Course: Artificial Intelligence and Machine Learning

Code: 20CS51I

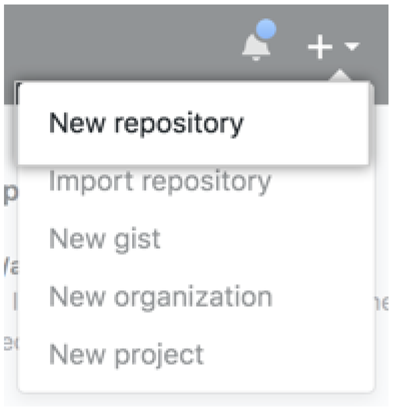
WEEK1- Artificial intelligence concepts

Session 7:

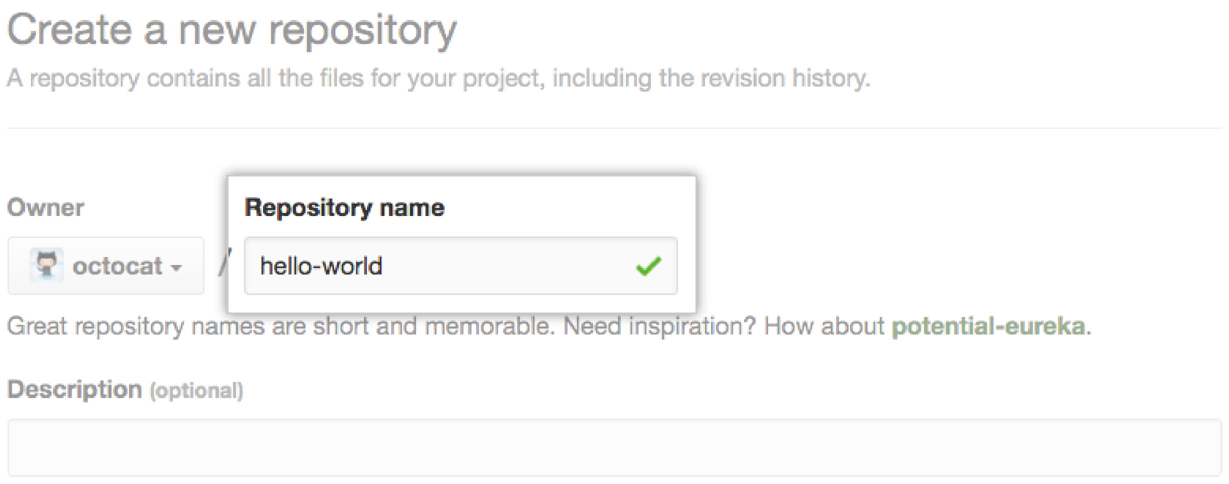
Practical session

CREATING A REPOSITORY

In the upper-right corner of any page, use the drop-down menu, and select **New repository**.



Type a short, memorable name for your repository. For example, "hello-world".



Create a new repository

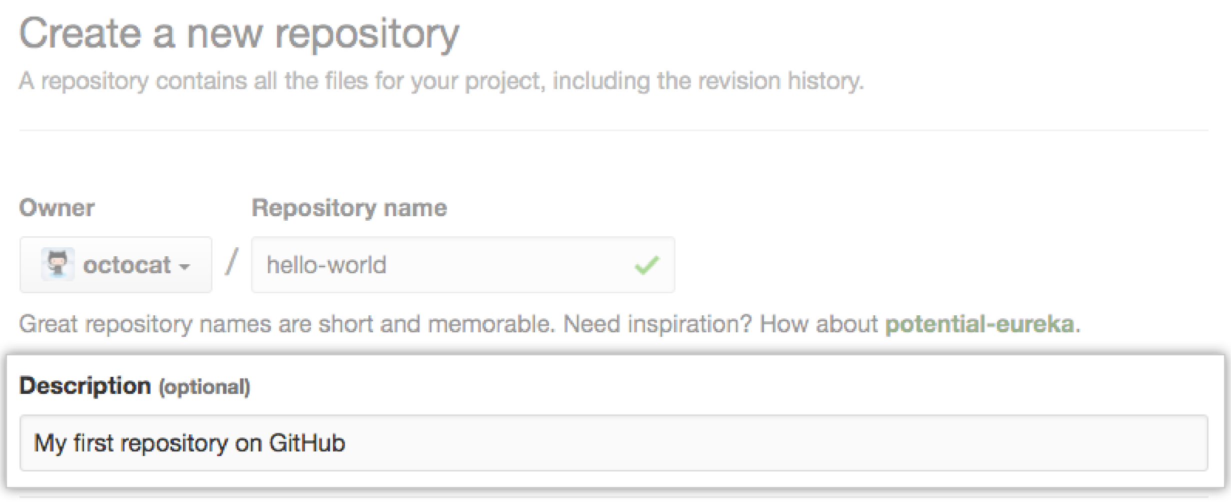
A repository contains all the files for your project, including the revision history.

Owner: octocat / Repository name: hello-world

Great repository names are short and memorable. Need inspiration? How about potential-eureka.

Description (optional)

Optionally, add a description of your repository. For example, "My first repository on GitHub."


Create a new repository

A repository contains all the files for your project, including the revision history.

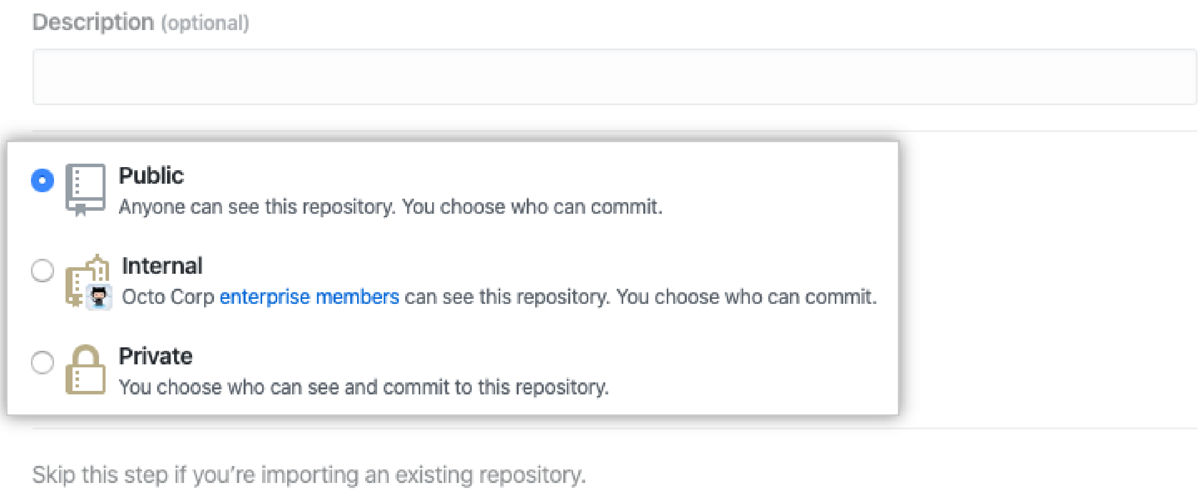
Owner: octocat / Repository name: hello-world

Great repository names are short and memorable. Need inspiration? How about **potential-eureka**.

Description (optional)

My first repository on GitHub

Choose a repository visibility. For more information, see "About repositories."


Description (optional)

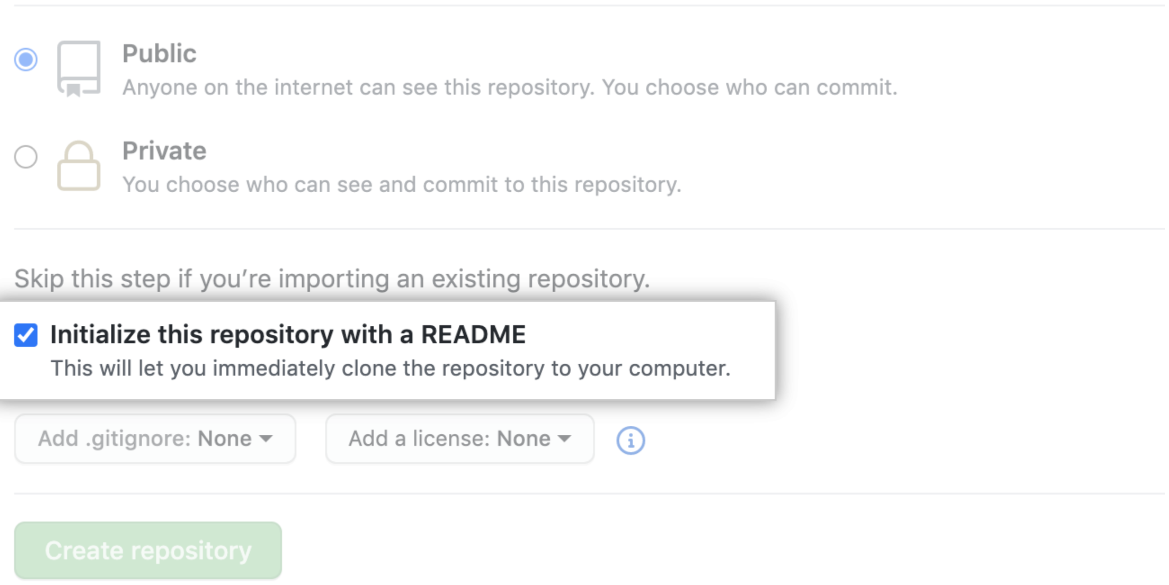
[✓ **Public**] Anyone can see this repository. You choose who can commit.

Internal Octo Corp enterprise members can see this repository. You choose who can commit.

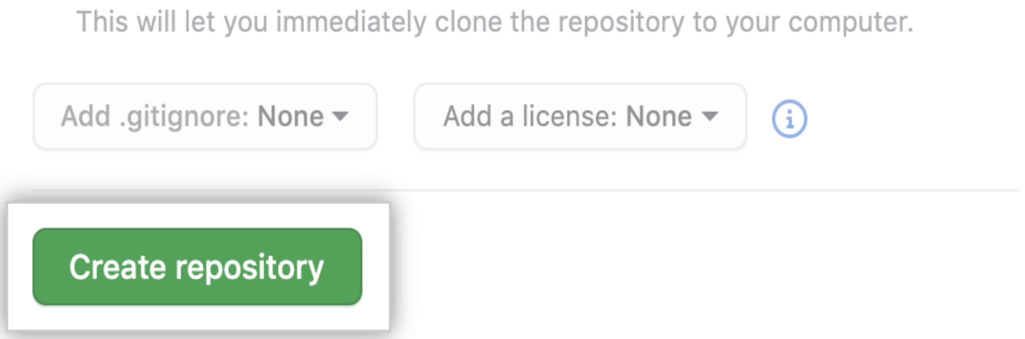
Private You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

Select **Initialize this repository with a README**.


Public
Anyone on the internet 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

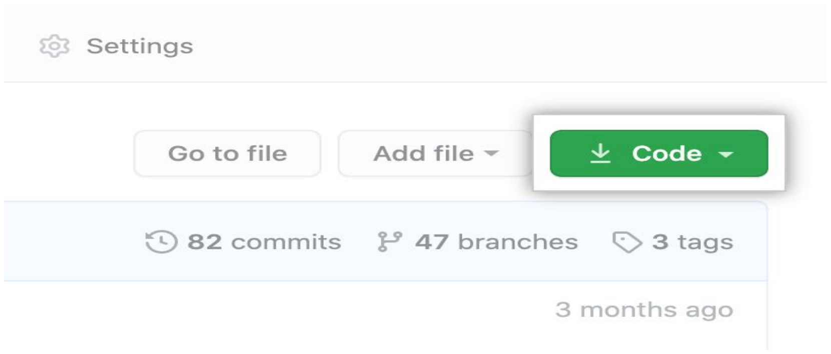
Click **Create repository**.


This will let you immediately clone the repository to your computer.
Add .gitignore: None
Add a license: None
Create repository

CLONING A REPOSITORY

On GitHub.com, navigate to the main page of the repository.

Above the list of files, click **Code**.

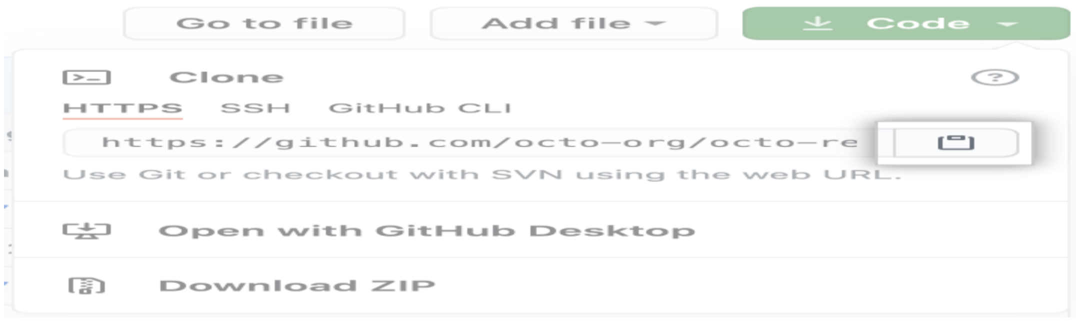


Copy the URL for the repository.

To clone the repository using HTTPS, under "HTTPS", click .

To clone the repository using an SSH key, including a certificate issued by your organization's SSH certificate authority, click **SSH**, then click .

To clone a repository using GitHub CLI, click **GitHub CLI**, then click .


Go to file
Add file
Code
Clone
HTTPS
SSH
GitHub CLI
https://github.com/octo-org/octo-re
Use Git or checkout with SVN using the web URL.
Open with GitHub Desktop
Download ZIP

Open Git Bash.

Change the current working directory to the location where you want the cloned directory.

Type git clone, and then paste the URL you copied earlier.

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

Press **Enter** to create your local clone.

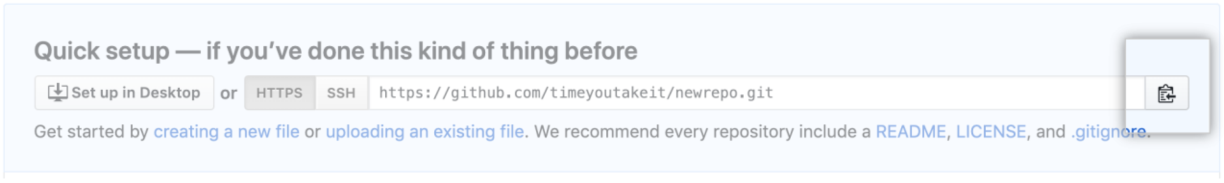
```
$ git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY
> Cloning into `Spoon-Knife`...
> remote: Counting objects: 10, done.
> remote: Compressing objects: 100% (8/8), done.
> remove: Total 10 (delta 1), reused 10 (delta 1)
> Unpacking objects: 100% (10/10), done.
```

Cloning an empty repository

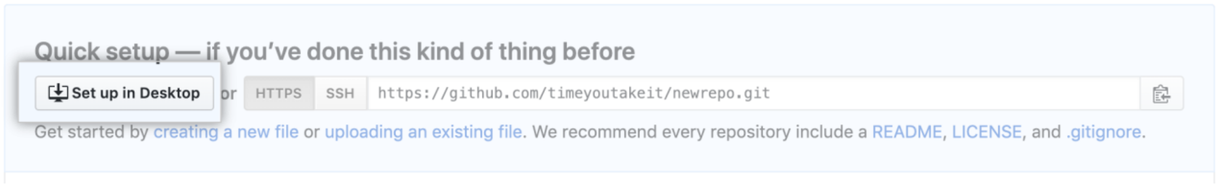
An empty repository contains no files. It's often made if you don't initialize the repository with a README when creating it.

On GitHub.com, navigate to the main page of the repository.

To clone your repository using the command line using HTTPS, under "Quick setup", click . To clone the repository using an SSH key, including a certificate issued by your organization's SSH certificate authority, click **SSH**, then click .


Quick setup — if you've done this kind of thing before
Set up in Desktop
or
HTTPS
SSH
https://github.com/timeyoutakeit/newrepo.git
Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

Alternatively, to clone your repository in Desktop, click **Set up in Desktop** and follow the prompts to complete the clone.


Quick setup — if you've done this kind of thing before
Set up in Desktop
or
HTTPS
SSH
https://github.com/timeyoutakeit/newrepo.git
Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

Open Git Bash.

Change the current working directory to the location where you want the cloned directory.

Type git clone, and then paste the URL you copied earlier.

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

Press **Enter** to create your local clone.

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

> Cloning into `Spoon-Knife`...

> remote: Counting objects: 10, done.

> remote: Compressing objects: 100% (8/8), done.

> remove: Total 10 (delta 1), reused 10 (delta 1)

> Unpacking objects: 100% (10/10), done.
```

MAKING AND RECORDING CHANGES IN GIT:

Add files to a Git repository

To tell Git about the file, we will use the `git add` command:

```
$ git add paper.md

$ git status

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

        new file:   paper.md
```

Now our file is listed underneath where it says **Changes to be committed**.

Commit changes

In order to tell Git to record our change, our new file, into the repository, we need to **commit** it:

```
$ git commit
```

# Type a commit message: "Add title and authors"

# Save the commit message and close your text editor (nano, notepad etc.)

Our default editor will now pop up. Why? Well, Git can automatically figure out that directories and files are committed, and by whom (thanks to the information we provided before) and even, what changes were made, but it cannot figure out why. So we need to provide this in a commit message.

If we save our commit message **and exit the editor**, Git will now commit our file.

```
[master (root-commit) 21cfbde]

1 file changed, 2 insertions(+) Add title and authors

create mode 100644 paper.md
```

This output shows the number of files changed and the number of lines inserted or deleted across all those files. Here, we have changed (by adding) 1 file and inserted 2 lines.

Now, if we look at its status,

```
$ git status
On branch master
nothing to commit, working directory clean
```

our file is now in the repository. The output from the git status command means that we have a clean directory i.e. no tracked but modified files.

STAGING AND COMMITTING GIT:

Staged means that you have marked a modified file in its current version to go into your next commit snapshot. Committed means that the data is safely stored in your local database.

Stage Files to Prepare for Commit

1. Enter one of the following commands, depending on what you want to do:

Stage all files: **git add .**

Stage a file: **git add example.html** (replace example.html with your file name)

Stage a folder: **git add myfolder** (replace myfolder with your folder path)

Keep in Mind:

If the file name/path has a space, wrap it in quotes.

You can repeat the above commands for different files and folders.

2. Check the status again by entering the following command:

git status

3. You should see there are changes ready to be committed.

Unstage a File

If you accidental stage something, use the following command to unstage it:

git reset HEAD example.html
(replace example.html with your file or folder)

Deleting Files

If you delete files they will appear in git status as deleted, and you must use **git add** to stage them. Another way to do this is using **git rm** command, which both deletes a file and stages it all with one command:

git rm example.html to remove a file (and stage it)

git rm -r myfolder to remove a folder (and stage it)

Commit Files

1. Enter this command:

git commit -m "Message that describes what this change does"

2. Check the status again by running this command:

git status

3. If all changes have been committed, and there are no untracked files, it should say: **nothing to commit, working tree clean**.

VIEWING THE HISTORY OF AL THE CHANGES IN GIT:

git log` command is used to view the commit history and display the necessary information of the git repository. This command displays the latest git commits information in chronological order, and the last commit will be displayed first.

UNDOING THE CHANGES CHANGES IN GIT:

If you have committed changes to a file (i.e. you have run both git add and git commit), and want to undo those changes, then you can use **git reset HEAD~** to undo your commit.

GIT BRANCHING AND MERGING:

Independent line of development or parallel development of code along with the main code.

Branches are used to develop a new feature or to fix a bug in the code.

In other words:

Branch is a reference to a commit.

<u>cd learn_branching</u>

#To visualize the graphical logs for current branch

```
git log --oneline --graph
* dff8df9 (HEAD -> master, origin/master) 2 commit: hello.sh code file added.
* 4871096 1 commit: Initial Project structure and Readme file added.
```

#To visualize the graphical logs for all the branches

```
git log --graph --oneline --all --decorate
```

--decorate: adds labels to the commits (HEAD, tags, remote branches and local branches)

Set an alias for this command in the config file

```
git config --global alias.showbranches 'log --graph --oneline --all --decorate'
git showbranches
```

CREATE A BRANCH

Project: learn_branching

--git list branches using the command: git branch

```
Divya1@Divya:learn_branching [master] $git branch
 iss53
* master
 new-feature
 newBranch
```

Create a new branch from the latest commit

--create branch git

```
git branch newBranch
```

Create a branch from an old commit

```
git branch firstBranch commitID
```

Create a branch and jump onto it

git checkout -b quickfix

SWITCH BETWEEN BRANCHES

Switch/jump to the new branch

git checkout newBranch

Switch back to master branch

git checkout master

MERGING LOCAL BRANCHES TOGETHER

`git merge`

We have two branches and `make_function` has the new feature, which makes `wordcount` a function. That's a better way to write scripts. Since the `make_function` branch contains the feature that we want and we think that it's ready to go, we can bring the contents of the `make_function` branch into the `master` branch by using `git merge`.

To do a merge (locally), `git checkout` the branch you want to merge *INTO*. Then type `git merge <branch>` where `<branch>` is the branch you want to merge *FROM*.

We are on the `master` branch and want to merge in `make_function` so we do:

```
$ git merge make_function
Updating 3f62d8f..1071b15
Fast-forward
 word_count.py | 23 ++++++++++++++++-------
 1 file changed, 16 insertions(+), 7 deletions(-)
```

Because the history of `master` and the history of `make_function` share common ancestors and don't have any divergence descendents, we get a "fast-forward" merge. That means that all of the changes we made in `make_function` look as if they were made directly in `master`.

Sometimes, you might want to force `git` to create a merge commit, so that you know where a branch was merged in. In that case, you can prevent `git` from doing a "fast-forward" by merging with the `-no-ff` flag.