# How to Upload your Selenium Project to GITHUB repository using GIT?

1. [Create a new repository](https://help.github.com/articles/creating-a-new-repository) on GitHub. To avoid errors, do not initialize the new repository with *README*, license, or gitignore files. You can add these files after your project has been pushed to GitHub.
2. Open Git Bash.
3. Change the current working directory to your local project.
4. Initialize the local directory as a Git repository.

(git init)

1. Add the files in your new local repository. This stages them for the first commit.

(git add .)

#Adds the files in the local repository and stages them for commit. To unstage a file, use 'git reset HEAD *YOUR-FILE*'.

1. Commit the files that you've staged in your local repository.

(git commit -m "First commit")

# Commits the tracked changes and prepares them to be pushed to a remote repository. To remove this commit and modify the file, use 'git reset --soft HEAD~1' and commit and add the file again.

1. At the top of your GitHub repository's Quick Setup page, click to copy the remote repository URL.
2. In the Command prompt, [add the URL for the remote repository](https://help.github.com/articles/adding-a-remote) where your local repository will be pushed.

(git remote add origin *remote repository URL)*

# Sets the new remote

(git remote -v)

# Verifies the new remote URL

1. [Push the changes](https://help.github.com/articles/pushing-to-a-remote) in your local repository to GitHub.

(git push -u origin master)

# Pushes the changes in your local repository up to the remote repository you specified as the origin

1. Pull the changes to your local repository from GitHub.

(git pull origin master)

# Some Additional Commands are given below:

Initiate the directory

git init

Create

Dir

mkdir project.git

File

touch newFile.txt

echo “This text is added to the end of the file” >> newFile.txt

Clone

git clone gituser@git.server.com:project.git

git clone <repository link>

Perform changes

Appending changes

echo "abcd" >> File Name

replacing changes

echo "abcd" > File Name

Review changes

git status

git log

git show (commit id)

git diff

Commit and changing commit

git commit Filename -m "Message"

git commit --amend -m "Filename"

Push Operation

git push origin master

git push link of remote repository

Stash operation

git stash

git stash list

git stash pop

Move operation

git mv File name location

Rename operation

git commit -a -m 'Added Makefile and renamed OldfileName to

NewfileName

Delete

git rm <filename>

Tag

$ git tag

git tag TagName

git push --tags

git show TagName

Patch

Create patch file

git format-patch master --stdout > newFile

apply patch

git apply --stat FileName.patch

Branch

git branch

git branch BranchName

git checkout BranchName

git checkout -b NewBranchName

git checkout -d NewBranchName

git branch -m new\_branch RenamedBranchName

Merging

git merge BranchName

git merge BranchName --no-commit

Remote

git remote add origin Link

Fetch

git fetch RemoteName

remove directory

rm -rf DeleteFolderName

**Git Commands:**

Git command summary cheat sheet:

| **Command** | **Description** | **Man Page** |
| --- | --- | --- |
| git **--help** | List Git commands | git |
| git --help *command* | Help on given "*command*" | git help |
| git **add** *path/filename*  git add file1 file2 file3  git add --update  git add --all | Add a file or directory to Git CM control. This does not commit the files to the repository but merely adds files to "staging" status. Must also perform: git commit to enter changes in the repository.  View staged files: git ls-files --stage  Empty directories are not added. Git manages files only.  --update: stage all tracked and modified files.  Undo an add: git reset HEAD *path/filename* | git add |
| git **blame** *filename* | Show file line numbers annotated with author information. Also see git log | git blame |
| git **branch** -a | List all branches (remote and local) | git branch |
| git **checkout** *parentPath/filename* git checkout master git checkout *e3b43d63* | Undo local changes. Revert changes. This directive is misleading as it implies a client-server action. Instead, it relies on the local repo and sets the working copy to the unchanged version, reverting changes.  Checkout "master" will set the local work area to match the master branch. You can also specify the commit to which you want the local repo to view.  Also see git reset | git checkout |
| git **clean** -df  git clean -df  git -n clean  git -dn clean | Remove files not managed by Git.  -df: All unchecked files and directories are removed.  -n: Dry run. Command not executed.  -d: directories  Clean has nothing to do with an Ant target of "clean" and is in no way related. Also see git gc | git clean |
| git **clone** *http://git.megacorp.com/git/repo/projectx.git* git clone*ssh://user@git.megacorp.com:/srv/git/repo/projectx.git* | Copy/clone/create a local repository from the remote repository specified.  [Potential Pitfall]: The following error often is a connectivity issue:  fatal: http://git.megacorp.com/git/repo/projectx.git/info/refs not valid: is this a git repository?  If a proxy error is keeping you from connecting. Fix: unset http\_proxy | git clone |
| git **commit**  git commit *dir-path/filename* git commit -m "Message goes here." *filename* | Check-in (commit) local staged "working" files into local repository. Files must be staged for commit using git add Files are recursively located and committed into the local git repository. Git commits are Atomic, i.e. all files are committed or none are committed, no incomplete check-in. | git commit |
| git **config** --global user.name "John Doe" git config --global user.email jdoe@megacorp.com  git config -l  git config --global --unset user.password | This configures ~/.gitconfig  Show your local GIT configuration for your local repo  Clear cached password. Will be prompted on next push/pull. [Potential Pitfall]: Error: fatal: Authentication failed for 'http://git.megacorp.com/git/repo/projectx.git/’ Fix: "git config --global --unset user.password" to clear passord cache. | git config |
| git **diff**  git diff e3b43d63 60fdba79 *directory/path* | Show file diffs between the working tree and local repo.  Show file diffs between two commits. | git diff |
| git **gc** | Cleanup unnecessary files and optimize/compress local repo. | git gc |
| git **log**  git log --oneline  git log *directory/path*/*file*  git log --author="*name*"  git log --graph --decorate --oneline | Show the Git log messages.  --oneline: brief description  --graph --decorate: display branching like gitk except for text terminals.  Note that the letter "q" will quit pagination mode. | git log |
| git **ls-files** --stage  git ls-files --cached  git ls-files --modified  git ls-files --others  git ls-files --deleted  git ls-files --unmerged  git ls-files --killed  ... | List files staged.  List cached files.  List modified files.  List other (un-tracked) files.  List deleted files.  List un-merged files.  List killed files that need to be removed due to file/directory conflicts for checkout-index to succeed. | git ls-files |
| git **ls-remote** -h *http://domain/git/repo/projectx.git*HEAD | List files under Git control on remote Git server. | git ls-remote |
| git **merge** *branch-name* git merge --abort | Merge a specified branch into your current branch you are in. If you are in the "master" branch, then "branch-name" will be merged into "master". --abort: After starting a merge, you might want to stop the merge and return everything to its pre-merge state.  --stat: Show a diffstat after merge completion -m: Message to be included in the merge commit | git merge |
| git **mv** *file-old-name* *file-new-name* | Rename a file. Moves/renames file. | git mv |
| git **mergetool** | View file conflicts in a merge tool. Default is [vimdiff](http://man.yolinux.com/cgi-bin/man2html?cgi_command=vimdiff)  Assign a GUI tool:  KDiff3:   * git config --global merge.tool [kdiff3](http://man.yolinux.com/cgi-bin/man2html?cgi_command=kdiff3) * git config --global mergetool.kdiff3.cmd 'kdiff3 $LOCAL $BASE $REMOTE -o $MERGED'   Meld:   * git config --global merge.tool meld * git config --global mergetool.meld.cmd 'meld --diff $LOCAL $BASE $REMOTE --output $MERGED'  or: meld $LOCAL $MERGED $REMOTE --output $MERGED  or: meld $LOCAL $BASE $REMOTE --output $MERGED  (preference of three panes ordered left to right)   Tools options include: p4merge, meld, kdiff3, tkdiff, gvimdiff, vimdiff, ... | git mergetool |
| git **pull** origin master | Update your local repo and working files with changes posted on the git remote server origin: refers to repository from which you will receive updates master: refers to the master branch from which you will receive updates This command is the same as performing the following two commands: git fetch git merge | git pull |
| git **push** origin master | Update the remote repo from the staged (identified with "git add") local working files origin: refers to repository from which you will receive updates master: refers to the master branch from which you will receive updates This command is the same as performing the following two commands: git fetch git merge | git pull |
| git **remote** -v | Show URL to origin server. | git remote |
| git **reset** HEAD^ *path/filename*  git **reset** HEAD~1 *path/filename*  git **reset** --hard HEAD^ *path/filename* | Remove from staging (local copy of files still modified)  --hard: Don't keep changes, remove modifications to files.  HEAD^: latest commit notation for Linux and bash shell terminals.  HEAD~1: latest commit notation for Microsoft CMD shell. Microsoft DOS shell does not support "^".  Reset the repo to the last good commit specified and ignoring everything after e3b43d63:  git reset --hard e3b43d63  git push –force  --hard : will erase your local work if you have anything stashed.  --mixed : Unstaging all changes but leave them in the working directory (default).  --soft : staged and working directory are not altered | git reset |
| git **revert** HEAD | Not a commit undo but overwrites the commit with a new commit to reverse changes. Usually better to use "git reset --hard HEAD^*path/filename*" when you want to abort your changes to a file. If you are trying to get back to a previously committed state, just use git checkout e3b43d63 | git revert |
| git **rm** *filename* git rm -r *directory/path* | Delete file from local working index. Don't use the UNIX command rm *file-name*. Must perform a "git commit" to update the repository.  -n: dry run  -r: recursive removal when given a directory name. | git rm |
| git **status** | Show tracked files which have been staged, committed, etc. | git status |
| git **tag** -a *April\_Sprint* -m "Source used for April sprint demo" git tag -a *April\_Sprint* -m "*Source used for April sprint demo*" *e3b43d63* | Add a "tag" to the commit to identify it in human readable terms. You can then push using the following: git push origin master *April\_Sprint* or  git push origin master --tags  -a: Generate an annotated tag object. -d: Delete existing tags with the given names. -f: Replace an existing tag with the given name. -m: Add a message. | git tag |