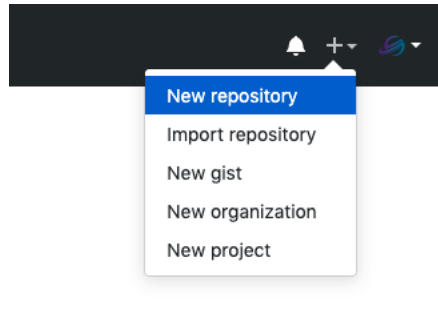
Git revert and git reset Will add information about git revert and git reset after the workshop for your reference.

# GitHub Workshop

## 1. Push your local project to GitHub

Now let's put the demo project to our GitHub account!

- 1) Log in to your GitHub account, click on the “+” icon in the upper right corner and choose “New Repository”:



- 2) Give a name to the “Repository name” and “Description”. Leave everything else as default, then click on “Create repository”

### Create a new repository

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

#### Repository template

Start your repository with a template repository's contents.

No template ▾

#### Owner

 digi-safari ▾

#### Repository name \*

Git-GitHub-Lab ✓

Great repository names are short and memorable. Need inspiration? How about [animated-waffle](#)?

#### Description (optional)

Git and GitHub workshop preparation.

☒ **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**

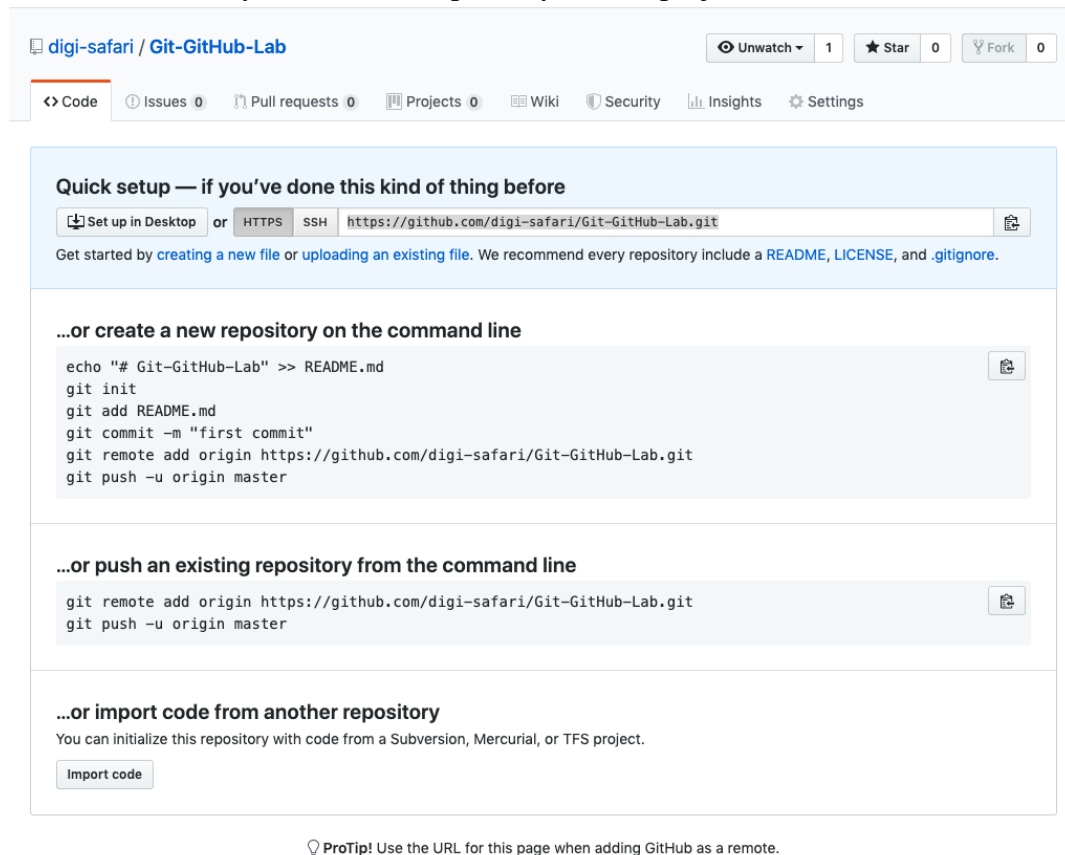
This will let you immediately clone the repository to your computer.

Add .gitignore: **None** ▾

Add a license: **None** ▾ ⓘ

Create repository

- 3) On the new page, you will have a Quick setup guide. We are going to follow the instructions on this page for “push an existing repository from the command line” because we already have a local repository for our project:



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

☐ Set up in Desktop or ☐ HTTPS ☐ SSH

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

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

```
echo "# Git-GitHub-Lab" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/digi-safari/Git-GitHub-Lab.git
git push -u origin master
```

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

```
git remote add origin https://github.com/digi-safari/Git-GitHub-Lab.git
git push -u origin master
```

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

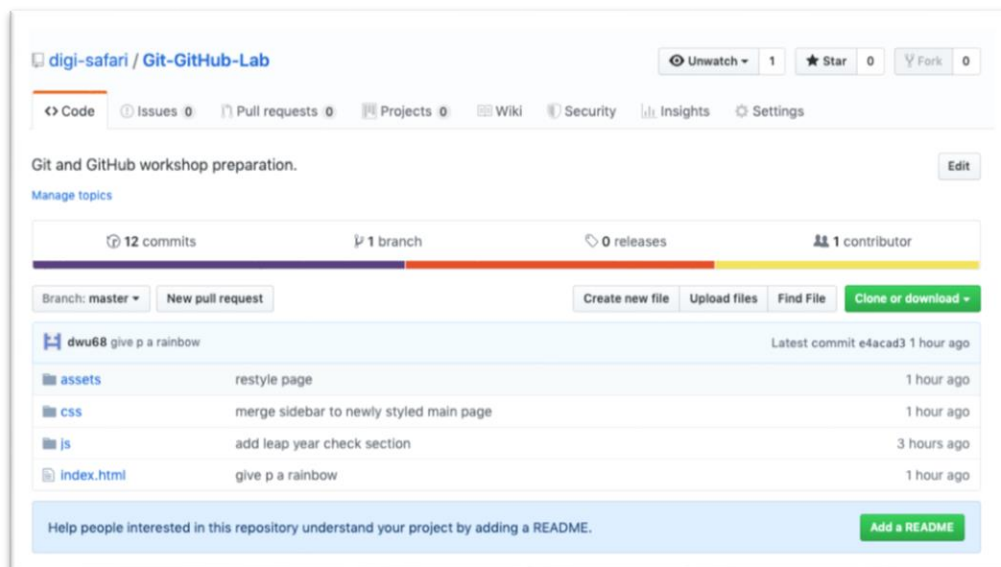
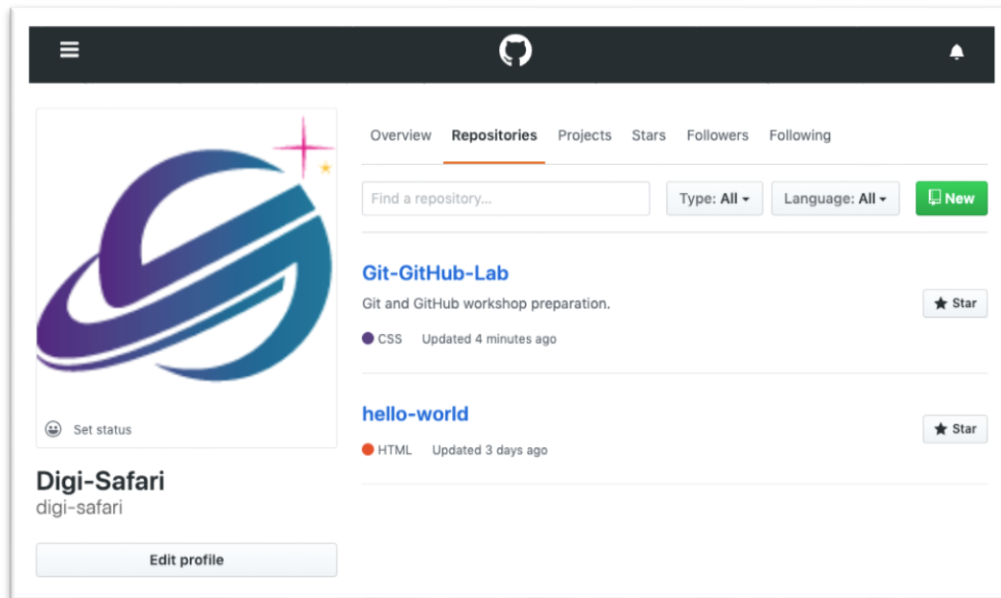
You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

ProTip! Use the URL for this page when adding GitHub as a remote.

- 4) In the command window, do a git status to make sure you have a clean tree. Then paste and run the commands:

```
(base) ~dongli (master) Git_demo_project $ git remote add origin https://github.com/digi-safari/Git-GitHub-Lab.git
(base) ~dongli (master) Git_demo_project $ git push -u origin master
Username for 'https://github.com': digi-safari
Password for 'https://digi-safari@github.com':
Enumerating objects: 54, done.
Counting objects: 100% (54/54), done.
Delta compression using up to 8 threads
Compressing objects: 100% (45/45), done.
Writing objects: 100% (54/54), 1.37 MiB | 1.24 MiB/s, done.
Total 54 (delta 13), reused 0 (delta 0)
remote: Resolving deltas: 100% (13/13), done.
To https://github.com/digi-safari/Git-GitHub-Lab.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
(base) ~dongli (master) Git_demo_project $
```

- 5) Now your local repo is successfully pushed to GitHub!
- 6) Later, if you need to update the repo, just do git push <REMOTENAME> <



## 2. Add a README

We recommend you to include a README.md for every project to introduce what you did to anyone who might be interested. “.md” stands for markdown. If you have ever worked with Jupyter notebook, you probably already know how to create a nice markdown to present a project. Even if this is the first time you ever heard of it, it’s okay, here is a helpful cheat sheet to bring you up to speed: <https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet>

Note that you can use raw html in markdown file!

For example:

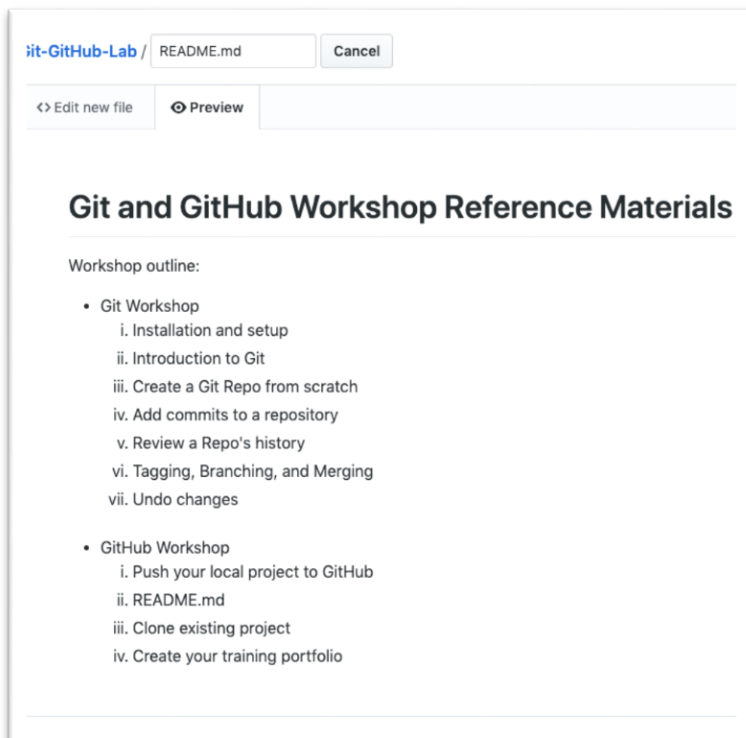


Git-GitHub-Lab / README.md Cancel

<> Edit new file

Preview

```
1 # Git and GitHub Workshop Reference Materials
2
3 Workshop outline:
4 <ul>
5   <li>Git Workshop
6     <ol>
7       <li>Installation and setup</li>
8       <li>Introduction to Git</li>
9       <li>Create a Git Repo from scratch</li>
10      <li>Add commits to a repository</li>
11      <li>Review a Repo's history</li>
12      <li>Tagging, Branching, and Merging</li>
13      <li>Undo changes</li>
14    </ol>
15  </li><br>
16  <li>GitHub Workshop
17    <ol>
18      <li>Push your local project to GitHub</li>
19      <li>README.md</li>
20      <li>Clone existing project</li>
21      <li>Create your training portfolio</li>
22    </ol>
23  </li>
24 </ul>
25
```



Let's commit this readme.md change directly to the master branch.

### 3. push changes to GitHub

Now suppose we created a new branch and made some changes on our local machine:

```
(base) dongli (master) Git_demo_project $ git branch readme
(base) dongli (master) Git_demo_project $ git branch
* master
  readme
  sidebar
(base) dongli (master) Git_demo_project $ git checkout readme
Switched to branch 'readme'
(base) dongli (readme) Git_demo_project $ git status
On branch readme
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    markdown_imgs/

nothing added to commit but untracked files present (use "git add" to track)
(base) dongli (readme) Git_demo_project $ git add .
(base) dongli (readme +) Git_demo_project $ git commit -m "add logo images to markdown_imgs folder"
[readme 0a7b9f3] add logo images to markdown_imgs folder
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100755 markdown_imgs/logo192.png
(base) dongli (readme) Git_demo_project $
```

To push this change to GitHub repo, you can either push the readme branch then merge in GitHub, or merge readme to master then push. Let's first do merge then push:

```
(base) dongli (readme) Git_demo_project $ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
(base) dongli (master) Git_demo_project $ git merge readme
Updating e4acad3..0a7b9f3
Fast-forward
 markdown_imgs/logo192.png | Bin 0 -> 13039 bytes
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100755 markdown_imgs/logo192.png
(base) dongli (master) Git_demo_project $ git push origin master
To https://github.com/digi-safari/Git-GitHub-Lab.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/digi-safari/Git-GitHub-Lab.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
(base) dongli (master) Git_demo_project $
```

Oops, the push request is rejected. Can you think of why? How to fix it?

## 4. Clone existing project to local machine

To clone other people's project to your local machine, we can create an empty folder which will hold the project first, then use `git clone`. For practice purpose, please clone the online retail maintenance project from Digi-Safari: <https://github.com/digi-safari/MaintenanceOnlineRetail>

- Use `mkdir` and `cd` to create enter the new folder
- `git clone https://github.com/digi-safari/MaintenanceOnlineRetail.git`
  - note, this link can be found on the repo page
- after the git clone is done, the whole repository is setup in the folder you created

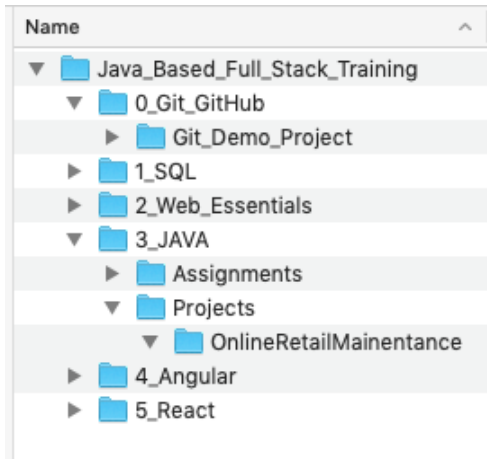
```
[(base) dongli ~ $ cd /Users/dongli/Desktop/Digi_Safari/Content/Git_GitHub_Demo/student-account
[(base) dongli student-account $ mkdir clone_demo
[(base) dongli student-account $ ls
Java-Based-Full-Stack-Training  clone_demo
[(base) dongli student-account $ cd clone_demo
[(base) dongli clone_demo $ git clone https://github.com/digi-safari/MaintenanceOnlineRetail.git
Cloning into 'MaintenanceOnlineRetail'...
remote: Enumerating objects: 258, done.
remote: Counting objects: 100% (258/258), done.
remote: Compressing objects: 100% (166/166), done.
remote: Total 258 (delta 68), reused 258 (delta 68), pack-reused 0
Receiving objects: 100% (258/258), 2.97 MiB | 586.00 KiB/s, done.
Resolving deltas: 100% (68/68), done.
[(base) dongli clone_demo $ ls
MaintenanceOnlineRetail
[(base) dongli clone_demo $ cd MaintenanceOnlineRetail/
[(base) dongli (master) MaintenanceOnlineRetail $ ls
Modules                                Required Jar files
OnlineRetail_DBScripts.txt            markdown_imgs
README.md
[(base) dongli (master) MaintenanceOnlineRetail $ ]
```

## 5. Create your training portfolio

We've found that candidates who can present their training projects on GitHub tends to have a higher chance to get hired. So we request you to start organizing your work and keep them up to date on your GitHub.

Let me help you to get started:

1. Create a folder on your laptop to organize your training work. It may include notes, assignments, and/or projects. You make the decisions.  
Here is a sample folder I will be using for later steps:



2. In this example, I will put the files related to online retail maintenance project to GitHub. Before we move on, let's make this folder a repo:

```

((base) dongli Java_Based_Full_Stack_Training $ git init
Initialized empty Git repository in /Users/dongli/Desktop/Digi_Safari/Content/Git_GitHub_Demo/Java_Based_Full_Stack_Training/.git/
((base) dongli (master #) Java_Based_Full_Stack_Training $ git add .
((base) dongli (master +) Java_Based_Full_Stack_Training $ git status
On branch master

No commits yet

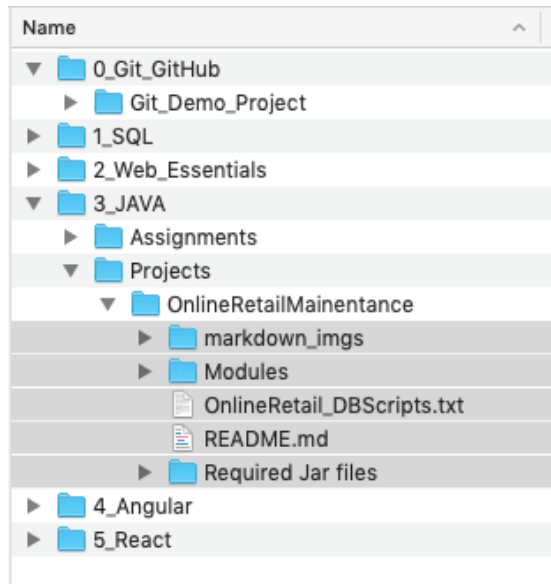
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   0_Git_GitHub/Git_Demo_Project/README.md
        new file:   0_Git_GitHub/Git_Demo_Project/assets/sky_bg.jpg
        new file:   0_Git_GitHub/Git_Demo_Project/assets/sky_bg_2.jpg
        new file:   0_Git_GitHub/Git_Demo_Project/css/sidebar_style.css
        new file:   0_Git_GitHub/Git_Demo_Project/css/style.css
        new file:   0_Git_GitHub/Git_Demo_Project/index.html
        new file:   0_Git_GitHub/Git_Demo_Project/js/app.js
        new file:   0_Git_GitHub/Git_Demo_Project/markdown_imgs/logo192.png

((base) dongli (master +) Java_Based_Full_Stack_Training $ git commit -m "initial commit"
[master (root-commit) 0d5ccd9] initial commit
8 files changed, 167 insertions(+)
create mode 100644 0_Git_GitHub/Git_Demo_Project/README.md
create mode 100644 0_Git_GitHub/Git_Demo_Project/assets/sky_bg.jpg
create mode 100644 0_Git_GitHub/Git_Demo_Project/assets/sky_bg_2.jpg
create mode 100644 0_Git_GitHub/Git_Demo_Project/css/sidebar_style.css
create mode 100644 0_Git_GitHub/Git_Demo_Project/css/style.css
create mode 100644 0_Git_GitHub/Git_Demo_Project/index.html
create mode 100644 0_Git_GitHub/Git_Demo_Project/js/app.js
create mode 100755 0_Git_GitHub/Git_Demo_Project/markdown_imgs/logo192.png
(base) dongli (master) Java_Based_Full_Stack_Training $

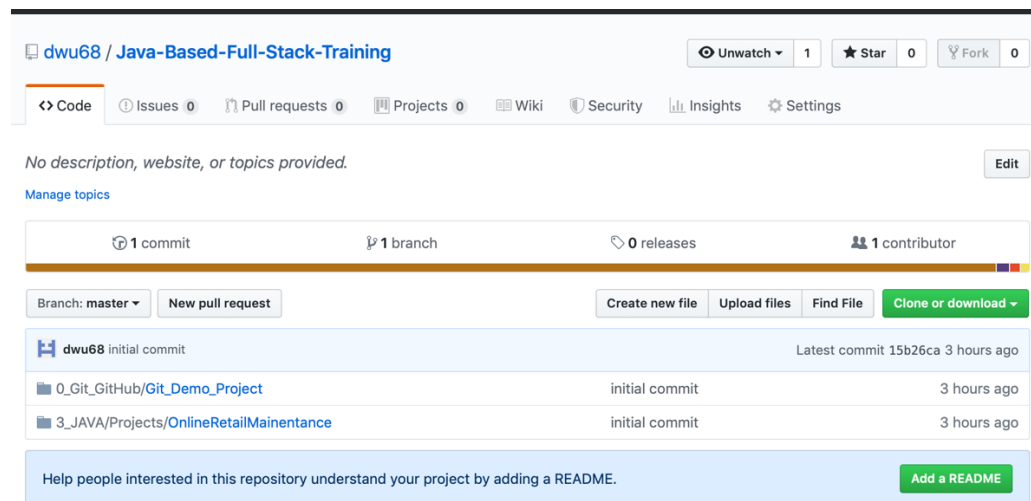
```

3. Now in addition to submit Online Retail Maintenance project to LMS, we would also request you to push this project on your GitHub page. To do so, we will need to:
  - a. First download/clone the repo from this link: <https://github.com/digi-safari/MaintenanceOnlineRetail> ( you probably already have it done in previous step )
  - b. Copy the files to your OnlineRetailMaintenance folder. Be careful to copy only the files not the entire git repo because git inside a git is not a recommended practice.



- c. Stage and commit.
- d. Also put your solution files in the folder. Stage and commit.
- e. Lastly, push this repo to your GitHub as what you did in GitHub workshop part 1.

If everything goes well, you will have this training repository created on your GitHub:



As you might have noticed, git ignored all the empty folders we have created in the local working directory. It's ok. Once you put your assignment/project files in, they will be pushed to correctly through the "git add, git commit, git push" workflow.

#### NOTE:

If you enter permission denied error during the last step, follow the steps listed in this link to fix it: <https://gist.github.com/adamjohnson/5682757>