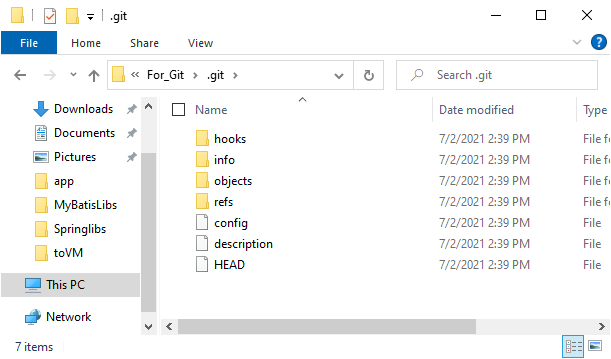
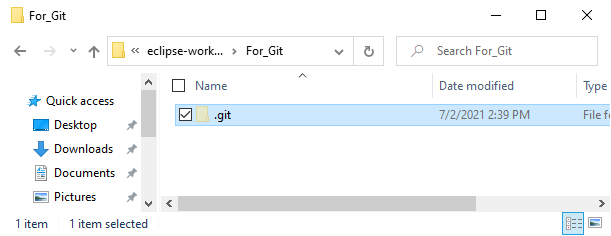
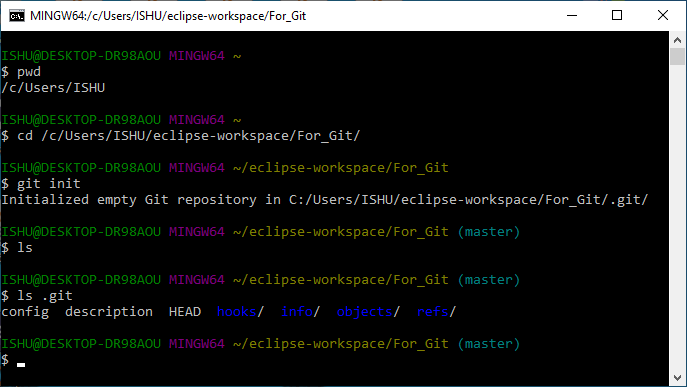
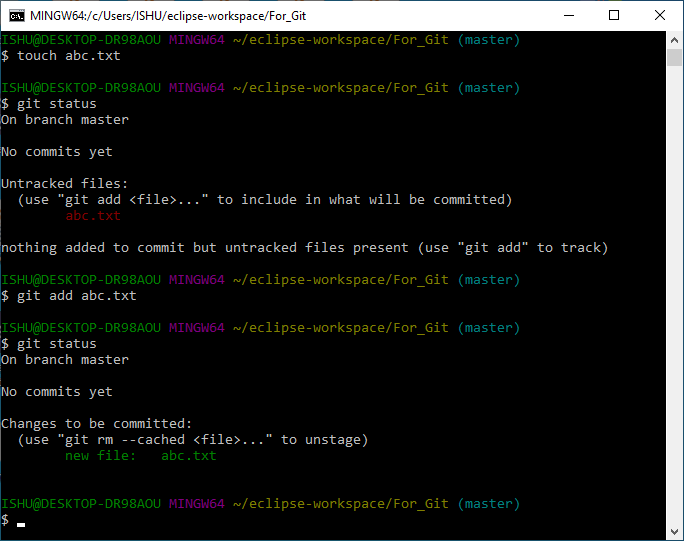
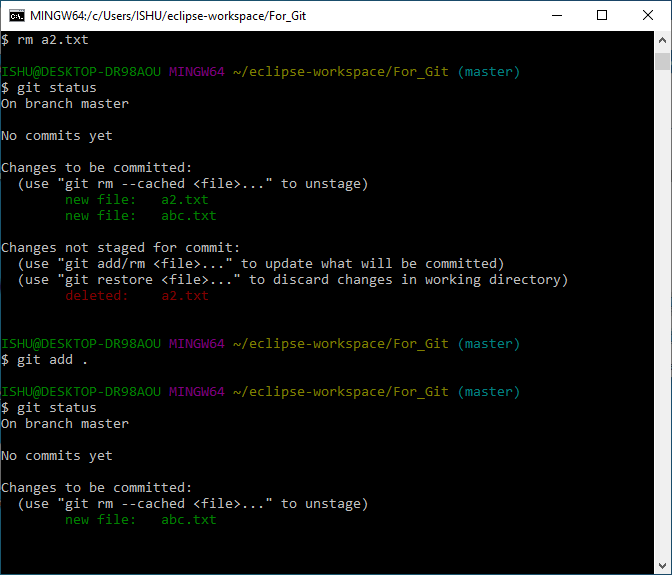
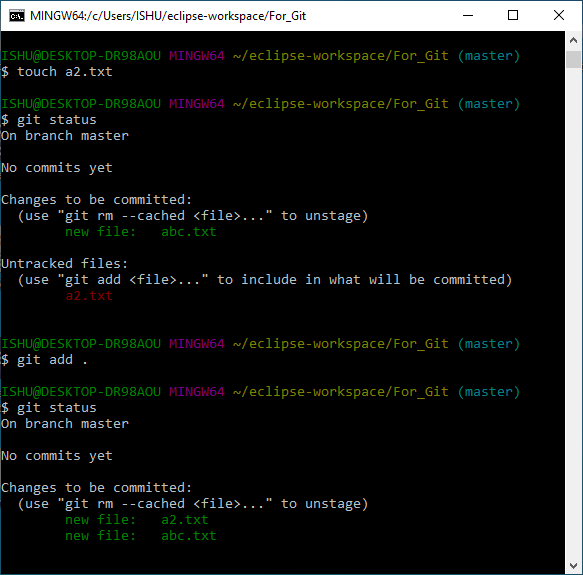
* Go to eclipse workspace->Go to project directory
* Create a hidden folder .git in your project directory by the command:
  + Open git bash
  + $cd /c/Users/ISHU/eclipse-workspace/For\_Git/
  + $ git init
    - Initialized empty Git repository in C:/Users/ISHU/eclipse-workspace/For\_Git/.git/
    - ISHU@DESKTOP-DR98AOU MINGW64 ~/eclipse-workspace/For\_Git (master)
  + $ ls
    - ISHU@DESKTOP-DR98AOU MINGW64 ~/eclipse-workspace/For\_Git (master)
  + $ ls .git
    - config description HEAD hooks/ info/ objects/ refs/
    - ISHU@DESKTOP-DR98AOU MINGW64 ~/eclipse-workspace/For\_Git (master)



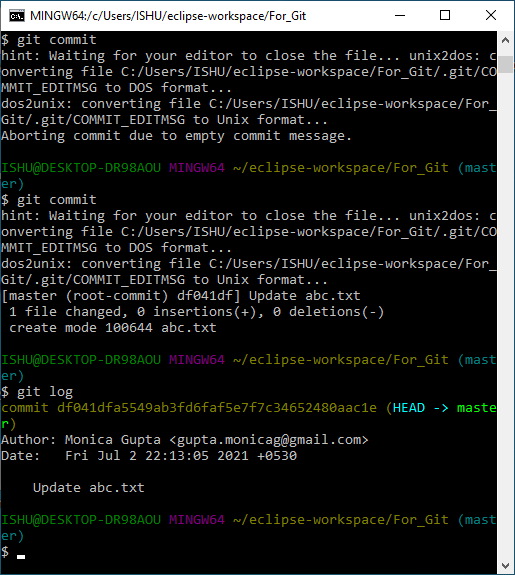
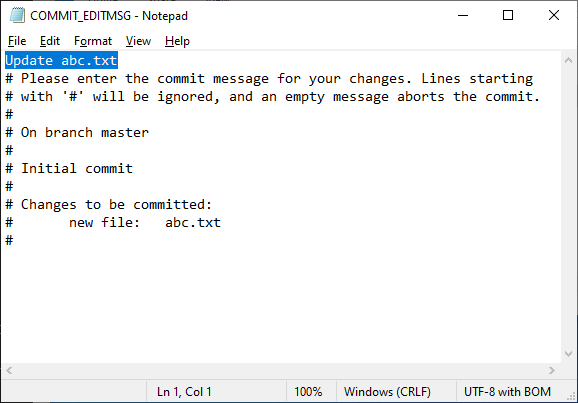
* Add a file to git
  + Open git bash
  + Create an empty file
    - $ touch abc.txt
  + Check the status to see the non-added file
    - $ git status
  + Add the file to git
    - $ git add abc.txt
  + Adding all files into git:
    - $ git add -A
    - Or
    - $ git add .
  + Check status now
    - $ git status



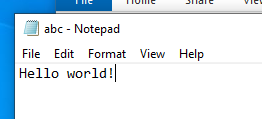
* Remove a file from git
  + Open git bash
  + Create an empty file
    - $ touch a2.txt
  + Adding all files into git:
    - $ git add .
  + Check status
    - $ git status
  + Remove the file
    - $ rm a2.txt
  + Remove the file from git:
    - $ git add .
    - Or
    - $ git add a2.txt
  + Check Status
    - $ git status

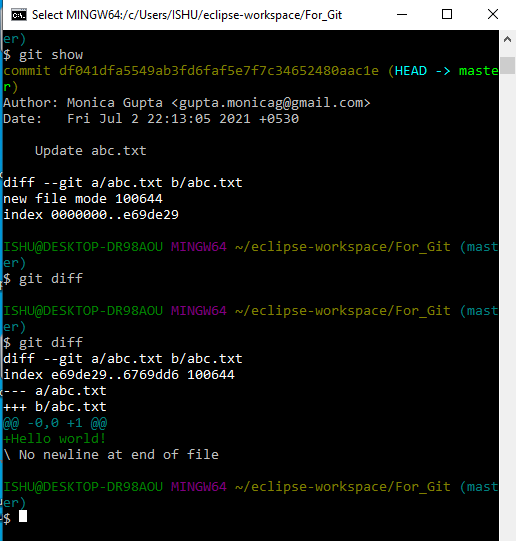


* The git commit command
* The commit command will commit the changes and generate a commit-id. The commit command without any argument will open the default text editor and ask for the commit message. We can specify our commit message in this text editor. It will run as follows:
  + $ git commit
* The above command will prompt a default editor and ask for a commit message.
* As we run the command, it will prompt a default text editor and ask for a commit message. Type there:
  + Update <filename>
* If the editor is Vi: Press the **Esc** key and after that '**I**' for insert mode. Type a commit message whatever you want. Press **Esc** after that '**:wq**' to save and exit from the editor. Hence, we have successfully made a commit.
* Else save and close the editor.
* We can check the commit by git log command:
  + $ git log

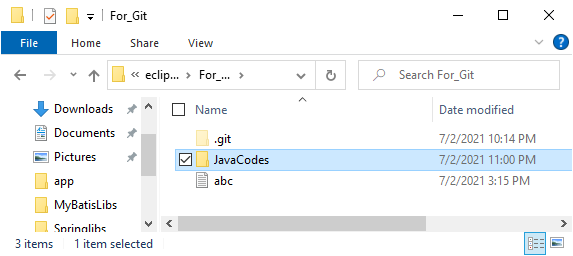
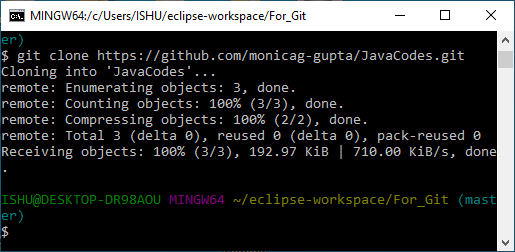
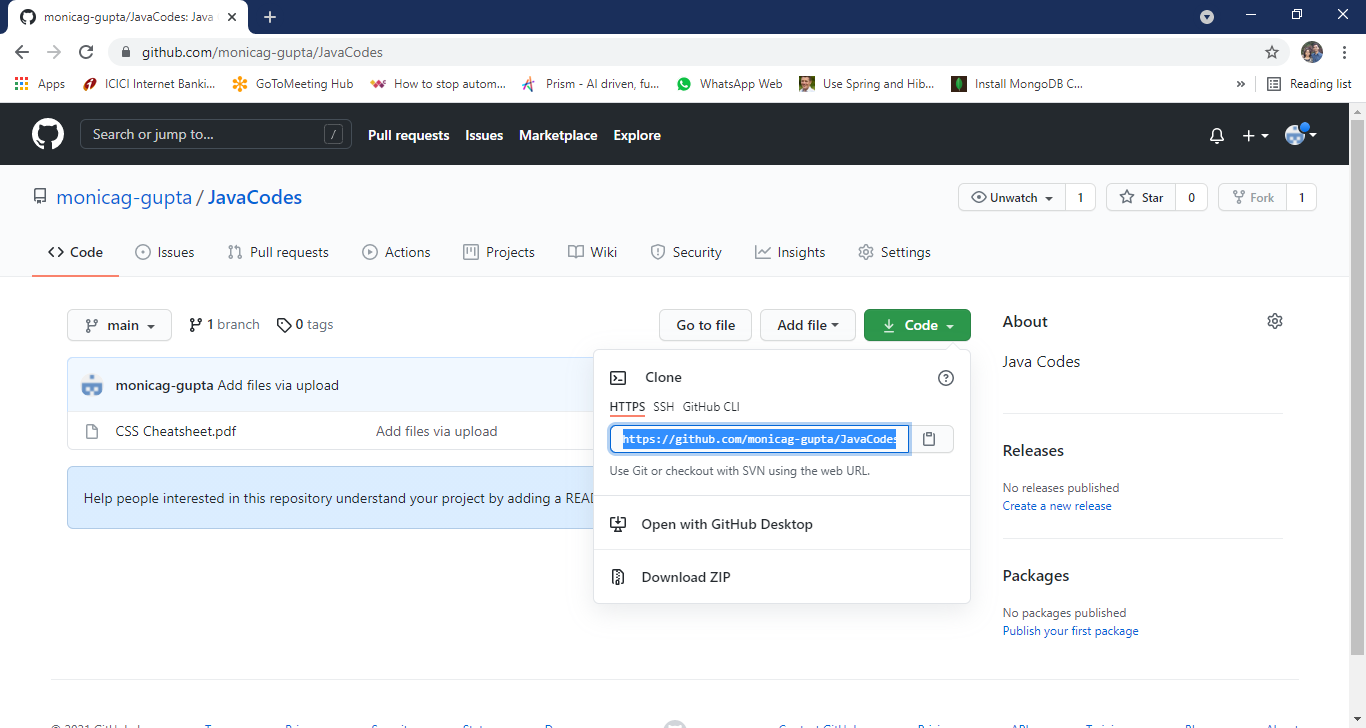


* **Git - Review Changes**
* Git log
* We can check the commit by git log command:
  + $ git log
* Git show
* The git show command takes SHA-1 commit ID as a parameter.
  + $ git show
* Git diff
* Git diff shows **'+'** sign before lines, which are newly added and **'−'** for deleted lines.
  + $ git diff

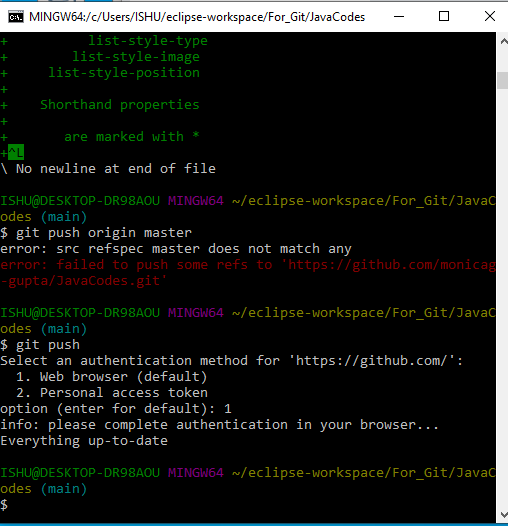
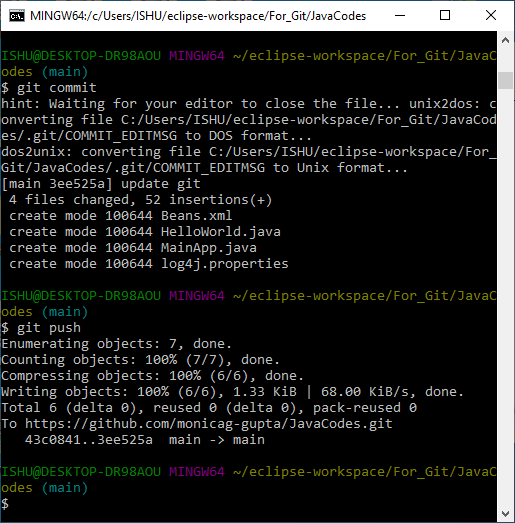
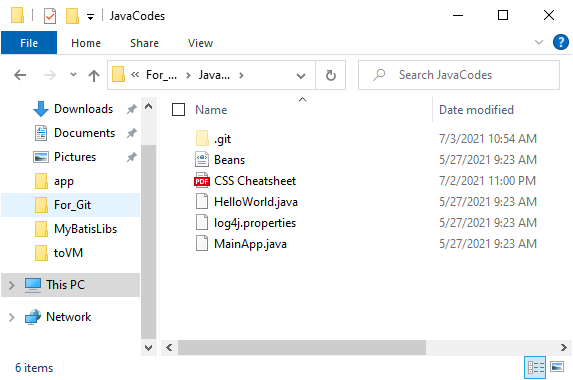


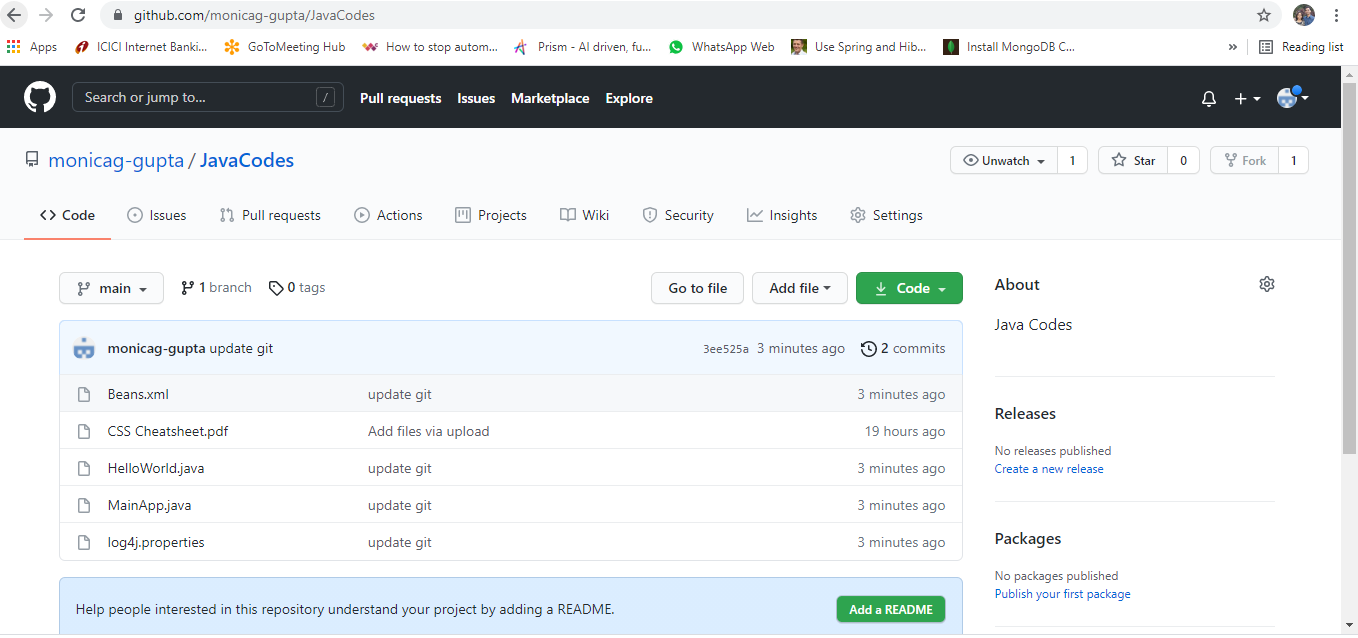


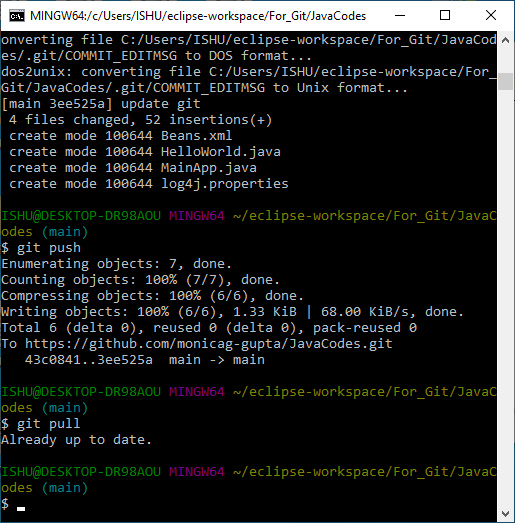
* The git clone
* Goto the repository page:
  + <https://github.com/monicag-gupta/JavaCodes>
* Copy URL from clone:
  + <https://github.com/monicag-gupta/JavaCodes.git>
* Open git bash
* Clone the git:
  + $ git clone https://github.com/monicag-gupta/JavaCodes.git



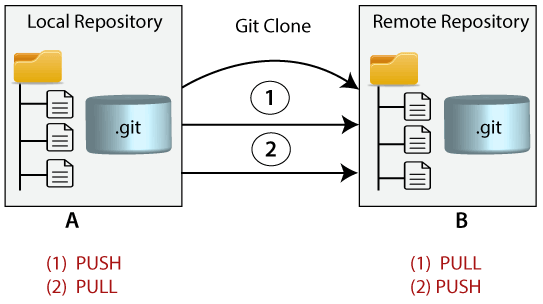
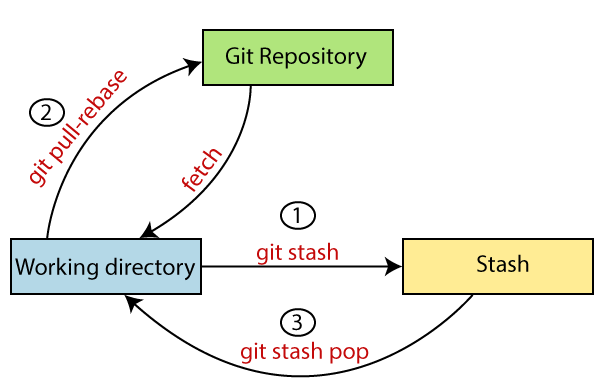
* **After adding files**
* **$ git add .**
* **$ git commit**
* **Git - Push**
  + $ git push
* (Use Browser to authenticate and give all authority to git)



* **Check github**
* Git pull
* Fetch Latest Changes from github:
* $ git pull
* Check for the differences in the code if any.
* Then we can push into github the files we have changed.



Getting Started-Notes

* Add all Modified and Deleted Files
* Git add facilitates us with a variety of options. There is another option that is available in Git, which allows us to stage only the modified and deleted files. It will not stage the newly created file. To stage all modified and deleted files only, run the below command:
  + $ git add -u
* Add Files by Wildcard
* Git allows us to add all the same pattern files at once. It is another way to add multiple files together. Suppose I want to add all java files or text files, then we can use pattern .java or .txt. To do so, we will run the command as follows:
  + $ git add \*.java
* The above command will stage all the Java files. The same pattern will be applied for the text files.
* To undo an add operation, run the below command:
  + $ git reset <filename>
* The commit command also provides -a option to specify some commits. It is used to commit the snapshots of all changes. This option only consider already added files in Git. It will not commit the newly created files.
* $ git commit -a
* The above command will prompt our default text editor and ask for the commit message. Type a commit message, and then save and exit from the editor. This process will only commit the already added files. It will not commit the files that have not been staged.
* The -m option of commit command lets you to write the commit message on the command line. This command will not prompt the text editor. It will run as follows:
* $ git commit -m "Commit message."
* The above command will make a commit with the given commit message.
* We can also use the -am option for already staged files. This command will immediately make a commit for already staged files with a commit message. It will run as follows:
* $ git commit -am "Commit message."
* Git Commit Amend (Change commit message)
* The amend option lets us to edit the last commit. If accidentally, we have committed a wrong commit message, then this feature is a savage option for us. It will run as follows:
* $ git commit -amend
* The above command will prompt the default text editor and allow us to edit the commit message.
* We may need some other essential operations related to commit like revert commit, undo a commit, and more, but these operations are not a part of the commit command. We can do it with other commands. Some essential operations are as follows:
* Git undo commit, Git revert commit, git remove commit.
* Git Clone
* In Git, cloning is the act of making a copy of any target repository. The target repository can be remote or local. You can clone your repository from the remote repository to create a local copy on your system. Also, you can sync between the two locations.
* The **git clone** is a command-line utility which is used to make a local copy of a remote repository. It accesses the repository through a remote URL.
* Usually, the original repository is located on a remote server, often from a Git service like GitHub, Bitbucket, or GitLab. The remote repository URL is referred to the **origin**.
* **Syntax:**
* $ git clone **<repository** URL**>**
* 
* Suppose, you want to clone a repository from GitHub, or have an existing repository owned by any other user you would like to contribute. Steps to clone a repository are as follows:
* Step 1:
* Open GitHub and navigate to the main page of the repository.
* Step 2:
* Under the repository name, click on Clone or download.
* Step 3:
* Select the **Clone with HTTPs section** and **copy the clone URL** for the repository. For the empty repository, you can copy the repository page URL from your browser and skip to next step.
* Step 4:
* Open Git Bash and change the current working directory to your desired location where you want to create the local copy of the repository.
* Step 5:
* Use the git clone command with repository URL to make a copy of the remote repository. See the below command:
* $ git clone https://github.com/abc/xyz.git
* Now, Press Enter. Hence, your local cloned repository will be created.
* Cloning a Repository into a Specific Local Folder
* Git allows cloning the repository into a specific directory without switching to that particular directory. You can specify that directory as the next command-line argument in git clone command. See the below command:
* $ git clone https://github.com/abc/xyz.git "new folder"
* The given command does the same thing as the previous one, but the target directory is switched to the specified directory.
* Git has another transfer protocol called SSH protocol. The above example uses the git:// protocol, but you can also use http(s):// or user@server:/path.git, which uses the SSH transfer protocol.
* Git Clone Branch
* Git allows making a copy of only a particular branch from a repository. You can make a directory for the individual branch by using the git clone command. To make a clone branch, you need to specify the branch name with -b command. Below is the syntax of the command to clone the specific git branch:
* Syntax:
* $ git clone -b <Branch name><Repository URL>
* $ git clone -b master https://github.com/abc/xyz.git "new folder"
* In the given output, only the master branch is cloned from the principal repository Git-Example.
* Git Stash
* Sometimes you want to switch the branches, but you are working on an incomplete part of your current project. You don't want to make a commit of half-done work. Git stashing allows you to do so. The **git stash command** enables you to switch branches without committing the current branch.
* Generally, the stash's meaning is "**store something safely in a hidden place**." The sense in Git is also the same for stash; Git temporarily saves your data safely without committing.
* Stashing takes the messy state of your working directory, and temporarily save it for further use. Many options are available with git stash. Some useful options are given below:
* **Git stash**
* **Git stash save**
* **Git stash list**
* **Git stash apply**
* **Git stash changes**
* **Git stash pop**
* **Git stash drop**
* **Git stash clear**
* **Git stash branch**
* 
* **Git Stash**
* $ git stash
* The work gets stashed in its current position. Now, the directory is cleaned. At this point, you can switch between branches and work on them.
* **Git Stash Save (Saving Stashes with the message):**
* $ git stash save "<Stashing Message>"
* **Git Stash List (Check the Stored Stashes)**
* $ git stash list
* **Git Stash Apply**
* You can re-apply the changes that you just stashed by using the git stash command. This restores the last stash.
* $ git stash apply
* $ git stash apply **<stash** id**>**
* **Git Stash Changes**
* We can track the stashes and their changes.
* $ git stash show
* We can exactly track what changes are made on the file. To display the changed content of the file, perform the below command:
* $ git stash show -p
* Here, -p stands for the partial stash. The given command will show the edited files and content.
* It acts the same as git diff command. The **git diff** command will also show the exact output.
* **Git Stash Pop (Reapplying Stashed Changes)**
* The popping option removes the changes from stash and applies them to your working file.
* The git stash pop command is quite similar to git stash apply. The main difference between both of these commands is stash pop command deletes the stash from the stack after it is applied.
* $ git stash pop
* The above command will re-apply the previous commits to the repository.
* **Git Stash Drop (Unstash)**
* The git stash drop command is used to delete a stash from the queue. Generally, it deletes the most recent stash. Caution should be taken before using stash drop command, as it is difficult to undo if once applied.
* The only way to revert it is if you do not close the terminal after deleting the stash.
* $ git stash drop
* $ git stash drop **<stash** id**>**
* Assume that I have two stashes available in my queue, and I don't want to drop my most recent stash, but I want to delete the older one. Then, it will be operated as:
* $ git stash drop stash@{1}
* **Git Stash Clear**
* The git stash clear command allows deleting all the available stashes at once. It deletes all the available stashes that exist in the repository.
* $ git stash clear
* **Git Stash Branch**
* If you stashed some work on a particular branch and continue working on that branch. Then, it may create a conflict during merging. So, it is good to stash work on a separate branch.
* **The git stash branch command allows the user to stash work on a separate branch to avoid conflicts.**
* **$ git stash branch <Branch Name>**
* **The above command will create a new branch and transfer the stashed work on that.**
* **It will avoid the merge conflict on the master branch.**