

Hands-on Lab: Get familiar with Git Commands



Estimated time needed: 30 mins

Objectives

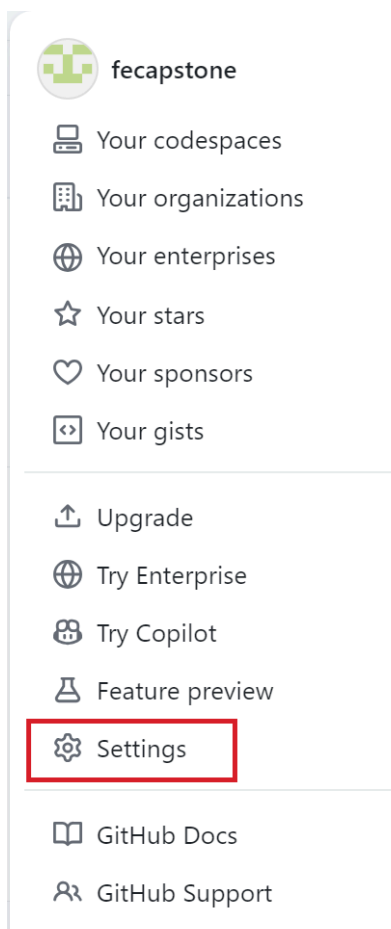
After completing this lab, you will be able to:

1. Create a personal access token
2. Fork existing repository using the UI
3. Clone forked repository in the lab environment
4. Add and commit to the local branch
5. Push changes to the forked repository

Note The images incorporated in these instructions are exclusively for illustrative purposes.


Exercise 1: Create PAT to authenticate the `git commit` and `git push` commands”

1. To begin, go to your GitHub account and click your profile icon located in the top-right corner. Then, click **Settings**.




2. Next, select **Developer settings**. This option is typically available towards the bottom of the window.


Security

 Code security and analysis

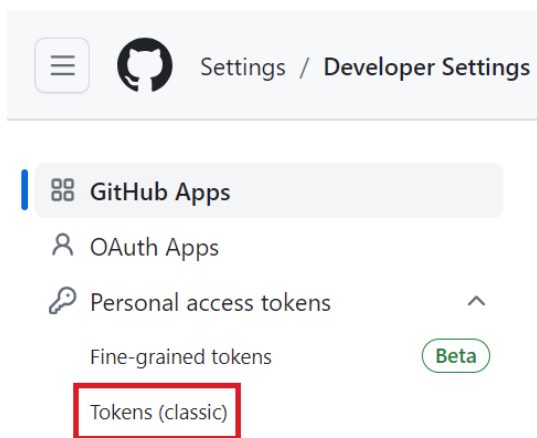
Integrations

 Applications Scheduled reminders

Archives

 Security log Sponsorship log **Developer settings**

3. Navigate to **Tokens (classic)** under **Personal access tokens**.



4. To generate an access token, click **Generate a personal access token**.

Personal access tokens (classic)

[Generate new token ▼](#)

Need an API token for scripts or testing? [Generate a personal access token](#) for quick access to the [GitHub API](#).

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

5. In the **Generate token** page, fill in the required details and click the **repo** checkbox to enable access for `git` commands.

New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

fecapstone

What's this token for?

Expiration *

30 days

The token will expire on Tue, Sep 5 2023

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

- | | |
|---|--------------------------------------|
| <input checked="" type="checkbox"/> repo | Full control of private repositories |
| <input checked="" type="checkbox"/> repo:status | Access commit status |
| <input checked="" type="checkbox"/> repo_deployment | Access deployment status |
| <input checked="" type="checkbox"/> public_repo | Access public repositories |
| <input checked="" type="checkbox"/> repo:invite | Access repository invitations |
| <input checked="" type="checkbox"/> security_events | Read and write security events |

6. Then, click **Generate token**.

- | | |
|---|--|
| <input type="checkbox"/> admin:ssh_signing_key | Full control of public user SSH signing keys |
| <input type="checkbox"/> write:ssh_signing_key | Write public user SSH signing keys |
| <input type="checkbox"/> read:ssh_signing_key | Read public user SSH signing keys |

Generate token

Cancel


7. Your personal access token will be generated. The token is only valid for **30 days**. You will need to generate a new token once the current token expires.

REMEMBER: Make sure to copy your personal access token now. You won't be able to see it again!

Exercise 2: Fork the repository

To fork a source repository, complete the following steps:

1. Log in to GitHub and go to this project's [sample source repository](#). This is the upstream repository for your project.
2. At the top right of the screen, click Fork and select your own GitHub account as the destination for the fork.


 **ibm-developer-skills-network / gkpbt-css-circle** Public







generated from [ibm-developer-skills-network/coding-project-template](#)


Unwatch 2 Fork

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

main 1 branch 0 tags Go to file Add file Code

 **upkarliddar** Automatically close PRs 729ceb2 2 days ago 5 commits






 .github/workflows	Automatically close PRs	2 days ago
 .gitignore	Initial commit	3 days ago
 LICENSE	Initial commit	3 days ago
 README.md	Update README.md	3 days ago
 circle.html	Create circle.html	3 days ago
 style.css	Create style.css	3 days ago

README.md 

Readme

About

css-circle

-  Readme
-  Apache-2.0 License
-  0 stars
-  2 watching
-  1 fork

Releases

No releases published


[Create a new release](#)

Packages




No packages published

[Publish your first package](#)




A copy of the source repository has now been added as one of your GitHub repositories. This is the origin repository.

 **upkarlidderr / gkpbt-css-circle** Public



forked from [ibm-developer-skills-network/gkpbt-css-circle](#)







 Pin  Watch **0**  Fork **1**


[Code](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

 main  1 branch  0 tags [Go to file](#) [Add file](#) [Code](#)






This branch is up to date with [ibm-developer-skills-network:main](#). [Contribute](#) [Fetch upstream](#)

 **upkarlidderr** Automatically close PRs 729ceb2 2 days ago  **5** commits

 <code>.github/workflows</code>	Automatically close PRs	2 days ago
 <code>.gitignore</code>	Initial commit	3 days ago
 <code>LICENSE</code>	Initial commit	3 days ago
 <code>README.md</code>	Update README.md	3 days ago
 <code>circle.html</code>	Create circle.html	3 days ago
 <code>style.css</code>	Create style.css	3 days ago

README.md 

Readme

About
css-circle
 Readme
 Apache-2.0 License
 **0** stars
 **0** watching
 **1** fork

Releases
No releases published
[Create a new release](#)

Packages
No packages published
[Publish your first package](#)

Languages

HTML 69.9% **C**

Exercise 3: Clone the forked repository

A clone is a local copy of a repository. Before you can clone the forked repository, you first need its HTTPS URL, which provides secure access to it.

To clone the forked repository, complete the following steps:

1. In your list of repositories, click the forked repository. On the repository's main page, click the **Code** button.
2. Click the clipboard icon to copy the URL. Make sure the HTTPS tab is active.

upkarliddler / gkpbt-css-circle Public

forked from ibm-developer-skills-network/gkpbt-css-circle

Code Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags

This branch is up to date with ibm-developer-skills-network:main

upkarliddler Automatically close PRs

File	Commit	Time
.github/workflows	Automatically close PRs	3 days ago
.gitignore	Initial commit	3 days ago
LICENSE	Initial commit	3 days ago
README.md	Update README.md	3 days ago
circle.html	Create circle.html	3 days ago
style.css	Create style.css	3 days ago

README.md

Readme

About

css-circle

Readme

Apache-2.0

0 stars

0 watching

1 fork

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Languages

HTML 69.9%

- Open the terminal in the lab environment by using the menu in the editor: Terminal > New Terminal and clone the forked repo using git clone command

```
git clone <your repository HTTPS URL>
```

- Click Project and expand the folder of the project you just cloned. You can open the files in the editor, on the right side, by clicking on the file name.

Exercise 4: Add and commit your changes

A commit is Git's way of recording your file changes, similar to how you might save an edited document. To commit the change that you made in the previous exercise, you first need to add it to a staging area. Git will then take the staged snapshot of changes and commit them to the project. Remember, Git will never change files unless you explicitly ask it to.

To commit your new file, complete the following steps:

- To move the changes from your working project directory to the staging area, type the following command in the Terminal window:

```
git add .
```

The `git add` command has several options. The single `.` adds all untracked files in the current directory and subdirectories to the staging area. Alternatively, you can add the single file you created by using the `git add <file-name>` command. Finally, you can use `git add -A` to recursively add all files from the top level git folder.

2. To commit the new file to the local repository, you need to first tell git who you are. Type in the following commands to set your email and username. The email should be the same as your GitHub email.

Set your email:

```
git config --global user.email "email@example.com"
```

Set your name:

```
git config --global user.name "Your Github User Name"
```

3. Type the following command in the Terminal window to commit the file.
- Note:** It's always a good practice to add a description for the commit so you can remember what the change was if you have to refer to it later.
- **-m flag:** It is used in Git commit commands to specify the commit message directly in the command line, allowing you to provide a brief description of the changes you are committing.

```
git commit -m "Initial Commit"
```

As you can check command , `git status` now says there is nothing to commit and the working tree is clean. The new file is now ready to be pushed from your local system to origin on GitHub.

Exercise 5: Push your changes to origin”

This push will synchronize all the changes you made on your local system with your fork repository on GitHub.

To push your update to GitHub, complete the following steps:

1. In the Terminal window, run the following command:

```
git push
```

Note : When prompted, enter your GitHub account username and the PAT key. Do not worry if the PAT key is not visible when you paste it in the terminal. The key not being visible is a security feature. As soon as you press Enter, the system will start pushing the latest changes to the repository.

If your username and password were accepted, you should see the changes pushed to GitHub in the terminal.

2. Go to the fork repository in your GitHub account and verify that the local changes have now been added to the main branch.

For a comprehensive understanding of Git commands, delve into these detailed explanations.

git config with email

	Description
Syntax	git config --global user.email 'email'
Purpose	Set the global email address for Git, which ensures that every commit you make across all repositories will be associated with the specified email address
Example	git config --global user.email 'testuser3@abcmail.com' Sets the email address to testuser3@abcmail.com

git config with username

	Description
Syntax	<code>git config --global user.name 'username'</code>
Purpose	Set the global username for Git, which ensures that every commit you make across all repositories will be associated with the specified username
Example	<code>git config --global user.name 'John Doe'</code> Sets the user name to John Doe

git add [for file updates]

	Description
Syntax	<code>git add [file(s)]</code>
Purpose	Add file changes to the staging area
Example	<code>git add</code> Adds all changes in the current directory to the staging area

git add [for all updates]

	Description
Syntax	<code>git add --a / git add -A / git add -all</code>
Purpose	Stage all changes, including new files, modifications, and deletions, for the next commit All three command variations are identical commands used interchangeably to achieve the same functionality
Example	<code>git add --a</code> Stages all changes

git status

	Description
Syntax	<code>git status</code>
Purpose	Display the status of the working directory and staging area

git commit

	Description
Syntax	<code>git commit -m "[commit message]"</code>
Purpose	Record changes to the repository with a commit message
Example	<code>git commit -m "Initial commit"</code> Commits staged changes with the message "Initial commit"

git push

	Description
Syntax	<code>git push [remote] [branch]</code>
Purpose	Push committed changes to a remote repository
Example	<code>git push origin main</code> Pushes committed changes from the "main" branch to the remote repository

Note about data management and persistence

To ensure the proper management and persistence of your data in a GitHub repository, it is crucial to follow a few essential steps:

Regular Updates: Whenever you make changes or add new components to your project, it is essential to add, commit, and push the updates to your GitHub repository. This ensures that your latest work is safely stored and accessible to collaborators.

Session Persistence: During an active session, your data remains accessible. However, it's important to note that if your session expires or you log out, you will need to clone the repository again to resume work.

Ignoring node modules: When pushing data to GitHub, it's best practice to exclude the node modules folder from both your server and client directories. This folder contains external dependencies and can be quite large, making the repository heavy and slowing down the process. By adding it to the .gitignore file, you prevent it from being pushed to the repository, keeping your commits cleaner and more focused.

By adhering to these guidelines, you can maintain a well-organized and efficient GitHub repository, ensuring that your work is securely stored and easily accessible to you and your collaborators.

Author(s)

Ritika Joshi

© IBM Corporation. All rights reserved.