

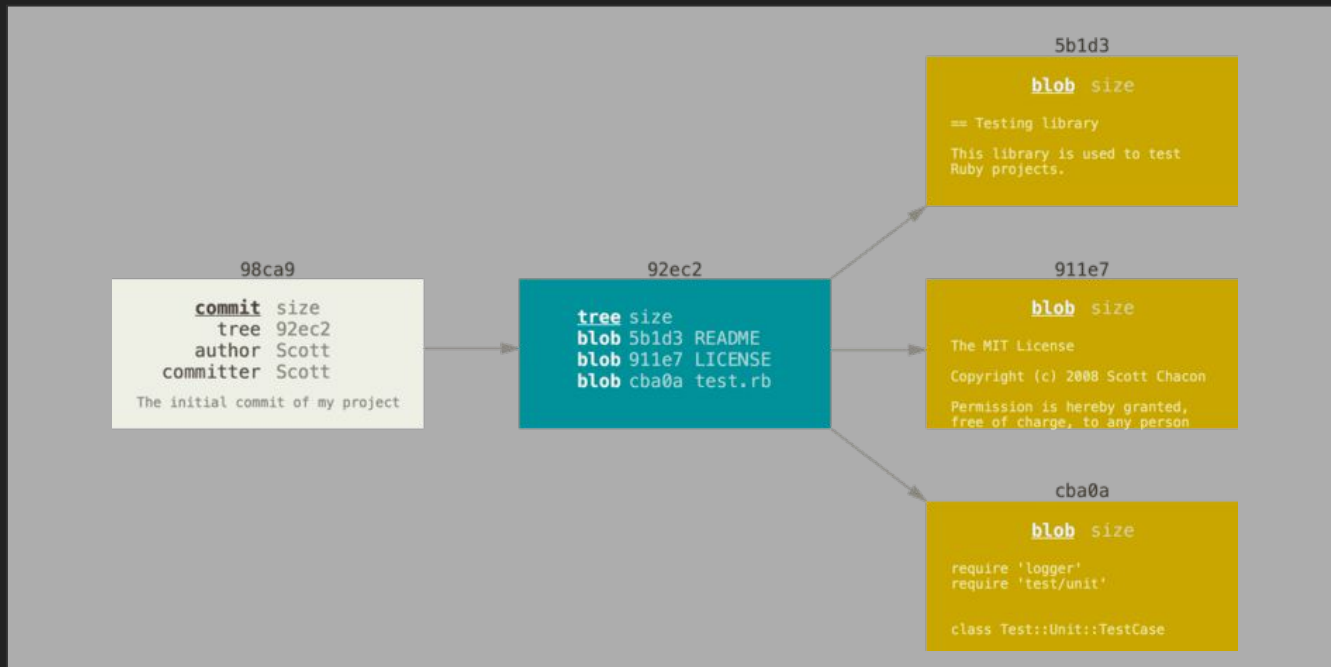


# Branch, Merge and Pull Request

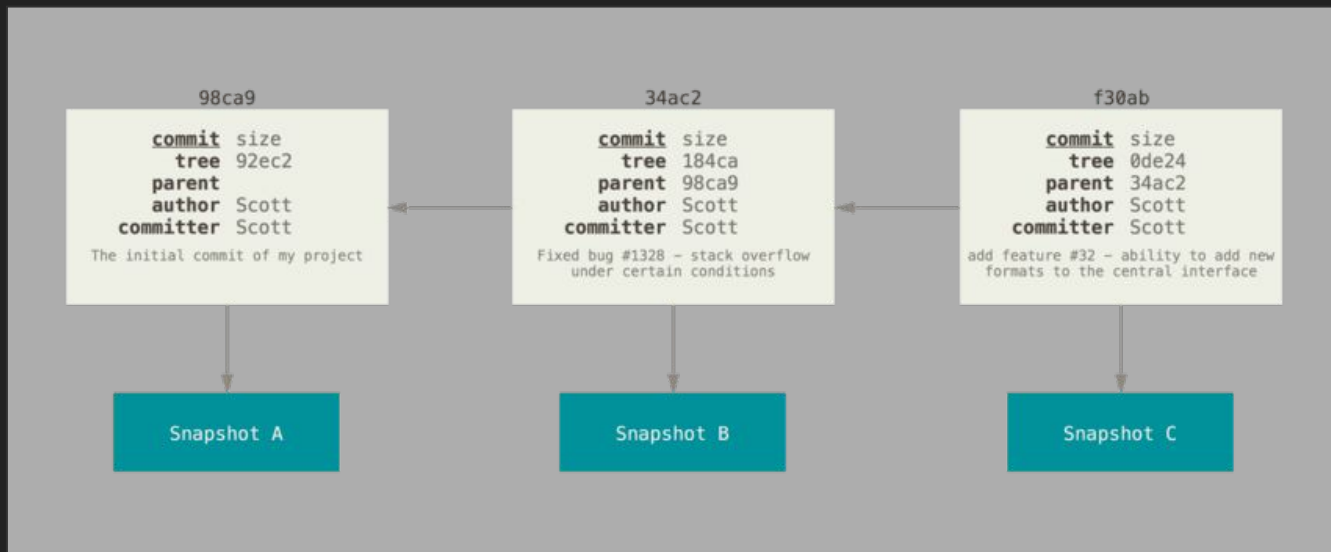
*Git killer features...*



# How Git Stores its Data



# Git Commit Command

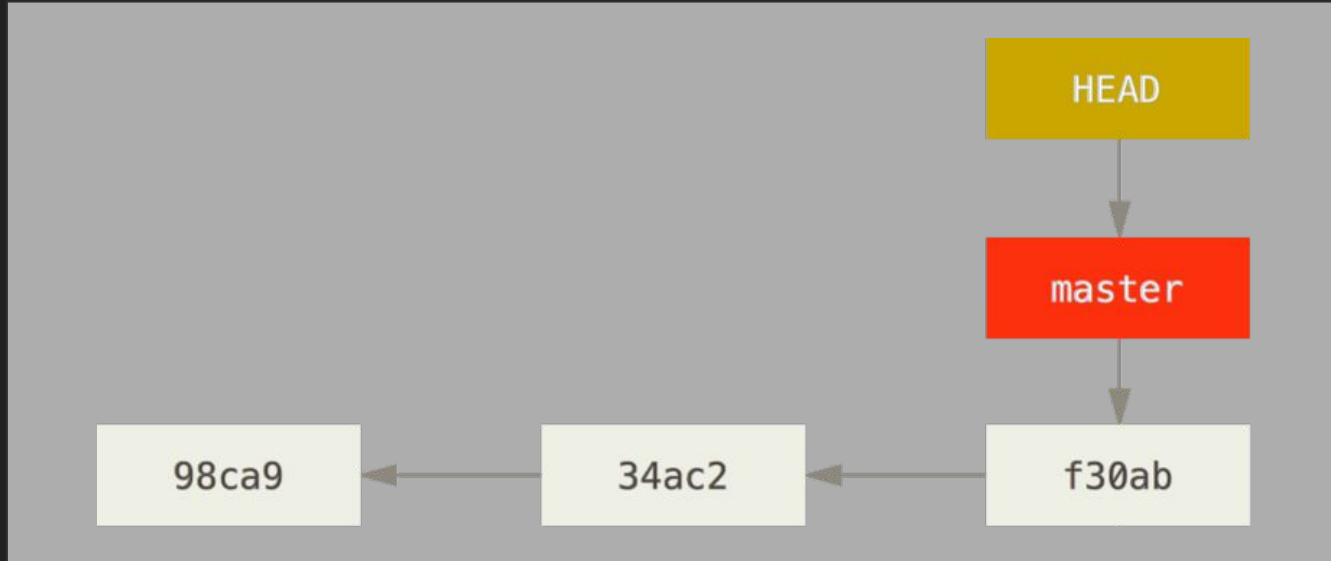


```
git commit -m 'The initial commit of my project'
```

```
git commit -m 'Fixed bug #1328 - stack overflow under certain conditions'
```

```
git commit -m 'add feature #32 - ability to add new formats to the central interface'
```

# Head



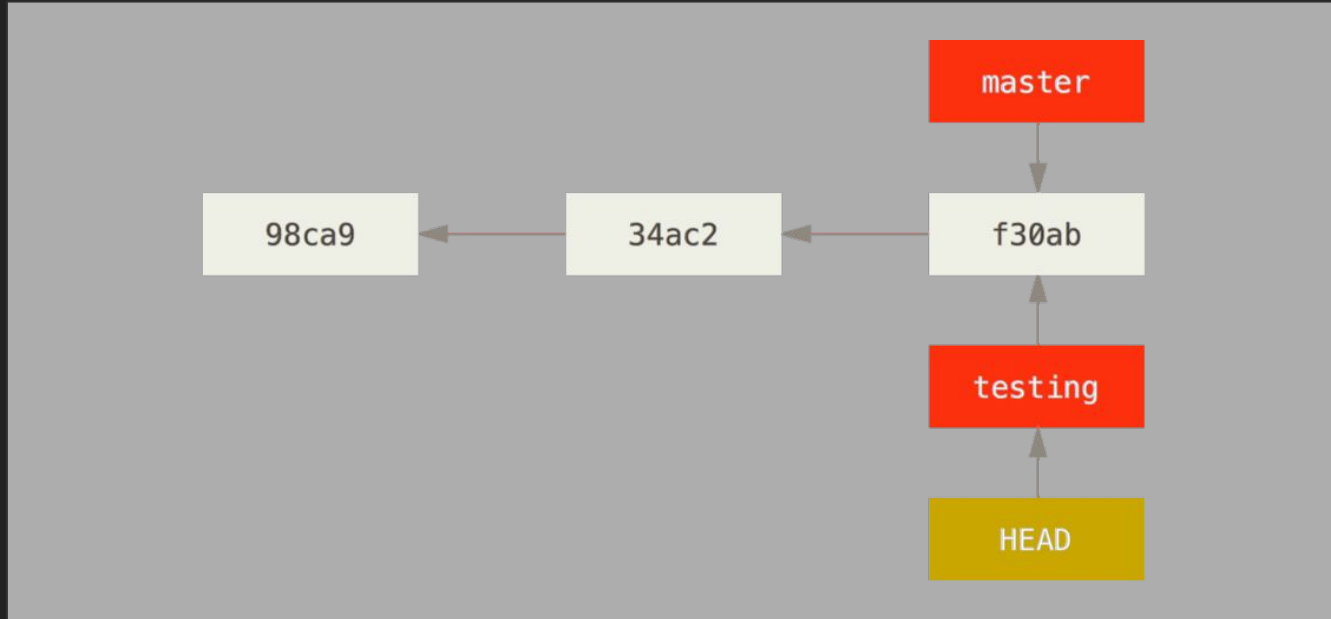
Head is a pointer to which branch you are right now

# Git Branch Command



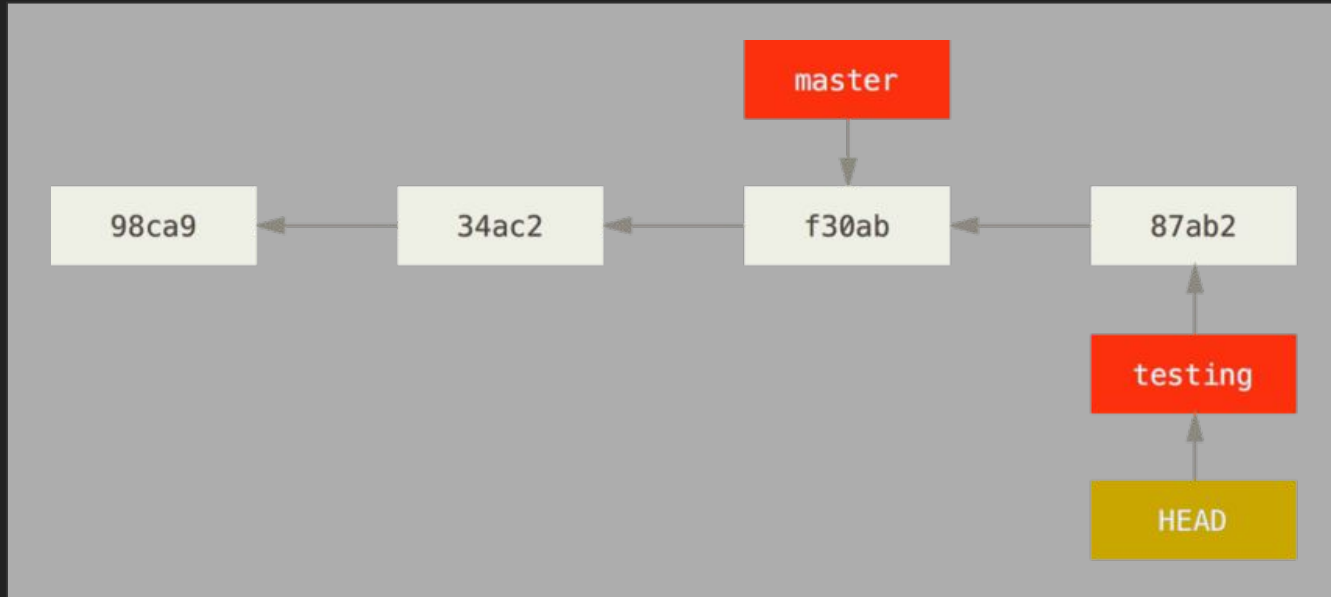
```
git branch testing
```

# Git Checkout Command



```
git checkout testing
```

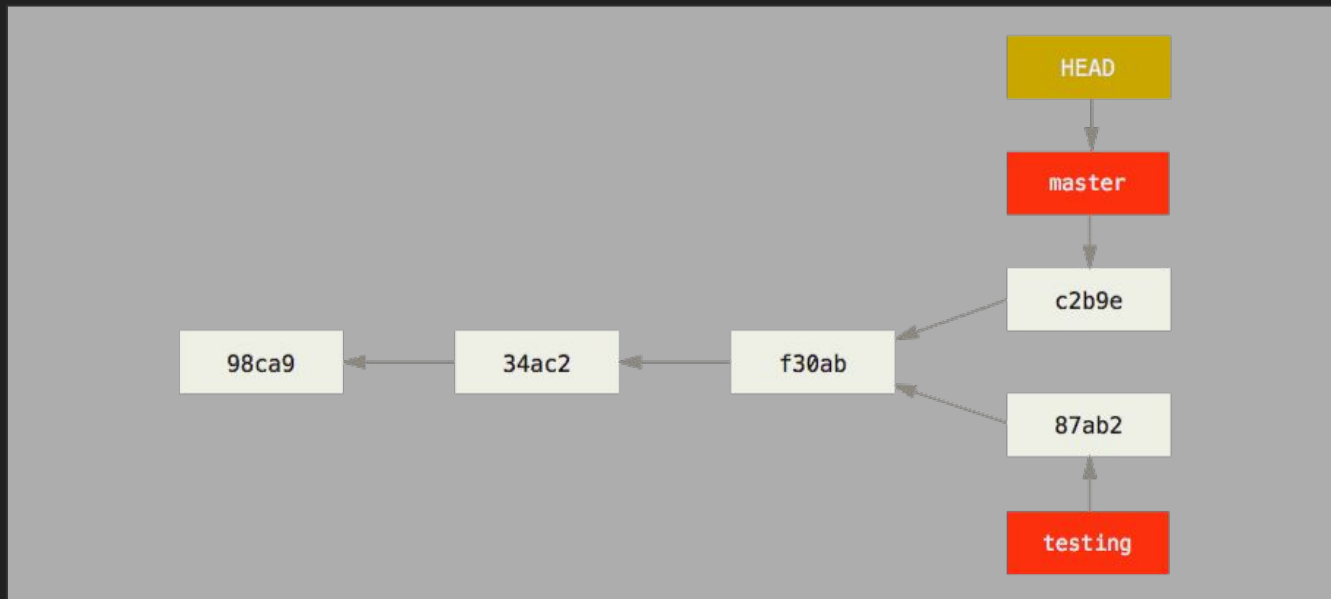
# Committing to a Branch



After you commit to testing branch, testing will move forward but master will stay behind. What happened to the files when you switch back the HEAD to master?



# Divergent History



After you commit to master, your Git repository will have a divergent history like above.

# Git Log Command

```
$ git log --oneline --decorate --graph --all
* c2b9e (HEAD, master) Made other changes
| * 87ab2 (testing) Made a change
|/
* f30ab Add feature #32 - ability to add new formats to the central
interface
* 34ac2 Fix bug #1328 - stack overflow under certain conditions
* 98ca9 initial commit of my project
```



# Scenario 1

1. You have a **perfectly working source code**,
2. You want to add a new feature but do not want to alter directly the **perfectly working source code** in fear of breaking it,
3. You will create a **branch** of the **perfectly working source code**, then you will add the feature in the **branch**,
4. After you test the newly created feature and 100% sure that it is working, you want to put the changes you made into the **perfectly working source code**, by merging it to the **branch**.

# Scenario 1.1

```
$ mkdir git_branch_demo
$ cd git_branch_demo/
$ git init
$ touch hello.js
$ code hello.js
$ git add .
$ git commit -m 'first commit'
$ code hello.js
$ git add .
$ git commit -m 'add hello variable'
$ code hello.js
$ git add .
$ git commit -m 'add function printHello'
```

```
console.log('hello');
```

```
var hello = 'hello';
```

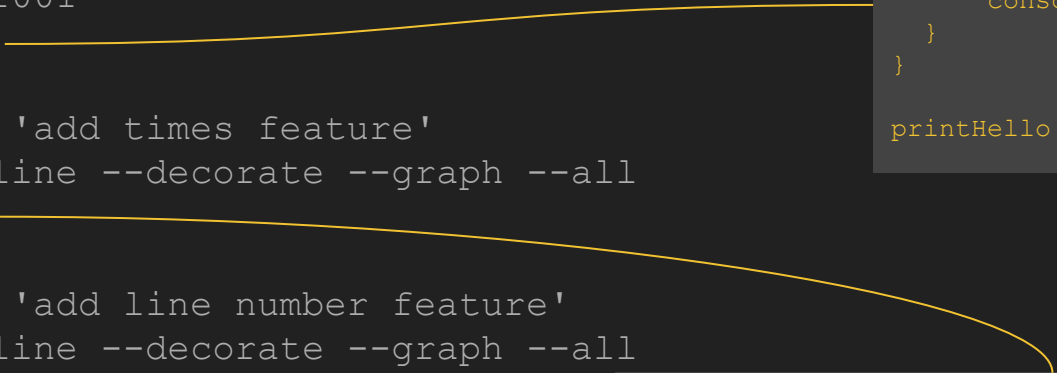
```
console.log(hello);
```

```
var hello = 'hello';
```

```
function printHello(sIn) {
  console.log(sIn);
}
```

# Scenario 1.2 and 1.3

```
$ git branch f001
$ git log --oneline --decorate --graph --all
$ git checkout f001
$ code hello.js
$ git add .
$ git commit -m 'add times feature'
$ git log --oneline --decorate --graph --all
$ code hello.js
$ git add .
$ git commit -m 'add line number feature'
$ git log --oneline --decorate --graph --all
```



```
var hello = 'hello';
```

```
function printHello(sIn, nTimes) {
  for (var i = 0; i < nTimes; i++) {
    console.log(sIn);
  }
}
```

```
printHello(hello, 10);
```


```
var hello = 'hello';
```

```
function printHello(sIn, nTimes, bLine) {
  for (var i = 1; i <= nTimes; i++) {
    console.log(`${bLine ? i + ':' : {}} ${sIn}`);
  }
}
```

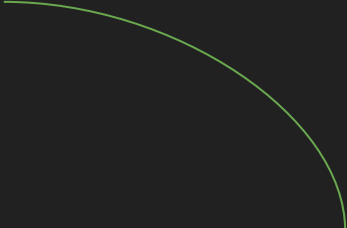
```
printHello(hello, 10, true);
```

# Scenario 1.4

```
$ git checkout master
$ git merge f001
$ git log --oneline --decorate --graph --all
$ git branch --delete f001
$ git log --oneline --decorate --graph --all
```



```
* e344cd4 (HEAD -> master, f001) add line number feature
* f7f0410 add times feature
* 68e8a96 add function printHello
* 3d27607 add hello variable
* 523cef2 first commit
```



```
* e344cd4 (HEAD -> master) add line number feature
* f7f0410 add times feature
* 68e8a96 add function printHello
* 3d27607 add hello variable
* 523cef2 first commit
```

## Scenario 2

1. You have a **working source code**,
2. You want to add a new feature but do not want to alter directly the **working source code** in fear of breaking it,
3. You will create a **branch** of the **working source code**,
4. Suddenly a customer found a problem in your **working source code**, then you decided to make a hotfix for it,
5. After the hotfix has been made, you continue your work on the feature,
6. After you test the newly created feature and 100% sure that it is working, you want to put the changes you made into the **working source code (hotfix has been applied)**, by merging it to the **branch**.



## Scenario 2.1

```
$ mkdir git_merge_demo
$ cd git_merge_demo/
$ git init
$ touch hello.js
$ code hello.js
$ git add .
$ git commit -m 'first commit'
```



```
var hello = 'hello';

function printHello(sIn, nTimes) {
    for (var i = 0; i <= nTimes; i++) {
        console.log(sIn);
    }
}

printHello(hello, 10);
```

## Scenario 2.2 and 2.3

```
$ git branch f001
$ git checkout f001
$ code hello.js
$ git add .
$ git commit -m 'add line number feature'
$ git log --oneline --decorate --graph --all
```

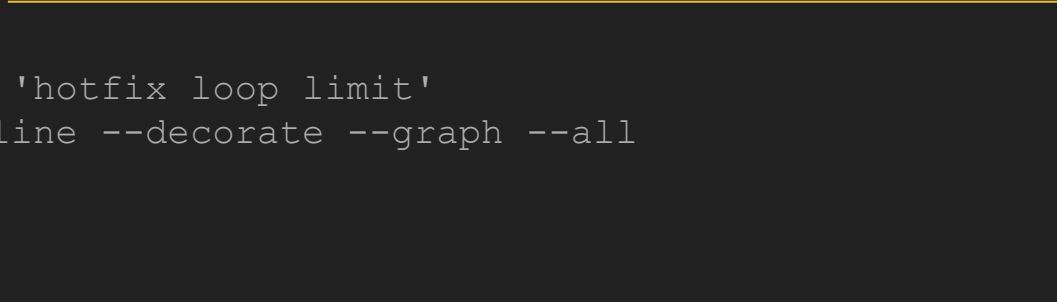
```
var hello = 'hello';

function printHello(sIn, nTimes, bLine) {
  for (var i = 0; i <= nTimes; i++) {
    console.log(`${bLine ? i + ':' : {}}
    ${sIn}`);
  }
}

printHello(hello, 10, true);
```

## Scenario 2.4

```
$ git checkout master  
$ code hello.js  
$ git add .  
$ git commit -m 'hotfix loop limit'  
$ git log --oneline --decorate --graph --all
```



```
var hello = 'hello';  
  
function printHello(sIn, nTimes) {  
    for (var i = 0; i < nTimes; i++) {  
        console.log(sIn);  
    }  
}  
  
printHello(hello, 10);
```

## Scenario 2.5

```
$ git checkout f001
$ code hello.js
$ git add .
$ git commit -m 'add line prefix'
$ git log --oneline --decorate --graph --all
```

```
var hello = 'hello';

function printHello(sIn, nTimes, bLine) {
  for (var i = 0; i <= nTimes; i++) {
    console.log(`${bLine ? 'line ' + i + ':' : {}}
    ${sIn}`);
  }
}

printHello(hello, 10, true);
```

# Scenario 2.6

```
$ git checkout master
$ git merge f001
$ code hello.js
$ git add .
$ git commit -m 'merge and fix conflicts'
$ git log --oneline --decorate --graph --all
```

Auto-merging hello.js  
CONFLICT (content): Merge conflict in hello.js  
Automatic merge failed; fix conflicts and then commit the result.

See next slide

```
* 675840d (HEAD -> master) merge and fix conflicts
| \
| * 74050f7 (f001) add line prefix
| * 6973481 add line number feature
* | 3db24c1 hotfix loop limit
|/
* 82ed675 first commit
```

# Editing Conflicts in VS Code

```
File Edit Selection View Go Run Terminal Help
JS hello.js x JS hello.js: Current Changes ↔ Incoming Changes
home > anpan > git_merge_demo > JS hello.js > ...
1 var hello = 'hello';
2
3 ===== HEAD (Current Change)
4 function printHello(sIn, nTimes) {
5   for (var i = 0; i < nTimes; i++) {
6     console.log(sIn);
7   }
8   =====
9   function printHello(sIn, nTimes, bLine) {
10     for (var i = 0; i <= nTimes; i++) {
11       console.log(`${bLine ? 'line ' + i + ':' : '{}'} ${sIn}`);
12     }
13   }
14
15 printHello(hello, 10, true);
```

Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes | Start Live Share Session

Try these options



# Scenario 3

1. You want to contribute to a **GitHub repository**,
2. Make a **clone** out of the **GitHub repository**,
3. Make a **branch** from the **cloned GitHub repository**,
4. Make the needed changes on the **branch**,
5. Push the **branch** to **GitHub repository**,
6. Create a **pull request**.



# Scenario 3.1 until 3.5

```
$ mkdir git_pull_request_demo
$ cd git_pull_request_demo/
$ git clone
git@github.com:phase-0-branch-exercises/melee-ranged-grouping.git
$ cd melee-ranged-grouping/
$ git checkout -b answer
$ touch answer.js
$ code answer.js
$ git add .
$ git commit -m 'commit answer'
$ git push -u origin answer
```

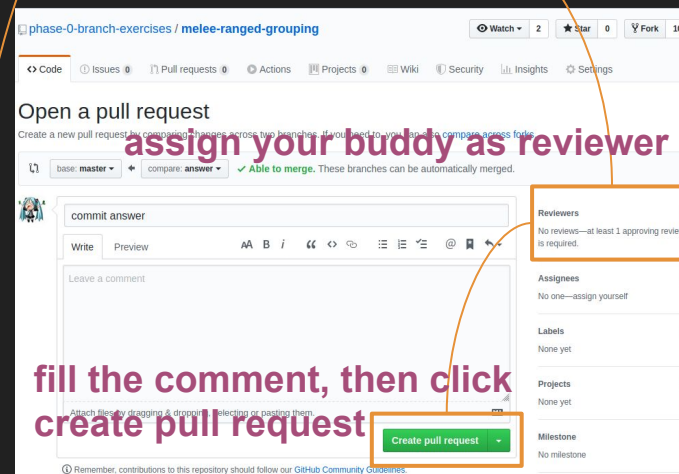
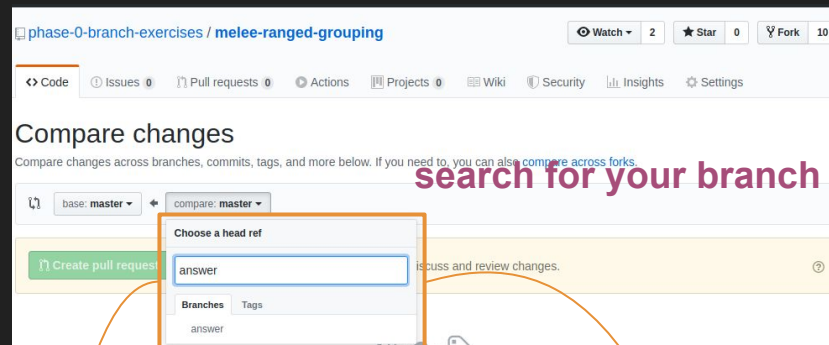
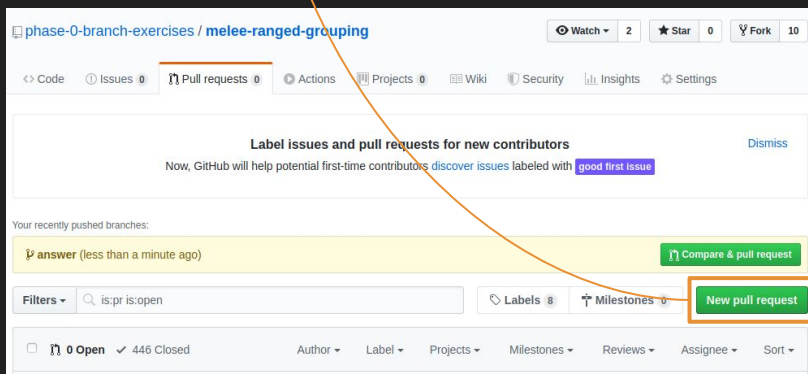
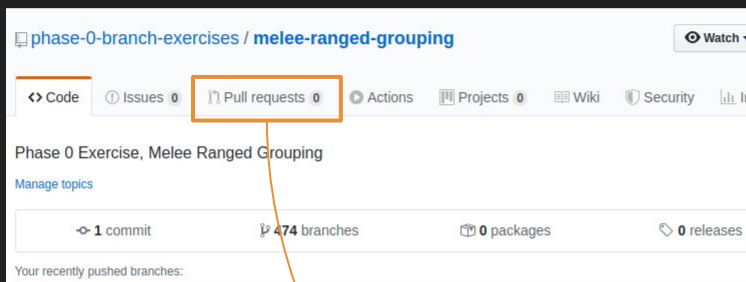
same as

```
$ git branch answer
$ git checkout answer
```

```
console.log('hello');
```

```
remote: Create a pull request for 'answer' on GitHub by visiting:
remote:   https://github.com/phase-0-branch-exercises/melee-ranged-grouping/pull/new/answer
```

# Scenario 3.6.1



# Scenario 3.6.2

Or simply just click the link in terminal

```
remote: Create a pull request for 'answer' on GitHub by visiting:  
remote: https://github.com/phase-0-branch-exercises/melee-ranged-grouping/pull/new/answer
```

**CLICK THIS!**