

# Git and GitHub Workshop

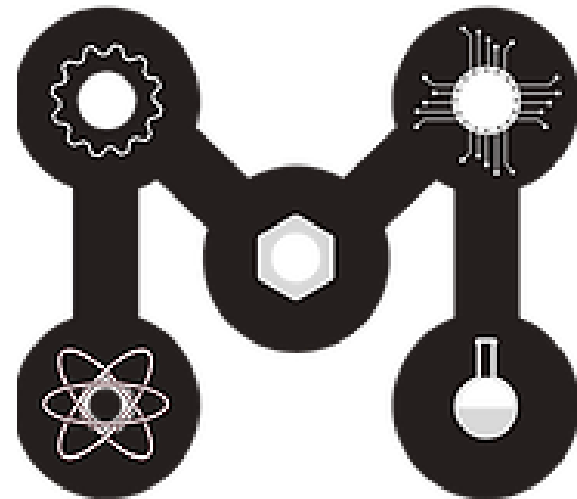
Be not afraid of Git

Hosted by:



**IEEE**

**UW Madison**




# More Workshops This Year!



<https://making.engr.wisc.edu>

← → ↻ ⓘ https://making.engr.wisc.edu

UNIVERSITY of WISCONSIN-MADISON

 **GRAINGER ENGINEERING DESIGN INNOVATION LAB**

HOME GET INVOLVED ^ EQUIPMENT POLICIES v ABOUT US v QUESTIONS & CONTACT INFO

Location

Events

Join the Makerspace email list

Teach Your Class at the Makerspace

Host your Event at the Makerspace

←

IMITLESS

# FIRST THING'S FIRST!

- Git  $\neq$  GitHub



- Git is **Form** of Version Control Software
  - AKA source control
  - Many other forms
- GitHub is a place to host your Git Repositories online
  - Other websites offer this too like BitBucket

# Ways to use Git

- There are pretty much two main ways to use Git
  - Command Lines
    - **Pros:** You are really forced to understand Git
    - **Cons:** You are really forced to understand Git
  - GUI Tools (GitHub Desktop, Git Extensions, etc)
    - **Pros:** Way easier to use and manage code
    - **Cons:** This is how you look to some people

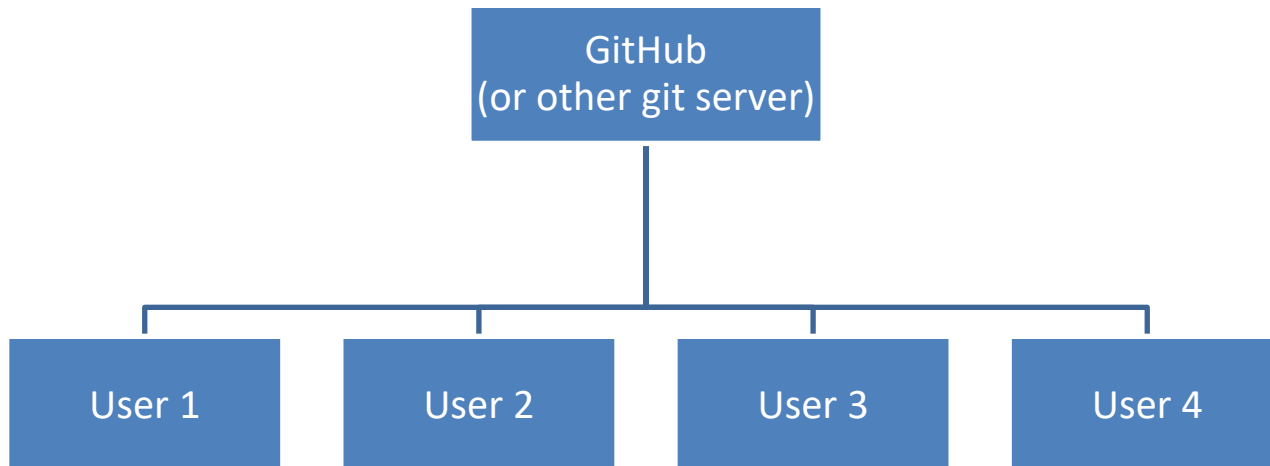


# Gotta Get Git

- Command Line
  - <https://git-scm.com/downloads>
    - Git Bash is just a Windows side application that opens a command prompt with Git
- GitHub Desktop GUI (what I recommend at first)
  - <https://desktop.github.com/>
- Make a GitHub account too!
  - Use Student email if possible

# Git 101- What happens to code

- Repository (the code) is saved on local computer
- Repositories can then be **Pushed** to a remote server where other people can push too



# How Git Works

- Keeps tracks of difference in lines of files

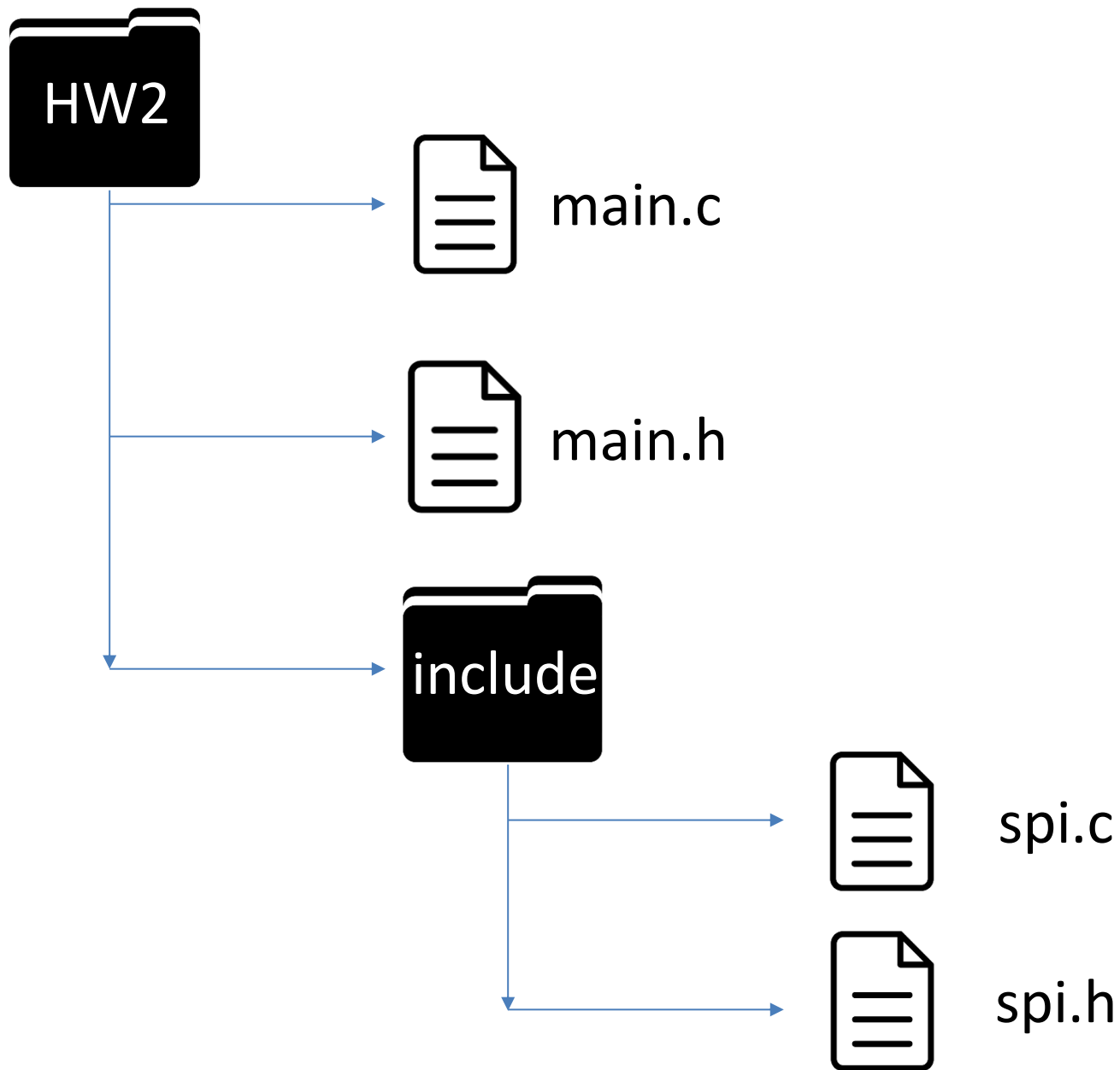
The screenshot displays a code diff tool interface with two panels. The top panel shows the diff for 'demo/demo.cpp', and the bottom panel shows the diff for 'server/server.c'. Each panel has a 'View' button in the top right corner. The diff tool uses color-coding: red for deletions, green for additions, and blue for context lines. The left side of each panel shows the original code, and the right side shows the modified code. The changes are highlighted with colored backgrounds.

**demo/demo.cpp**

```
@@ -18,13 +18,18 @@ void on_new_test(char* body) {
18 void onJoin(int uid){ printf("Join: %d\n", uid); }
19 void onLeave(int uid){ printf("Leave: %d\n", uid); }
20
21 -int main() {
22
23
24 WebSocket client_socket;
25 cout << "START" << endl;
26
27 - client_socket.connectSocket("192.168.1.105", 5000);
28
29 client_socket.setEvent(1, on_new_type);
30 client_socket.setEvent(2, on_new_color);
31
32 +int main(int argc, char* argv[]) {
33
34 + if (argc < 3) {
35 + printf("\n/demo <ip> <port>\n\n");
36 + exit(1);
37 + }
38 +
39 WebSocket client_socket;
40 cout << "START" << endl;
41
42 + client_socket.connectSocket(argv[1], atoi(argv[2]));
43
44 client_socket.setEvent(1, on_new_type);
45 client_socket.setEvent(2, on_new_color);
```

**server/server.c**

```
@@ -19,7 +19,7 @@
19 #define MAX_CLIENTS 16
20 #define DEFAULT_PORT 5000
21 #define MAX_MESSAGE_BUFFER 1024
22 -#define MAX_MESSAGE_KEYS 16
23
24 +#define DEFAULT_MAX_KEYS 16
```





# • HW2

— main.c

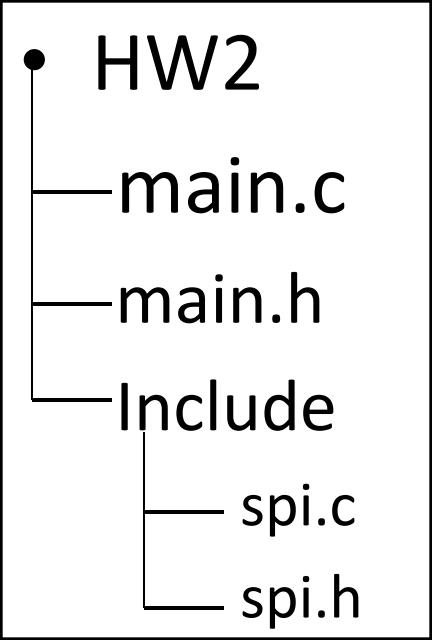
— main.h

— Include

— spi.c

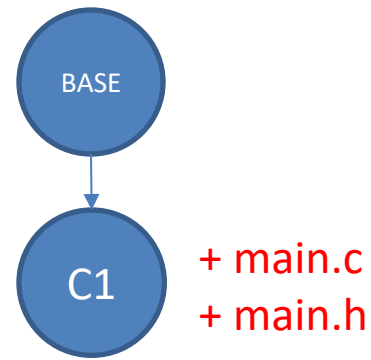
— spi.h

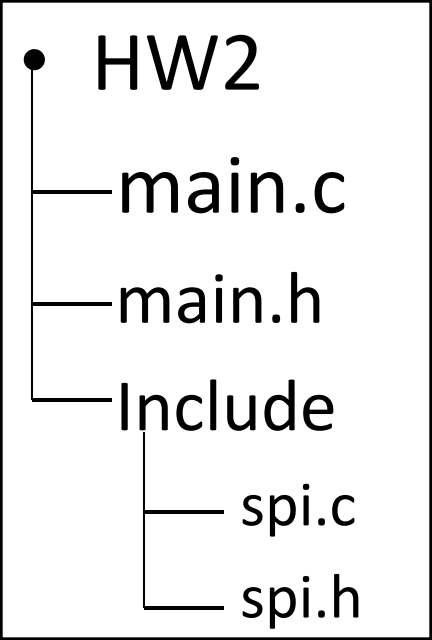
BASE



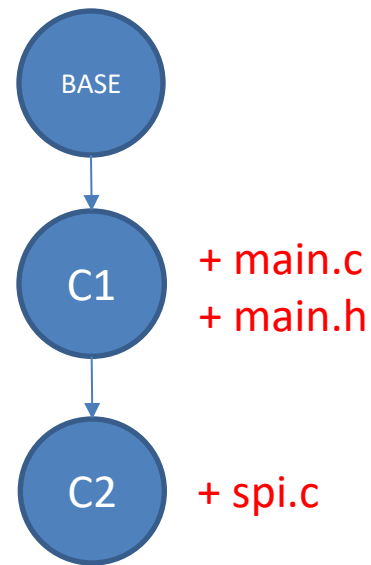
Change file

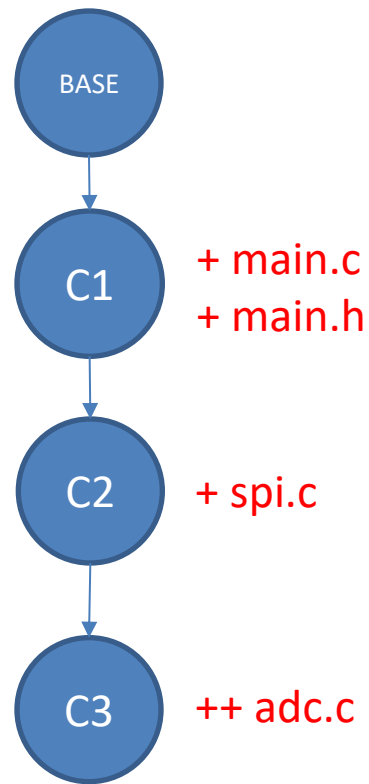
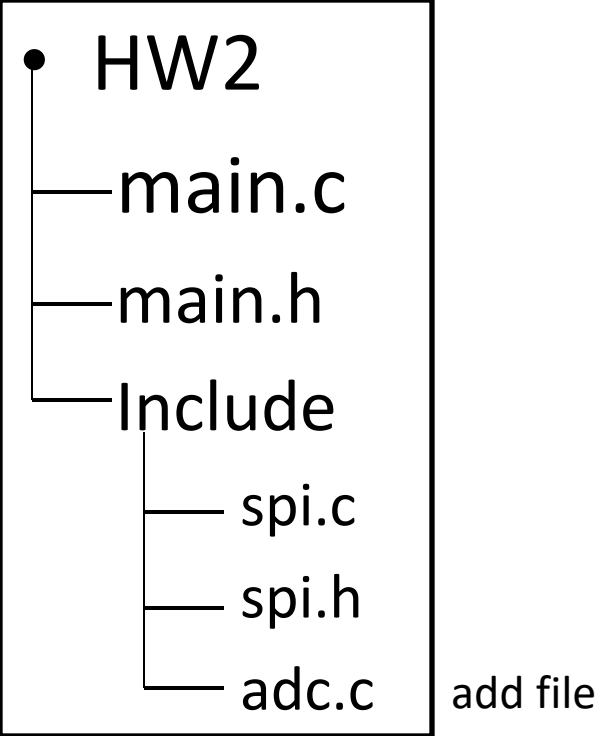
Change file

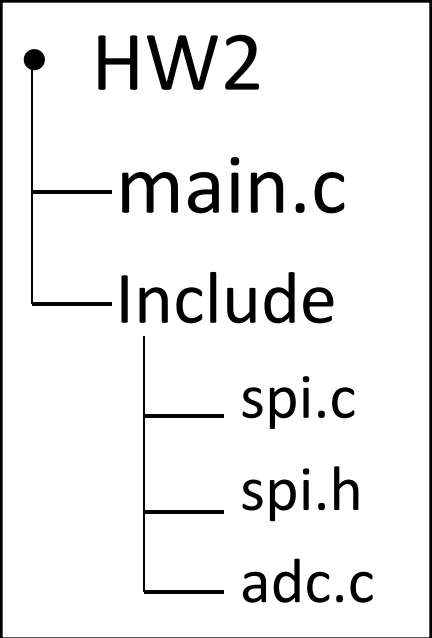




Change file

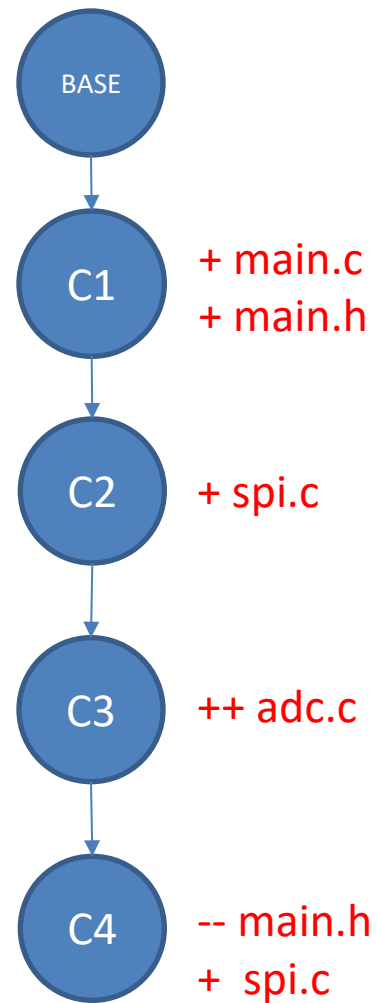






Removed file

Change file





**main.c**

v0

v1

v1

v1

v1

**main.h**

v0

v1

v1

v1

---

**Include/spi.c**

v0

v0

v1

v1

v2

**Include/spi.h**

v0

v0

v0

v0

v0

**Include/adc.c**

---

---

---

v0

v0

# It Remembers

## ...so you don't have too

- Git saves all past **Committed** saves in a .git file in the repository
- New people can go back to ANY old **Commit** made during life of repository
- It's the history of the project

# You can even “blame” others

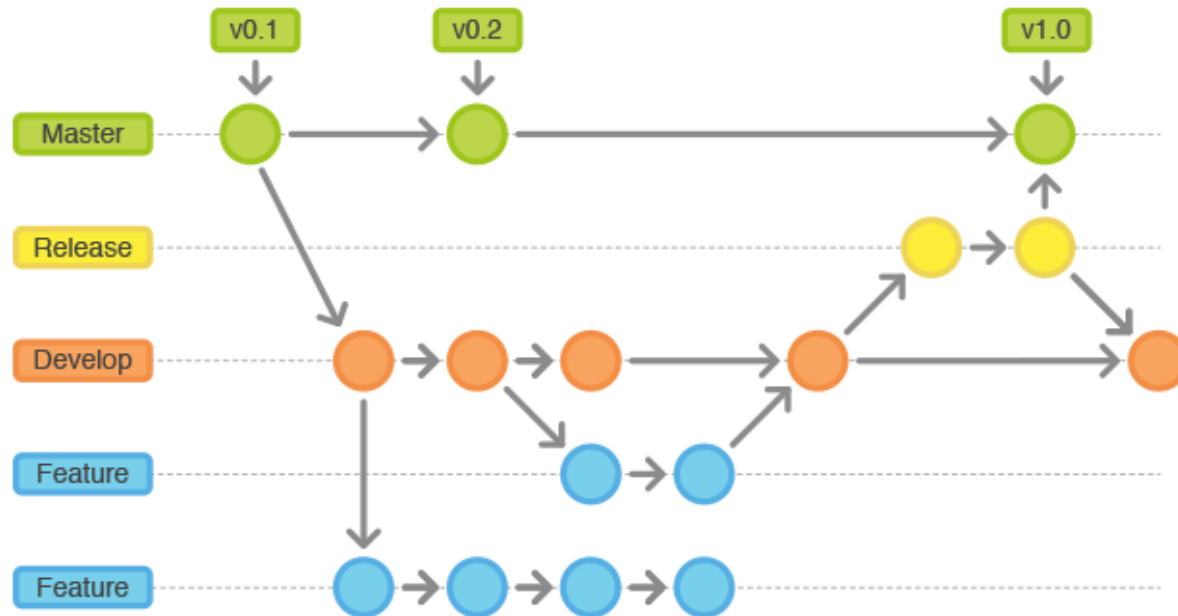
- Git’s blame feature allows you to see each line last change

Commit	Author	Time	Code
fscrypt: factor out bio specific functions	3 months ago	46	<code>struct workqueue_struct *fscrypt_read_workqueue;</code>
fs crypto: move per-file encryption from..	2 years ago	47	<code>static DEFINE_MUTEX(fscrypt_init_mutex);</code>
		48	
		49	<code>static struct kmem_cache *fscrypt_ctx_cache;</code>
		50	<code>struct kmem_cache *fscrypt_info_cache;</code>
		51	
		52	<code>/**</code>
		53	<code> * fscrypt_release_ctx() - Releases an encryption context</code>
		54	<code> * @ctx: The encryption context to release.</code>
		55	<code> *</code>
		56	<code> * If the encryption context was allocated from the pre-allocated pool, returns</code>
		57	<code> * it to that pool. Else, frees it.</code>
		58	<code> *</code>
		59	<code> * If there's a bounce page in the context, this frees that.</code>
		60	<code> */</code>
		61	<code>void fscrypt_release_ctx(struct fscrypt_ctx *ctx)</code>
		62	<code>{</code>
		63	<code>    unsigned long flags;</code>
		64	
fscrypt: Rename FS_WRITE_PATH_FL to ...	4 months ago	65	<code>if (ctx-&gt;flags &amp; FS_CTX_HAS_BOUNCE_BUFFER_FL &amp;&amp; ctx-&gt;w.bounce_page) {</code>
fs crypto: move per-file encryption from..	2 years ago	66	<code>    mempool_free(ctx-&gt;w.bounce_page, fscrypt_bounce_page_pool);</code>
		67	<code>    ctx-&gt;w.bounce_page = NULL;</code>
		68	<code>}</code>



# Git 101 - Branches

- Branches let you work on the code in your own crazy direction and **Merge** it back later
- Example: Make a “New-Feature” branch and when it is ready, **Merge** back to the Master Branch



# Git 101 – Workflow

- **Fetch/Pull**
- Make your edits
- **Stage** your changes
- **Commit** your work
- **Push**

# Git 101 – Workflow - Fetch

- If you have not cloned the repo  
`git clone https://the.git.repo.git`
- Get the latest updates before working  
`git pull`
- Don't make folder then git clone
  - Cloning makes a new folder for you

# Git 101 – Workflow - Edit

- Add, remove, edit all the files you want

# Git 101 – Workflow - Stage

- Add the files you want to commit
- This allows you to choice **what** to commit

`git add -A`

- Will add all difference

`git add main.c`

- Just adds main.c

`git add myFolder/*`

- Adds entire folder

# Git 101 – Workflow – Commit

- Take the “snapshot” of the repo
- Add a commit title

```
git commit -m “Best commit EVA”
```

- Optionally add a comment

```
git commit -m “Best commit EVA” -m “Here is more  
details”
```

# Git 101 – Workflow - Push

- When ready, push changes to server

`git push`

# GitHub Permissions

- For a project you can always just download the code and do what you want.
- If you want to make changes, either **Clone** or **Fork** the Repository
  - You can then send a **Pull Request** that will let someone in charge of Repo check your changes and **Merge** it
- If you set someone as a **Collaborator** they can **Push** code without having to submit a **Pull Request**

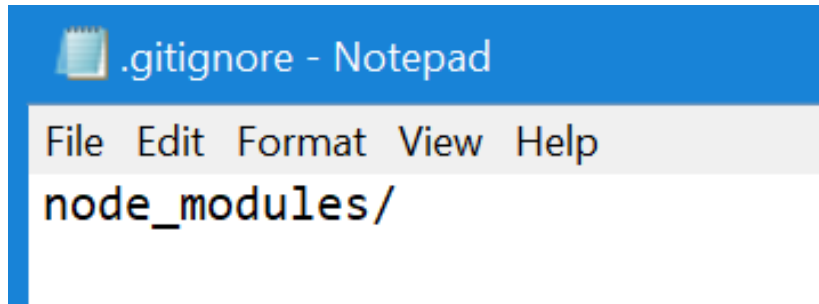


# Almost Forgot about .gitignore

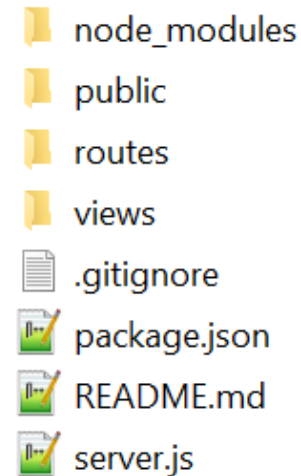
- Git is not the best with Binary files (.mp3, .pdf, .exe, .FileTypeCannotReadInNotepad)
- List all the folders and files that git will not recognize
- Almost all types of projects have a standard .gitignore template found on GitHub

# .gitignore example

- We don't want the node\_modules folder committed
- Add it to .gitignore



```
.gitignore - Notepad
File Edit Format View Help
node_modules/
```



# Merge Conflicts

- Not that scary
- Happens when same lines are altered in two different commits
- Debunking the myth that merge conflicts are hard

## Commit from User 1

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define VALUE 40
5
6  int main(int argc, char* argv[]) {
7
8      int a = 5;
9
10     if ( a > 6 ) {
11         printf("a is big");
12     }
13
14     return 0;
15
16 }
```

## Commit from User 2

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define VALUE 40
5
6  int main(int argc, char* argv[]) {
7
8      int a = 5;
9
10     if ( a > 4 ) {
11         printf("a is small");
12     }
13
14     return 0;
15
16 }
```

Merge Conflict will need to be resolved before able to merge

Just open the file in text editor

## The Merge Conflict File

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define VALUE 40
5
6  int main(int argc, char* argv[]) {
7
8      int a = 5;
9
10     <<<<<< commit/user1
11     if ( a > 4 ) {
12         printf("a is small");
13     }
14     if ( a > 6 ) {
15         printf("a is big");
16     }
17     >>>>>> Commit/user2
18
19     return 0;
20
21 }
```

Change file to look the way you want and save

...simple, I know

# GitHub GUI – Make new repo

- Can either create online or bring it to local computer.
- Start new repository and push to GitHub when ready.
- Can take a current set of code and make it into a Git repo.
  - Will have no history prior to initialization of git

# Recap

- Get Repository (**Clone**) or Sync it (**Fetch**)
- Know which **Branch** you are in
- Make your changes
- Set which changes you are **Staging**
- **Commit** the changes
- When ready, Sync or send a Pull Request

Time to get some practice!

**[tinyurl.com/GitWorkshopIEEE](https://tinyurl.com/GitWorkshopIEEE)**