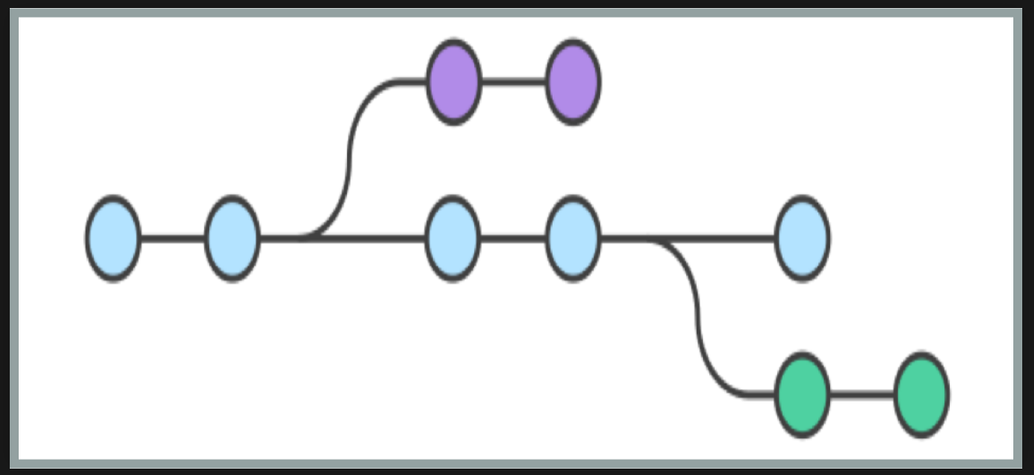
DevOps course by Ali Kahoot - Dice Analytics

BRANCHING BASICS

- Independent/isolated line of development
- Inexpensive/lightweight as compared to other VCS
- Way to work on different version of a repository
- Work in parallel
- Master is the default branch
- Branches are just pointers to commits
- Protected branches

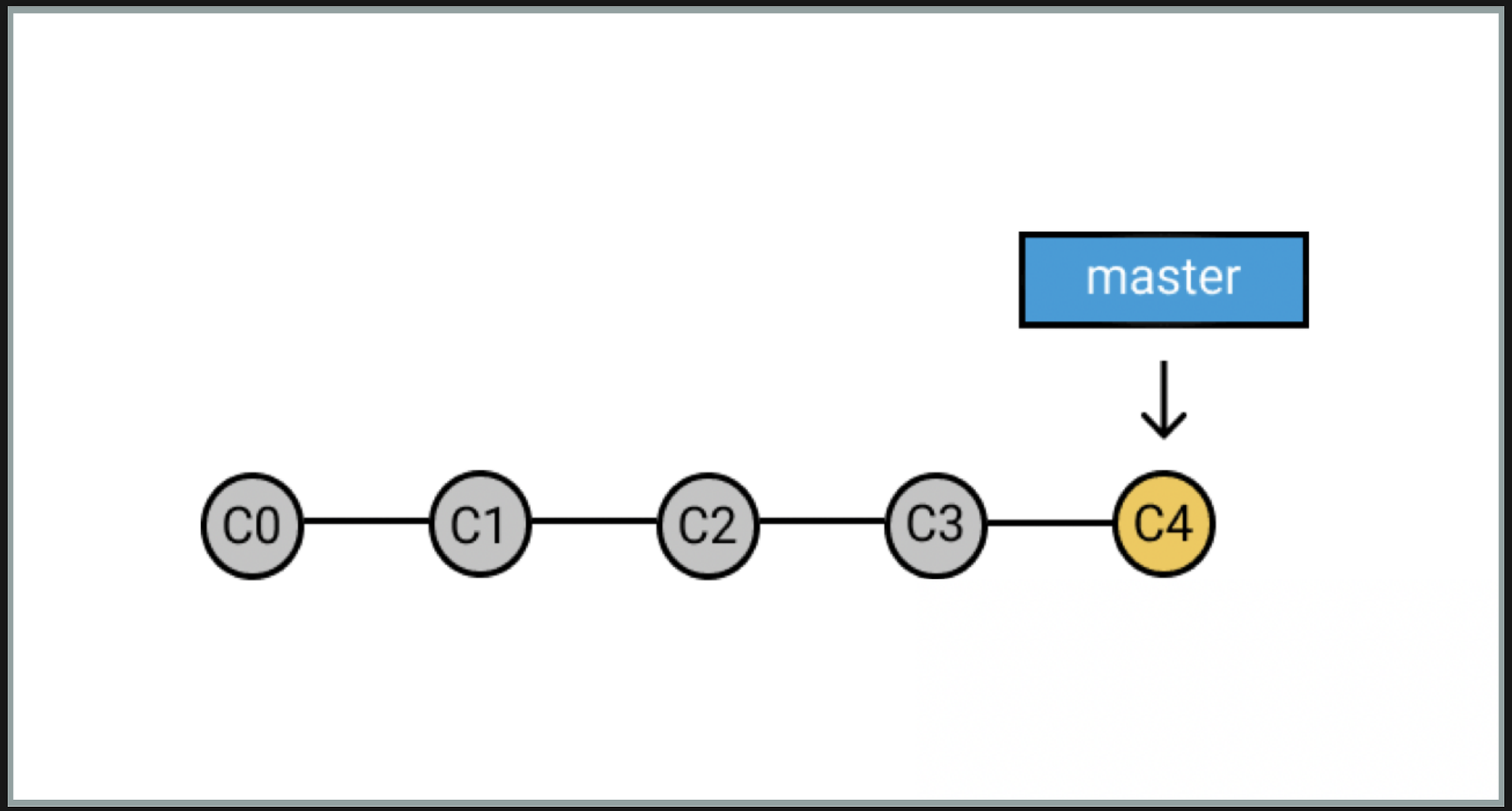


DevOps course by Ali Kahoot - Dice Analytics

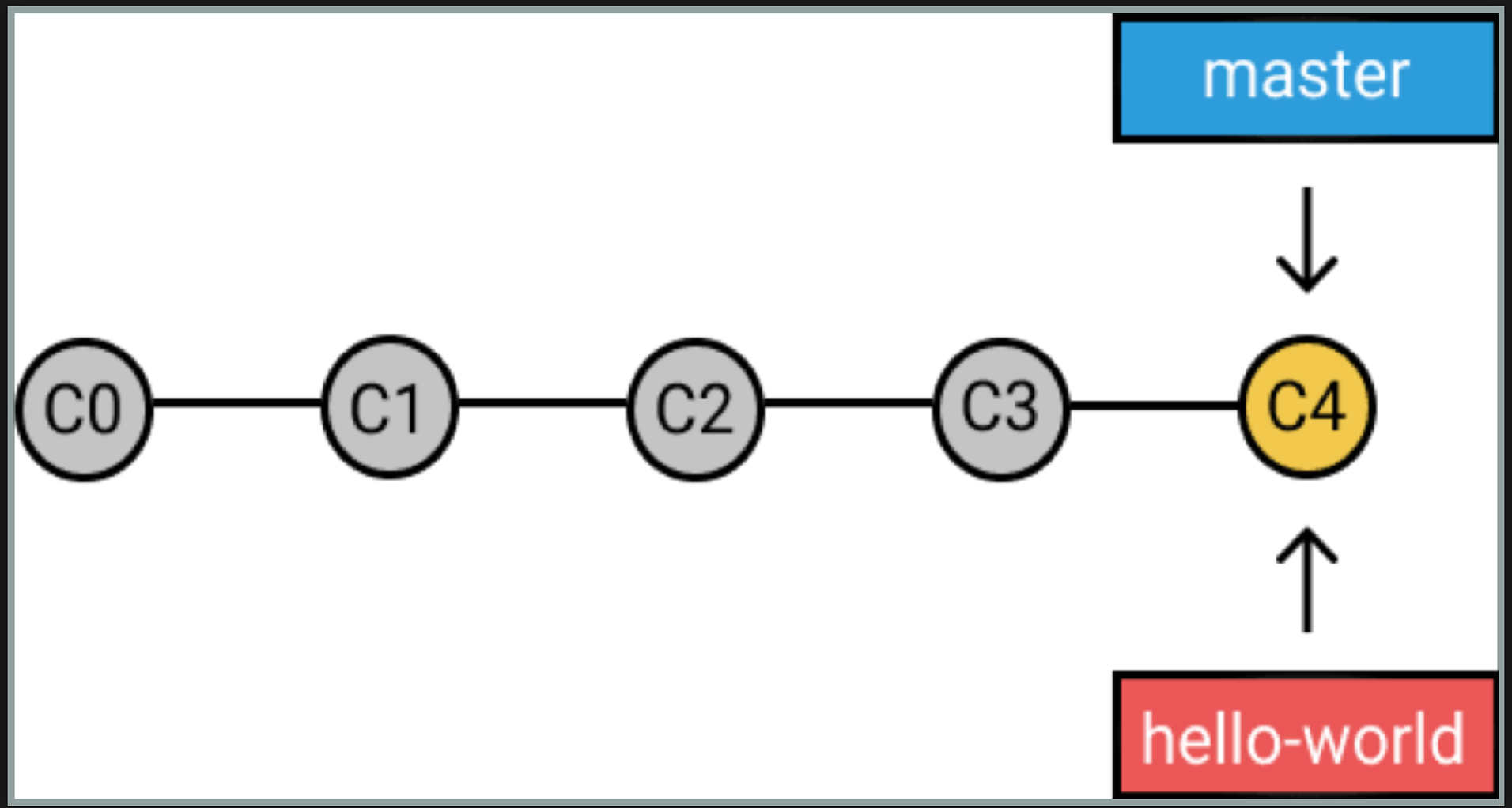
BRANCHING COMMANDS

- List Branches: git branch
- List all Branches: git branch -a
- Create Branch: git branch [branch_name]
- Delete Branch: git branch -d [branch_name]
- Delete remote Branch: git push origin --delete [branch_name]
- Create Branch and Switch: git checkout -b [branch_name]
- Switch Branch: git checkout [branch_name]

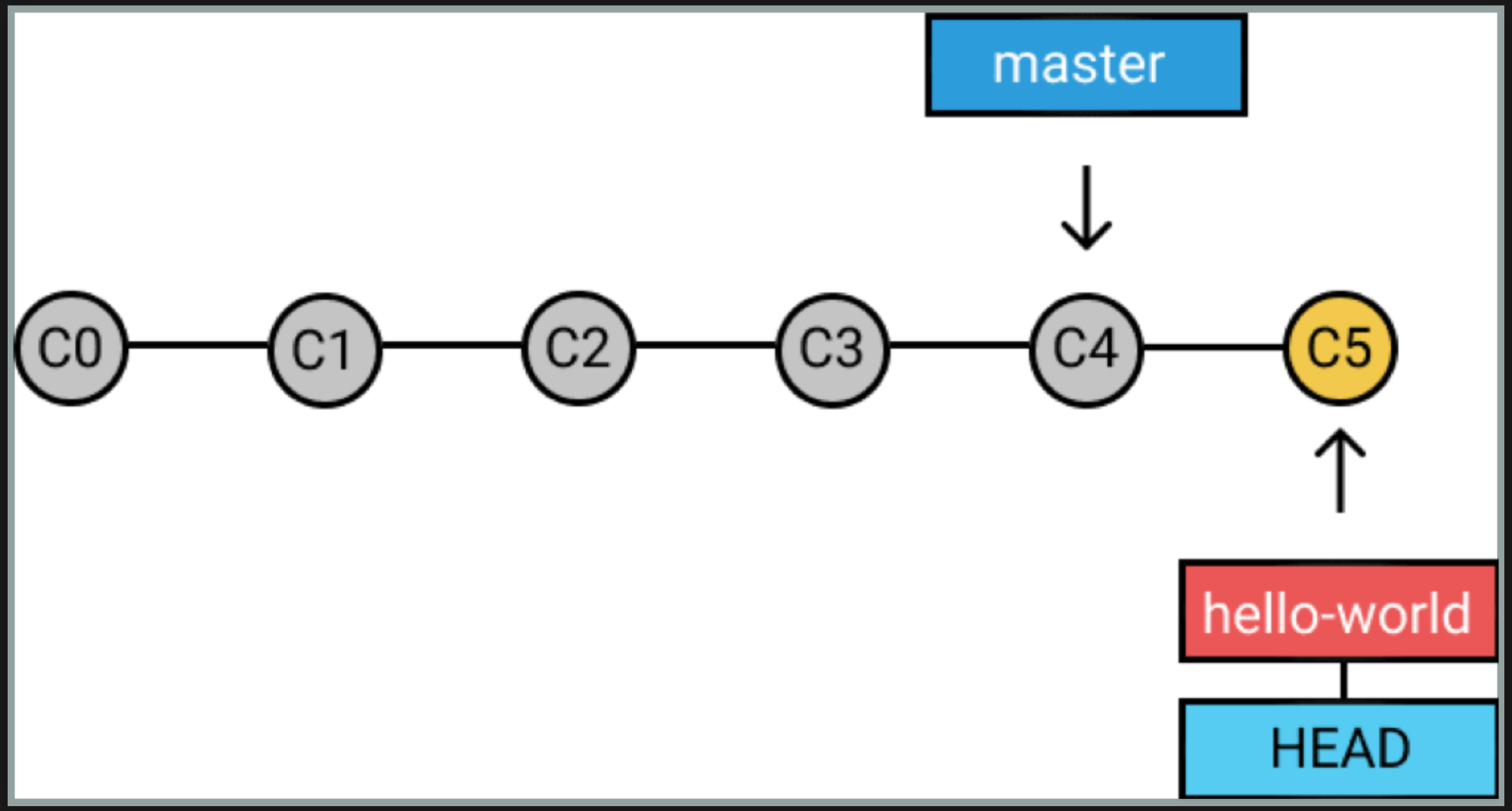
BRANCHING



BRANCHING

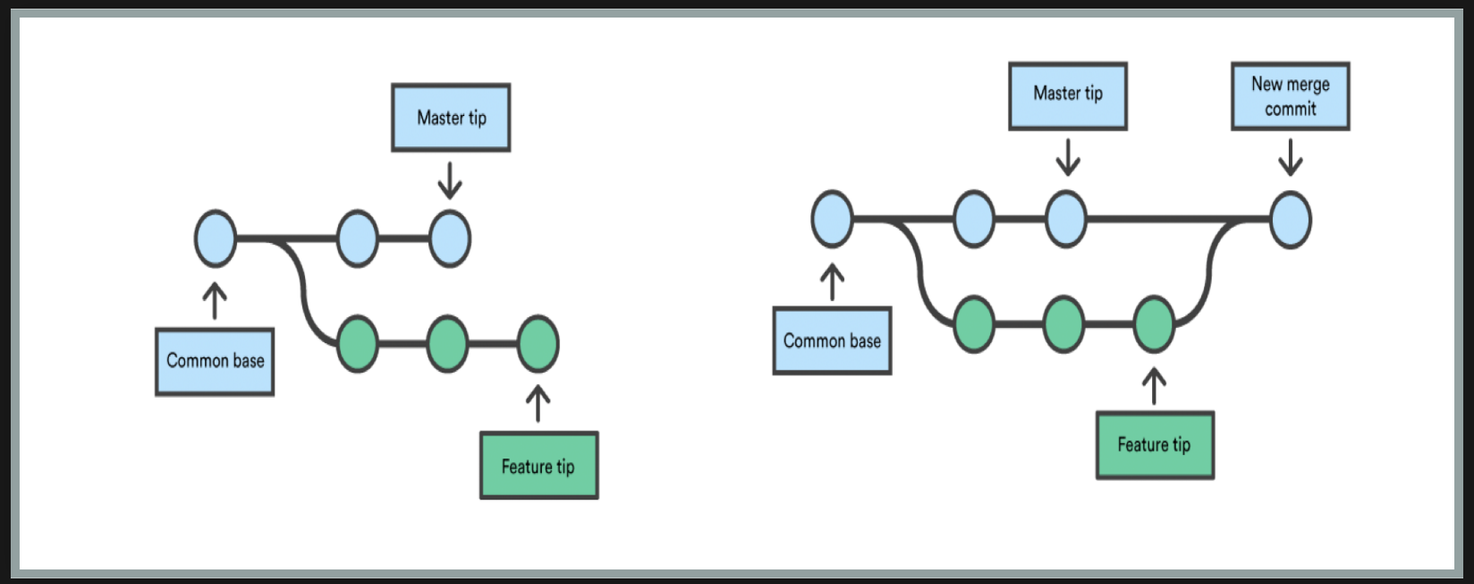


BRANCHING



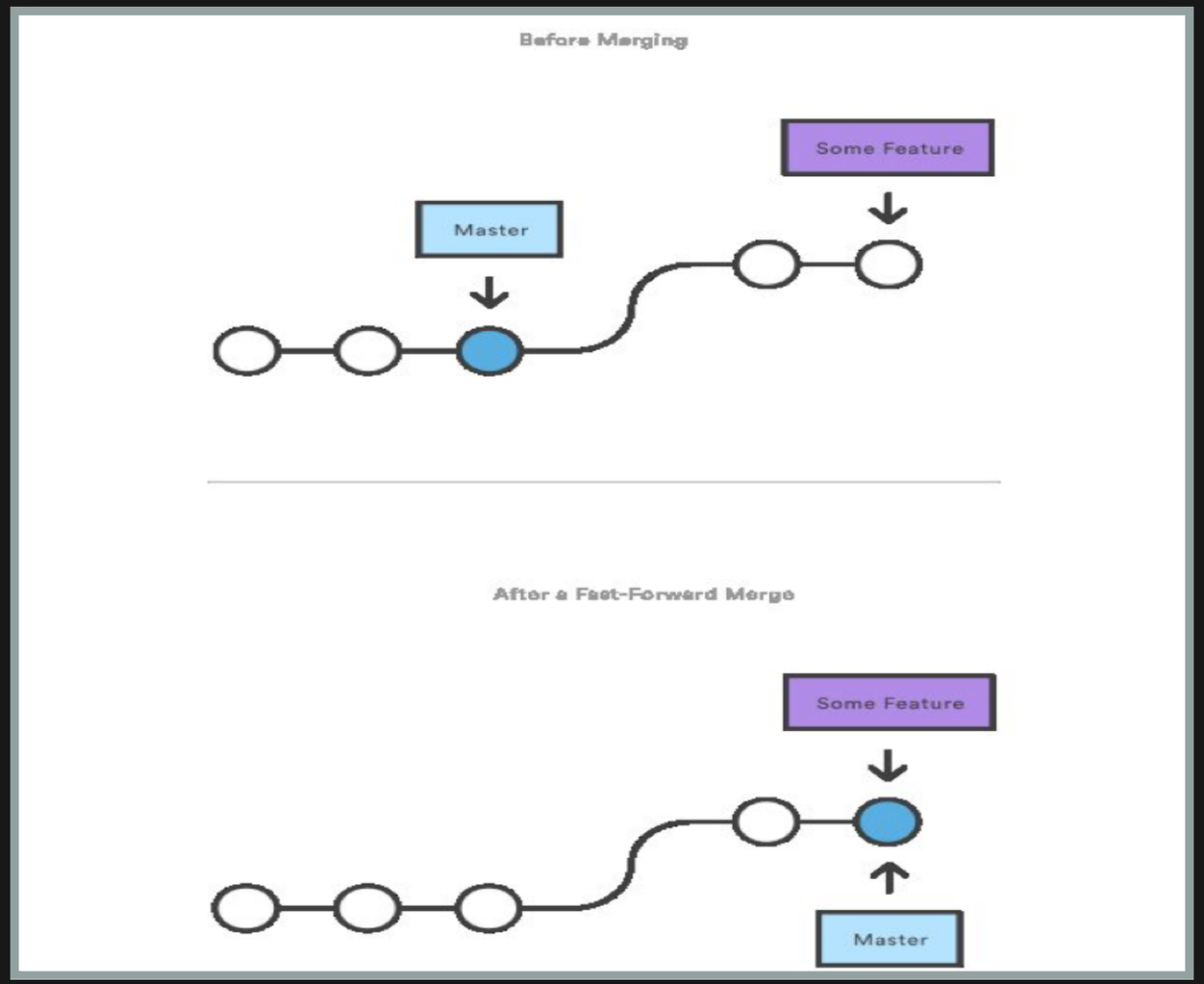
MERGING

- Combine multiple branches into one
- Combines multiple sequence into unified history
- Commands
  - Merge branch into active branch: git merge [branch name]
  - Merge branch into target branch: git merge [source branch] [target branch]

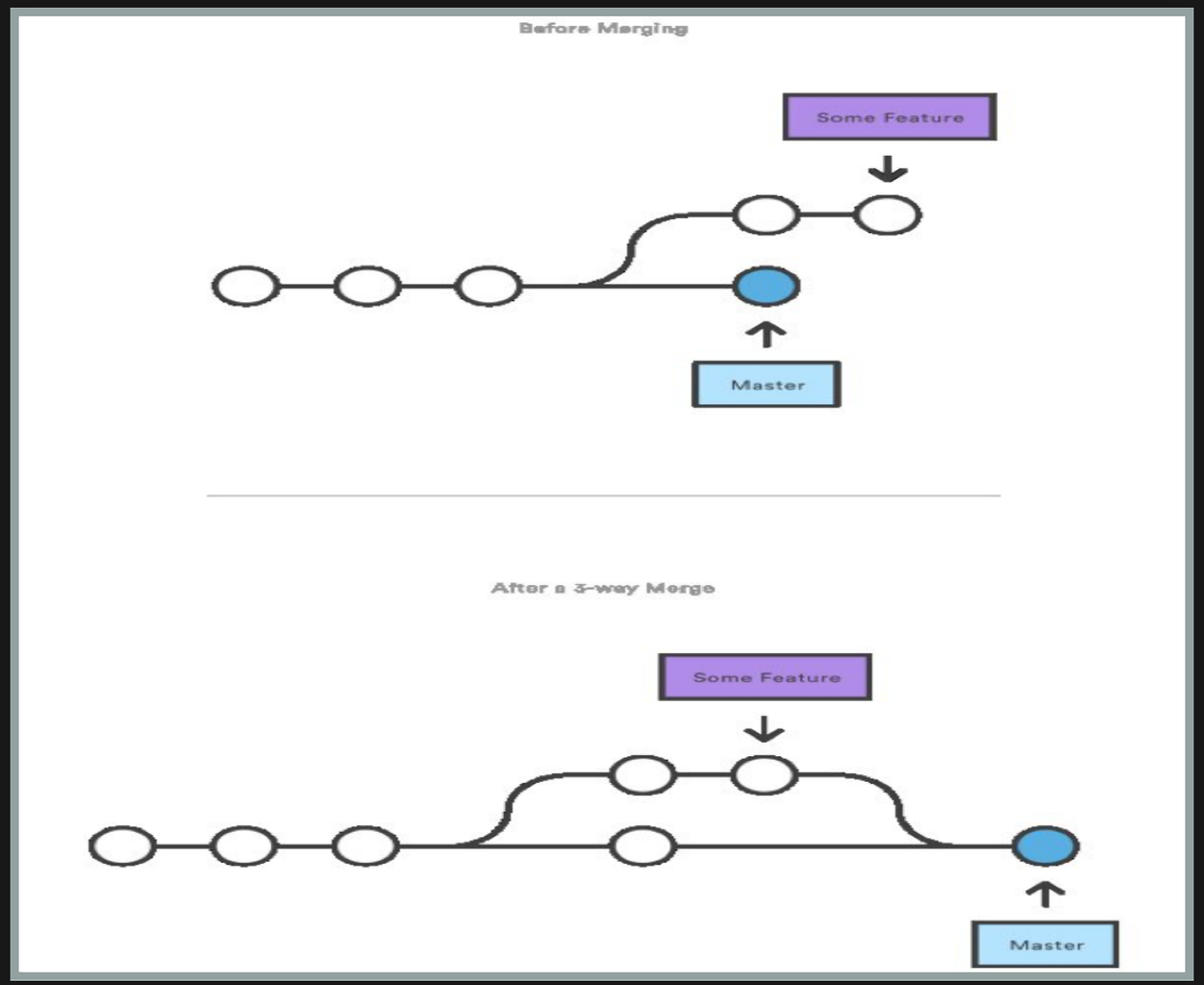


DevOps course by Ali Kahoot - Dice Analytics

MERGING - FAST FORWARD



MERGING - THREE WAY



MERGING - RESOLVING CONFLICT

Git automatically merge commits unless there are changes that conflict

```
On branch master
Unmerged paths:
(use "git add/rm ..." as appropriate to mark resolution)
both modified: hello.py
```

```
here is some content not affected by the conflict
<<<<<<< master
this is conflicted text from master
=======
this is conflicted text from feature branch
```

======= marker is the receiving branch and the part after is the merging branch

LAB - BRANCHING

- Create Branch
- List Branches
- Switch Branch
- Commit to new Branch
- Rename Branch
- Delete Branch

CREATE BRANCH

Create a new branch called test-branch by using the command:

```
$ git branch test-branch
```

LIST BRANCHES

LOCAL BRANCHES

List local branches by using the command:

```
$ git branch
or
$ git branch --list
```

REMOTE BRANCHES

List remote branches by using command:

```
$ git branch -r
```

LOCAL AND REMOTE BRANCHES

List local and remote branches by using the command:

```
$ git branch -a
```

SWITCH BRANCH

In order to commit to new branch, first you will have to switch to newly created branch i.e. testing, using the command:

```
$ git checkout test-branch
```

COMMIT TO NEW BRANCH

You have already switched to testing branch. Next create a file in new branch by using the command:

```
In Windows:
C:\> notepad test-branch.txt

In Linux:
$ vi test-branch.txt
```

Commit the file:

```
$ git add test-branch.txt
$ git commit -m "create test-branch.txt file"
```

RENAME BRANCH

Rename the branch testing to qa using the command:

```
$ git branch -m test-branch test-new-branch
```

DELETE BRANCH

In order to delete the branch first you will have to switch to another branch i.e. master in this case.

```
$ git checkout master
```

Delete the qa branch using the command:

```
$ git branch -d test-new-branch
```

LAB - MERGING

- Fast Forward Merge
- Three Way Merge
- Resolving Conflict

FAST FORWARD MERGE

You will first create a new branch and then add commit to it and then merge new branch to master branch.
First create a branch using the command:

```
$ git branch merge-ff
```

Switch to merge-ff branch:

```
$ git checkout merge-ff
```

Create a new file:

```
Windows:
C:\> notepad merge-ff.txt

Linux:
$ vi merge-ff.txt
```

Commit new file to merge-ff branch:

```
$ git add merge-ff.txt
$ git commit -m "create merge-ff.txt file"
```

Merge merge-ff branch to master:

```
$ git checkout master
$ git merge merge-ff
```

Check master logs for fast forward merge:

```
$ git log --oneline --graph --decorate
```

Delete mergeff branch:

```
$ git branch -d merge-ff
```

THREE WAY MERGE

You will first create a branch, then make a commit to it, then switch to master branch and make a commit to it so that it moves ahead from where the new branch was created, then merge new branch to master.

Create a new branch:

```
$ git checkout -b merge-tw master
```

Commit new file:

```
$ git add merge-tw
$ git commit -m "create merge-tw file"
```

Move to master branch:

```
$ git checkout master
```

Create a new file on master branch:

```
Windows:
C:\> notepad master-merge-ex.txt

Linux:
$ vi master-merge-ex.txt
```

Commit new file:

```
$ git add master-merge-ex.txt
$ git commit -m "create master-merge-ex file"
```

Merge merge-tw branch to master:

```
$ git merge merge-tw
```

Check master logs for three way merge:

```
$ git log --oneline --graph --decorate
```

Delete mergetw branch:

```
$ git branch -d merge-tw
```

RESOLVING CONFLICT

You will first create a branch and then edit an already existing file, next you will switch to master branch and edit the same file in a way that both changes conflict with each other.

Create a new branch:

```
$ git checkout -b conflict master
```

Edit an existing file newfilemaster.txt on conflict branch and append Hello World:

```
Windows:
C:\> notepad master-merge-ex.txt

Linux:
$ vi master-merge-ex.txt
```

Commit changes:

```
$ git add master-merge-ex.txt
$ git commit -m "edit master-merge-ex file"
```

Move to master branch:

```
$ git checkout master
```

Edit an existing file newfilemaster.txt on master branch and append Hello:

```
Windows:
C:\> notepad master-merge-ex.txt

Linux:
$ vi master-merge-ex.txt
```

Commit new file:

```
$ git add master-merge-ex.txt
$ git commit -m "edit master-merge-ex file on master branch"
```

Merge conflict branch to master:

```
$ git merge conflict
```

You will encounter a merge conflict, run the git status command to verify that:

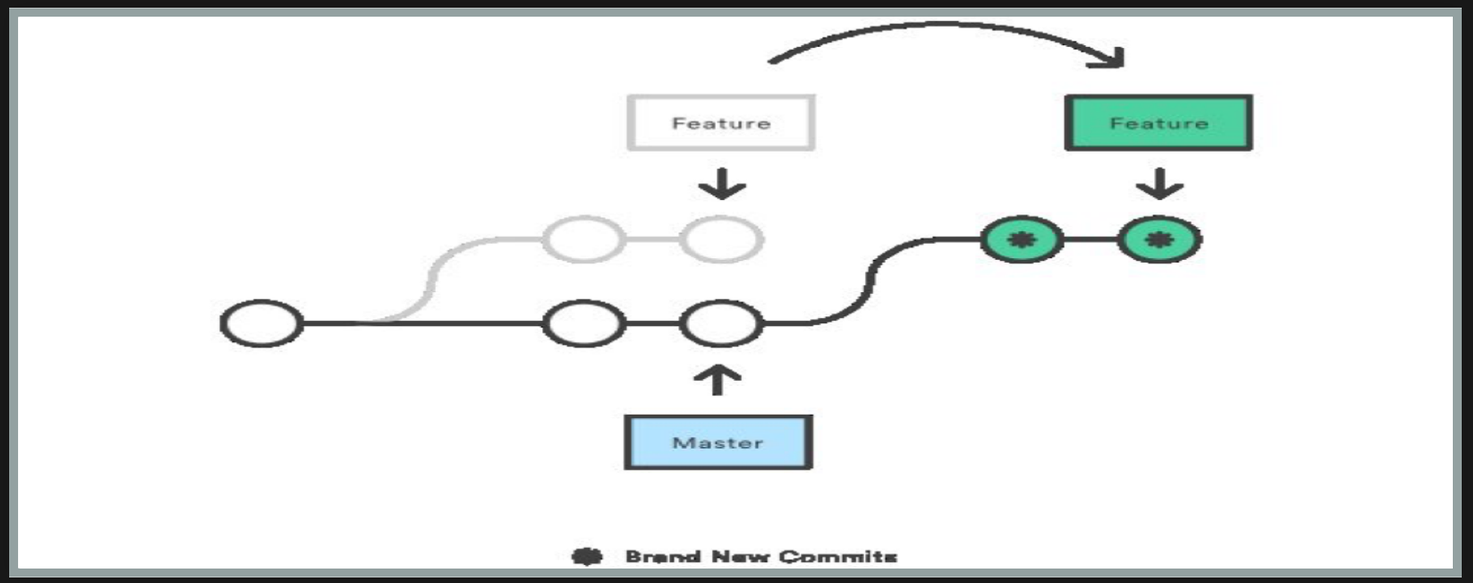
```
$ git status
```

Open the file with the conflict i.e. newfilemater and resolve the conflict. Once conflicts are resolved run the following commands:

```
$ git add master-merge-ex.txt
$ git commit -m "resolve merge conflict"
```

REBASING

- Alternative to integrate changes from one branch to another
- Changes the base of your branch from one commit to another
- Integrates upstream changes into your branch
- Linear history
- Command: git rebase



REBASING VS MERGE

STASHING

- Why do we need Stashing?
- Saves you uncommitted code for later use
- Reverts changes from your working copy
- You can make changes, create new commits, switch branches, and perform any other Git operations and then re-apply your stash when you're ready
- Local to your Git Repo
- Commands:
  - List Stashes: git stash list
  - Stash Changes: git stash
  - Apply Stash: git stash apply or git stash pop
  - Clear all Stashes: git stash clear
  - Stash with description: git stash save "message"

TAGGING

- Allows you to give commit a name
- Mark important checkpoint in the project
- Tags
  - Annotated: extra metadata e.g. tagger name, email, and date.
  - Lightweight: only tag name
- Branch that doesnâ□□t change
  - You can checkout to them
- Commands
  - List tags: git tag -l
  - Lightweight tag: git tag
  - Annotated tag: git tag -a -m
  - Tag Details: git show
  - Delete tag: git tag --delete

LAB - REBASING

REBASE FROM MASTER BRANCH

You will first create a new branch, next add a commit to it and then switch to master branch, next edit an existing file on master branch, then switch back to new branch and then rebase. This will move the base of your branch from earlier commit to current master branch commit

First create a branch using the command:

```
$ git checkout -b rebase-test master
```

Create a new file:

```
Windows:
C:\> notepad rebase-file.txt

Linux:
$ vi rebase-file.txt
```

Commit new file to rebase-test branch:

```
$ git add rebase-file.txt
$ git commit -m "create rebase-file in rebase-test branch"
```

Switch to master branch:

```
$ git checkout master
```

Edit an existing file master-merge-ex.txt:

```
Windows:
C:\> notepad master-merge-ex.txt

Linux:
$ vi master-merge-ex.txt
```

Commit new file:

```
$ git add master-merge-ex.txt
$ git commit -m "edit master-merge-ex file"
```

Switch to rebase-test branch:

```
$ git checkout rebase-test
```

Rebase using the following command:

```
$ git rebase master
```

Output the contents of newfilemaster.txt, you will see the changes made on the master branch

Switch to master branch and delete rebase-test branch:

```
$ git checkout master
$ git branch -d rebase-test
```

LAB - STASHING

- Stash Changes
- Apply Stashed changes

STASH CHANGES

You will create a branch, then make some changes to an existing file and then stash those

Create a branch:

```
$ git branch stash-example
$ git checkout stash-example
```

Edit an existing file newfilemaster.txt:

```
Windows:
C:\> notepad master-merge-ex.txt

Linux:
$ vi master-merge-ex.txt
```

Check state of working directory:

```
$ git status
```

Stash your changes:

```
$ git stash
```

To see which stashes youâ□□ve stored, you can use:

```
$ git stash list
```

Check state of working directory:

```
$ git status
```

After stashing the changes your working directory will be clean

APPLY STASHED CHANGES

You will reapply your stashed changes

Check state of working directory:

```
$ git status
```

Reapply your stashed changes:

```
$ git stash pop
```

Check state of working directory:

```
$ git status
```

You will see the changes you stashed earlier

Commit changes:

```
$ git add master-merge-ex.txt
$ git commit -m â??edit master-merge-ex file for stash example"
```

Delete the branch

```
$ git checkout master
$ git branch -d stash-example
```

LAB - TAGGING

- Create Lightweight Tag
- Create Annotated Tag
- List Tags
- Tag Old Commit
- Get Information On a Tag
- Delete Tags

CREATING LIGHTWEIGHT TAG

Create tag on latest commit:

```
$ git tag "first-tag"
```

CREATE ANNOTATED TAG

Create annotated tag with message on latest commit:

```
$ git tag -a "second-tag" -m "tag-test"
```

LIST TAGS

List tags using commands:

```
$ git tag -l
```

You will see the list of tags i.g. first-tag and second-tag

TAG OLD COMMIT

List old commit:

```
$ git log -n3
```

This will show list of last three commits

Copy the hash of second last commit and tag it using the command: $ git tag "third-tag"

GET INFORMATION ON A TAG

Get information on a lightweight tag:

```
$ git show first-tag
```

Get information on an annotated tag:

```
$ git show second-tag
```

DELETE TAG

Delete all the created tags:

```
$ git tag --delete first-tag
$ git tag --delete second-tag
$ git tag --delete third-tag
```

PULL REQUEST

- Method of submitting contribution
- Makes collaboration easier
- Workflow
  - Create Branch
  - Make Changes
  - Create Pull Request
  - Merge Once Approved

LAB - PULL REQUEST

You will first create a new branch, then add a commit to it, then create a pull request to master, once approved you will merge the changes to master branch

Create a new branch:

```
$ git checkout -b pr-example master
```

Create a new file:

```
Windows:
C:\> notepad pr-example.txt

Linux:
$ vi pr-example.txt
```

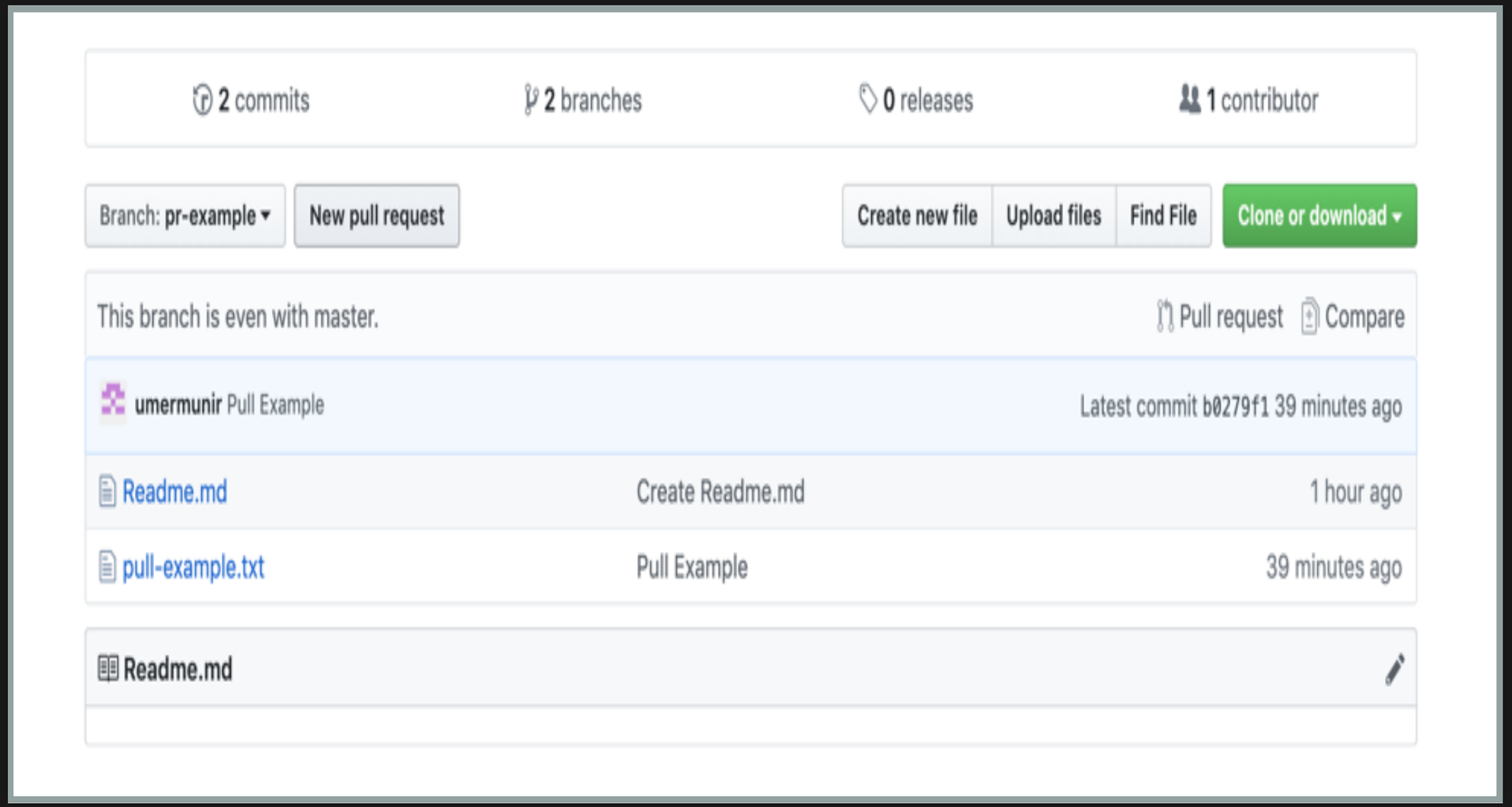
Commit new file:

```
$ git add pr-example.txt
$ git commit -m "add new file"
```

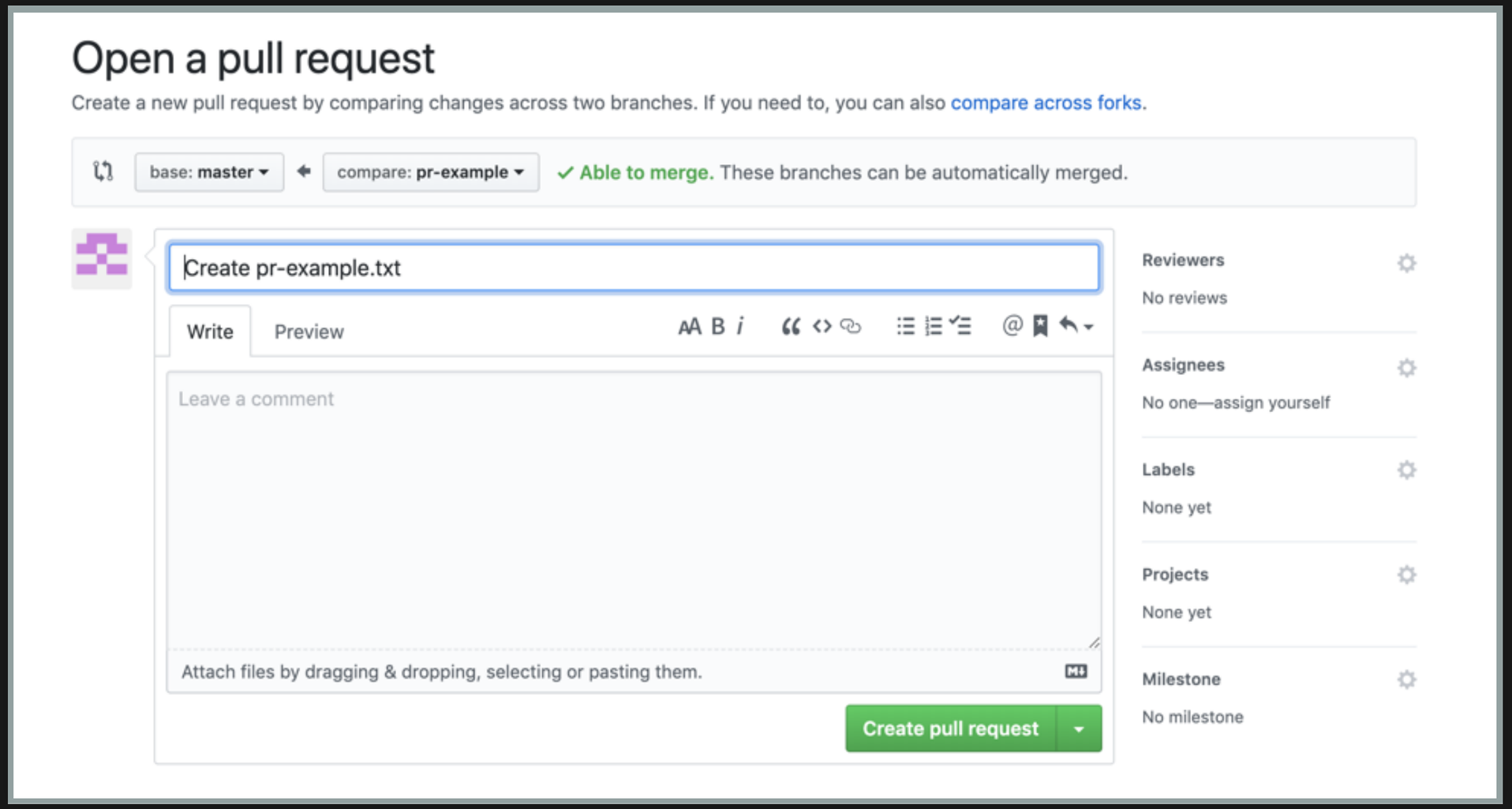
Push changes to the remote repository:

```
$ git push origin pr-example
```

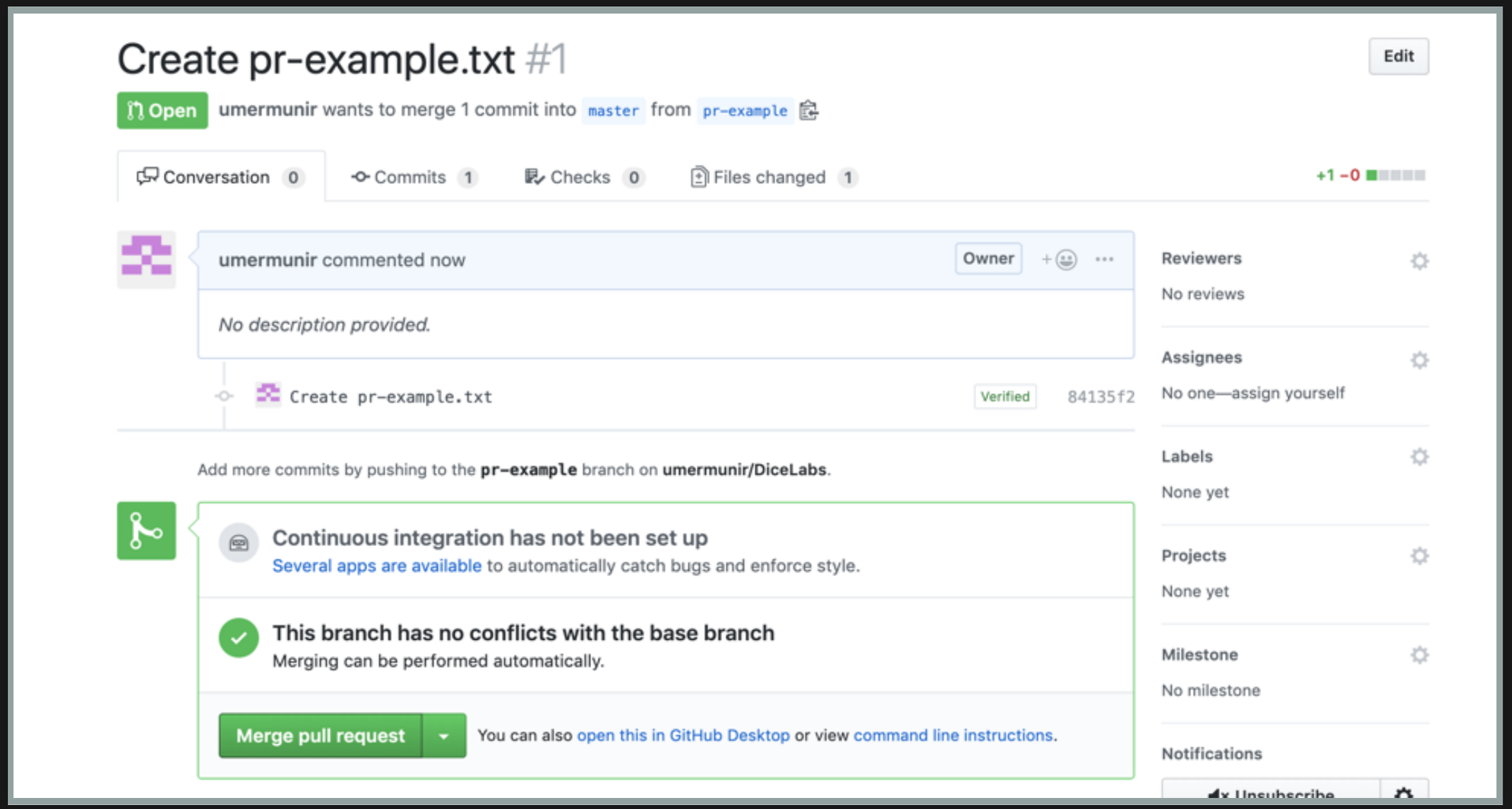
You will create a Pull Request from pr-example to master branch. Create a pull request for Github repository page and click on "Pull Request" button in the repo header



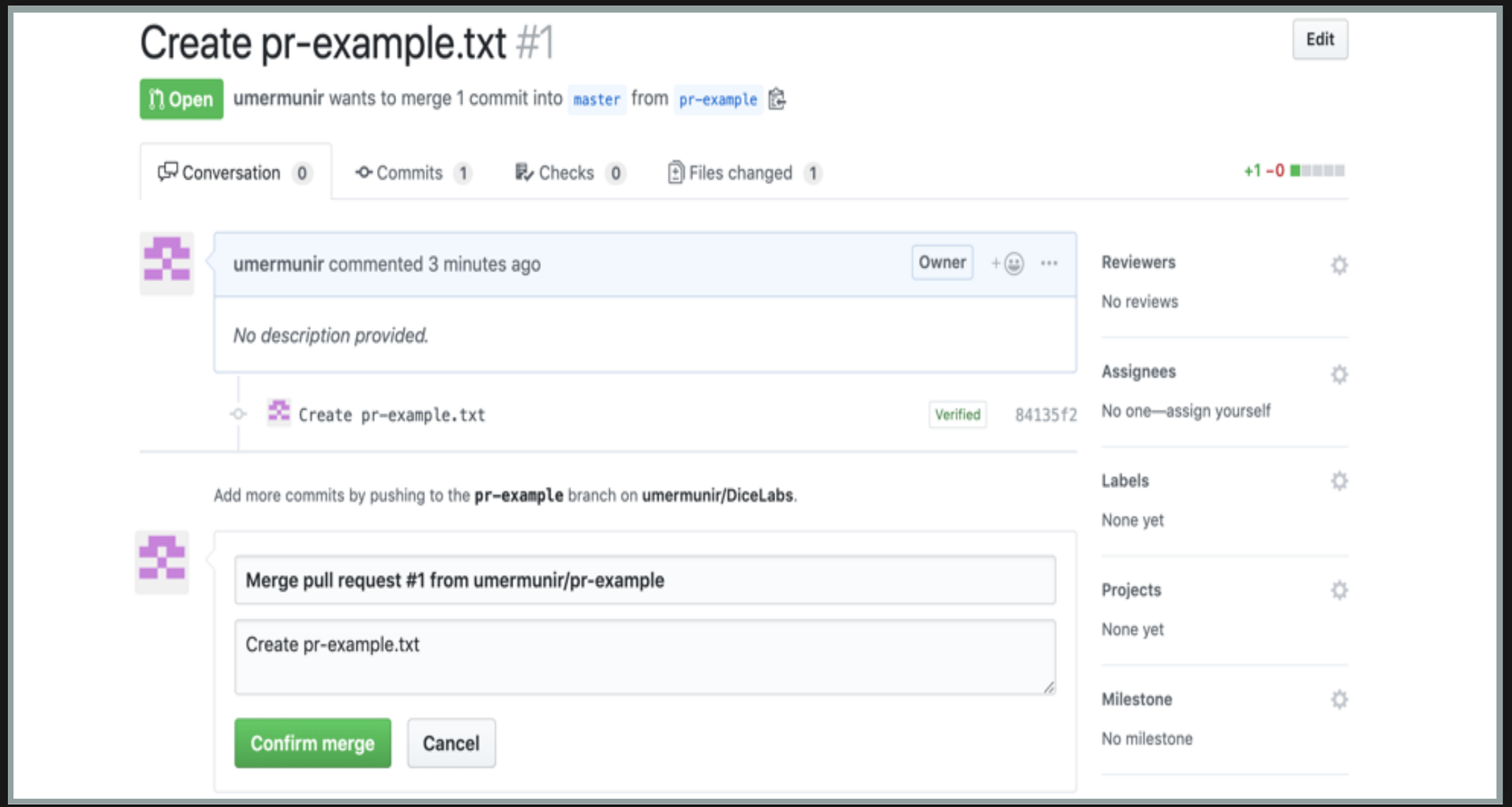
Select master branch in base and pr-example branch in compare and click Create pull request



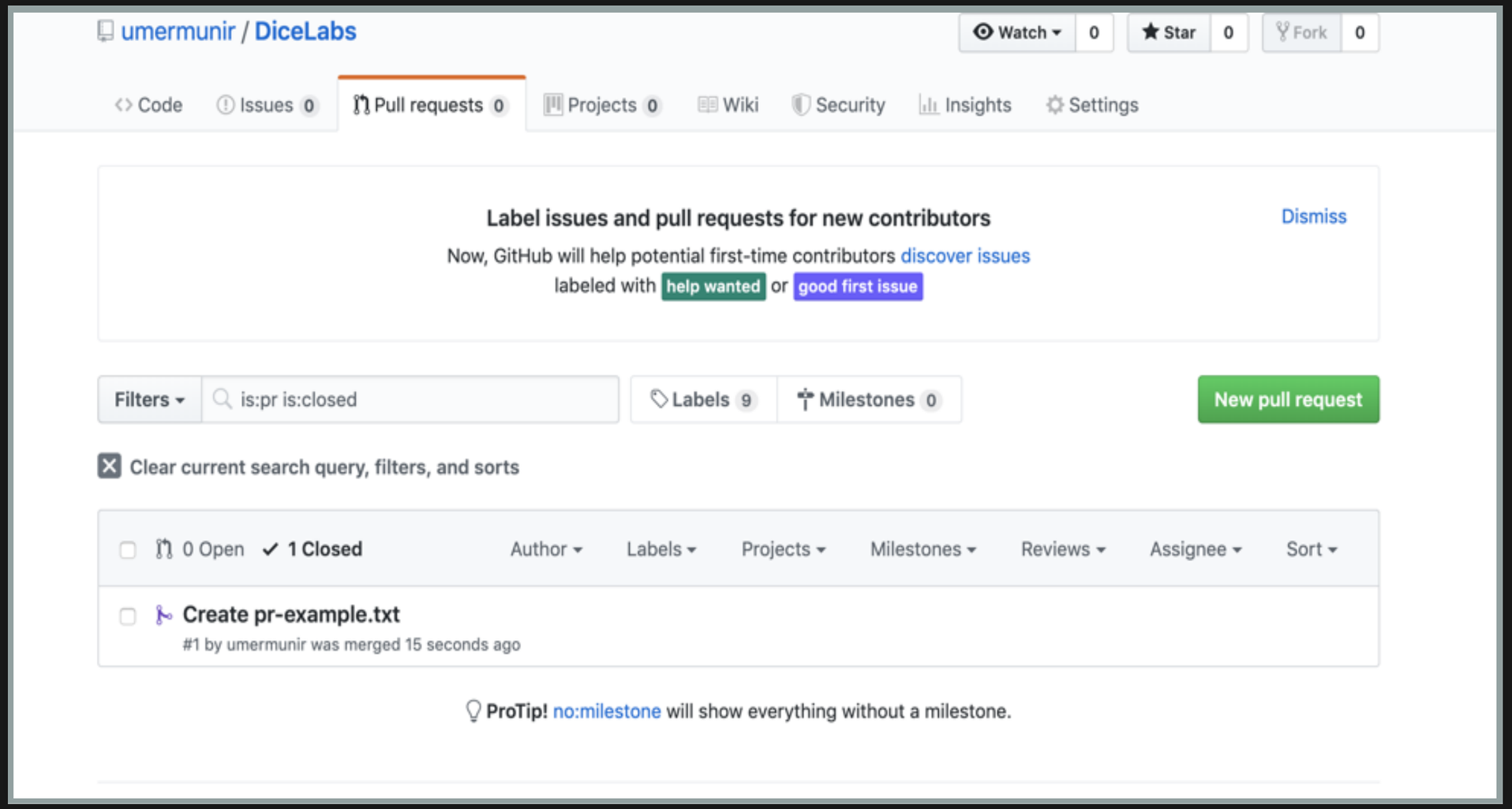
Since we have not specified any reviewers and your the administrator of the repository so you can merge the code with approval. If you were a contributor and not the administrator then you would not be able to merge with approval. Click Merge pull request



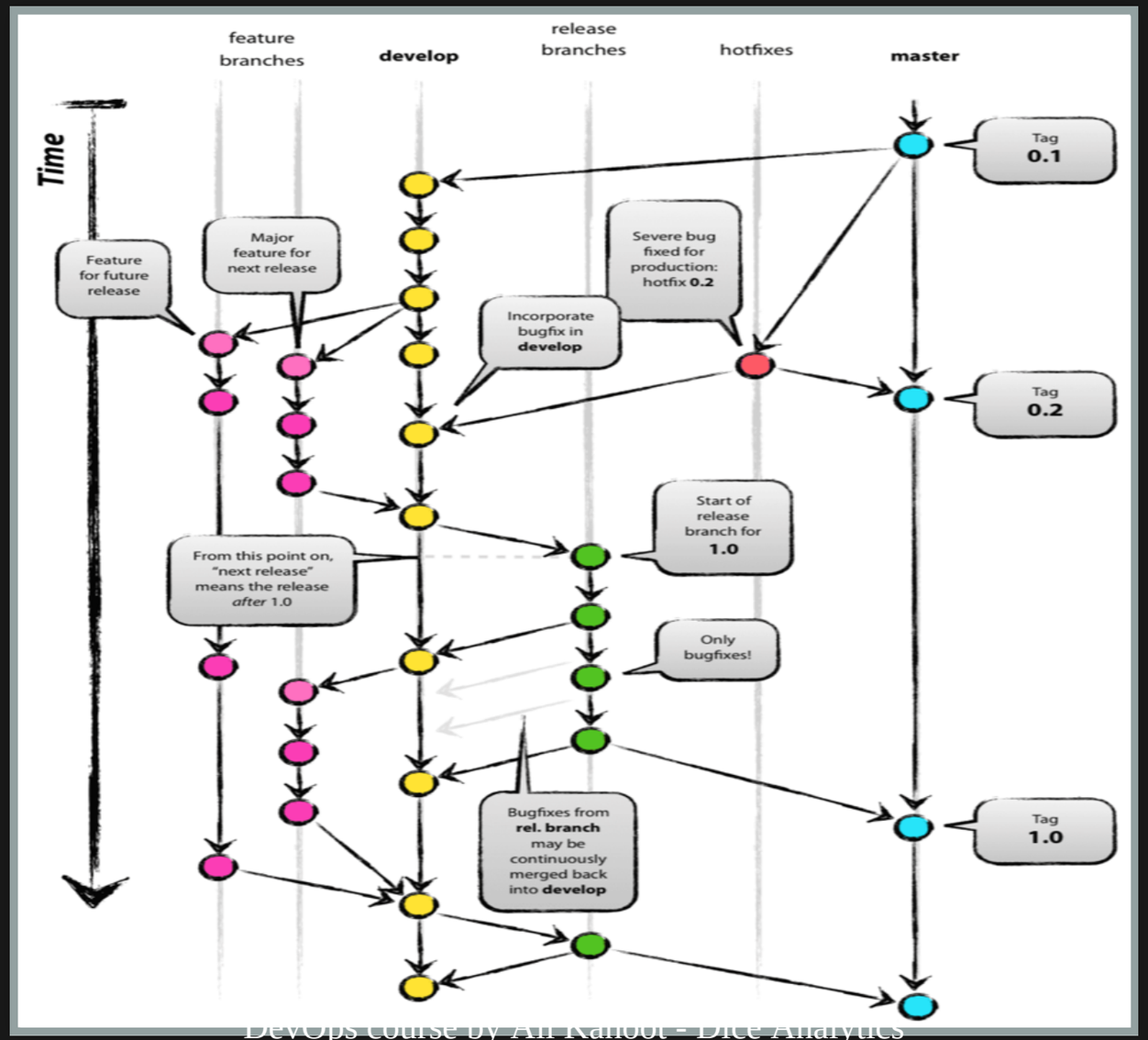
Add comments and click Confirm merge



You have successfully created and closed a pull request



BRANCHING MODEL



FORKING

- Copy of a repository
- Server-side copy as compared to clone
- Allows to freely experiment without affecting the original project
- Use Cases
  - Propose changes to someone else's project
  - Contribute to open source projects

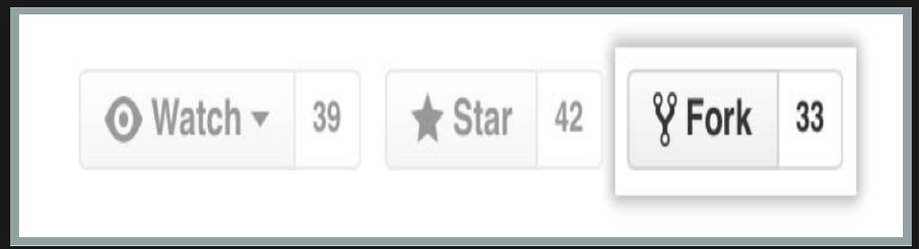
CONTRIBUTE TO OPEN SOURCE PROJECTS

- Fork the Project
- Clone your copy of Project
- Create an upstream remote and sync local copy
- Do the work
- Push to your remote repository
- Create a new Pull Request in GitHub
- Review by Maintainers
- Respond to any code review feedback

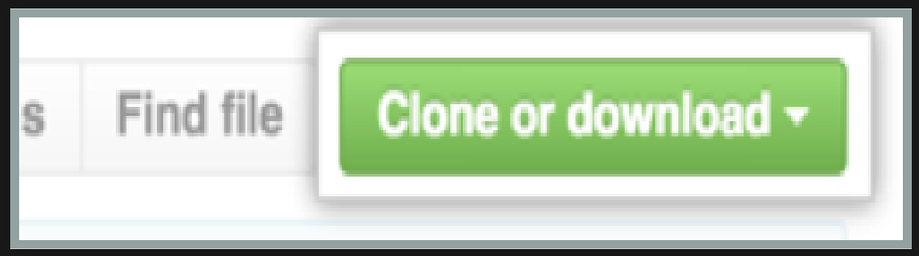
LAB - FORKING

You will fork a repository from GitHub this will create your server-side copy of repository, then you will clone repository, then make changes and push them to your copy of remote repository

On GitHub, navigate to the DiceLab repository. In the top-right corner of the page, click Fork



On GitHub, navigate to your fork copy. Under repository name click Clone or download



In the Clone with HTTPs section, click to copy the clone URL for the repository.

Clone your copy of repository using the command

```
$ git clone https://github.com/YOUR-USERNAME/DiceLab.git
```

Create a new file

```
Windows
C:\> notepad fork-example.txt

Linux
$ vi fork-example.txt
```

Commit new file

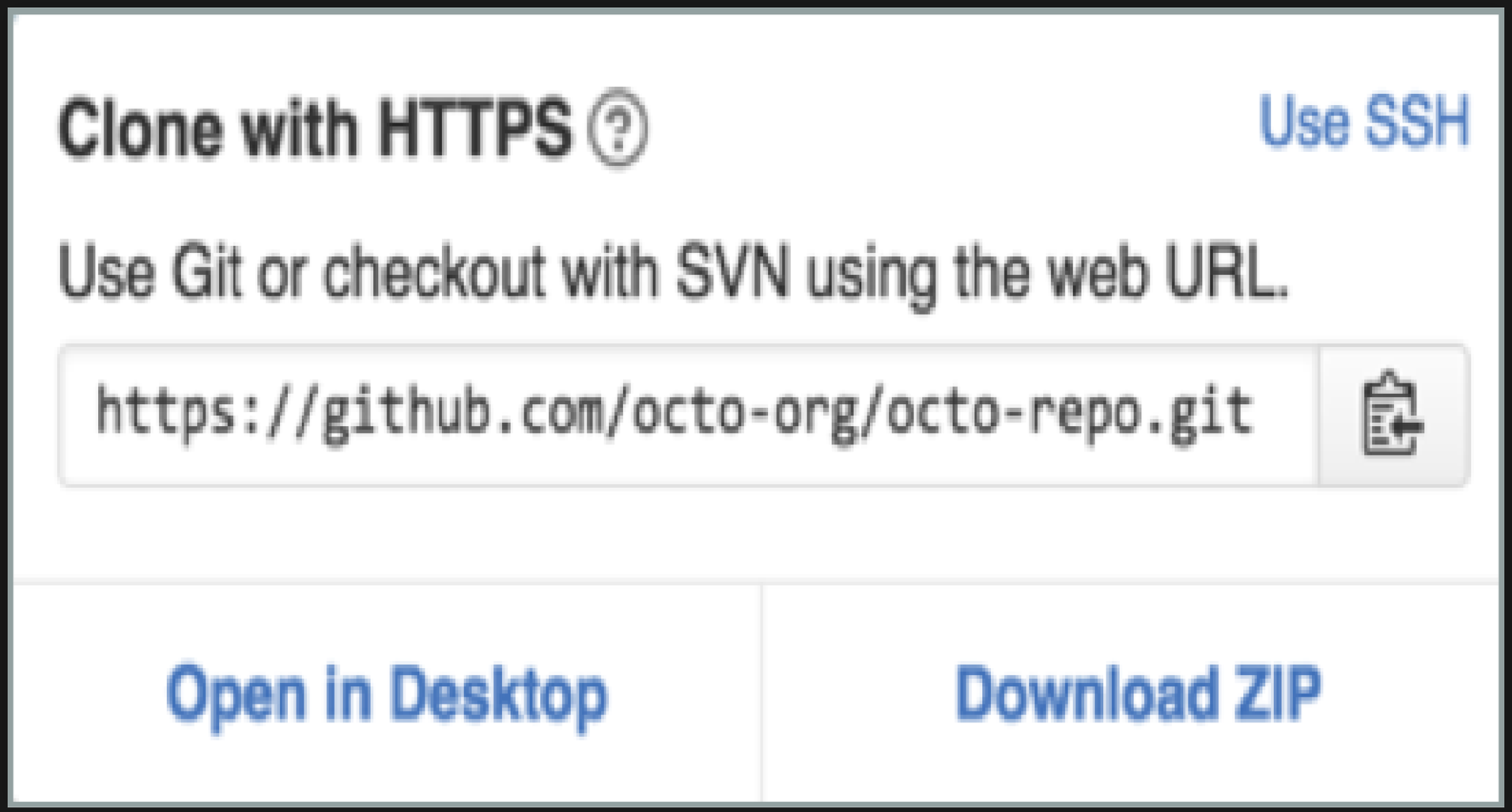
```
$ git add fork-example.txt
$ git commit -m "create file for fork example"
```

Push changes to your copy of repository

```
$ git push origin master
```

CREATE A PULL REQUEST

Create a Pull Request in my Repo using your branch.



BEST PRACTICES

- Don't commit directly to master
- Create .gitignore file for your projects
- Don't store credentials as code/config in GitHub
- Write meaningful commit message
- Test Your Code Before You Commit
- Use Branches
- Always pull the latest updates
- Protect your project
- The more approvals, the better
- Rebase your branches periodically
- Agree on workflow

GIT PLAYGROUND

- http://git-school.github.io/visualizing-git/
- https://learngitbranching.js.org/?NODEMO
- https://learngitbranching.js.org/