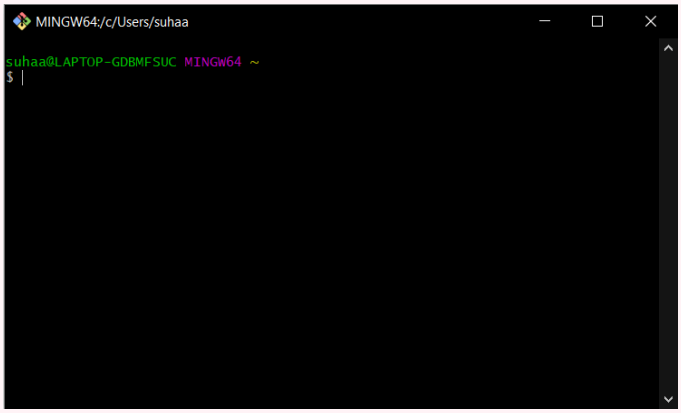
Version Control with Git and Github

Configuration:

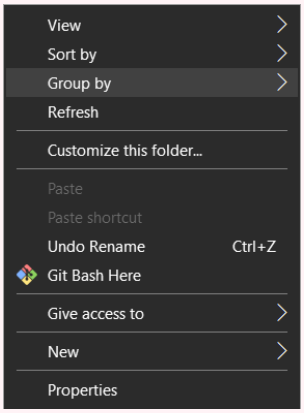
1. Install Git for your OS: https://git-scm.com/downloads. Configure with default settings. You may change the default editor to any editor of your choice.
2. Open 'Git Bash'. If everything goes right, this window will open.



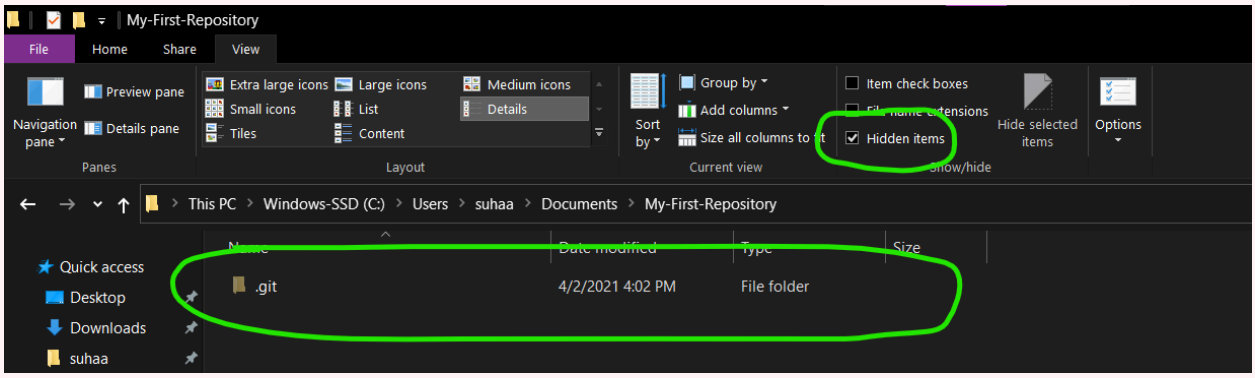
3. Configure git:
   a. `git config --global user.name "<Your-Full-Name>"`
   b. `git config --global user.email "<your-email-address>"`

Creating a repository:

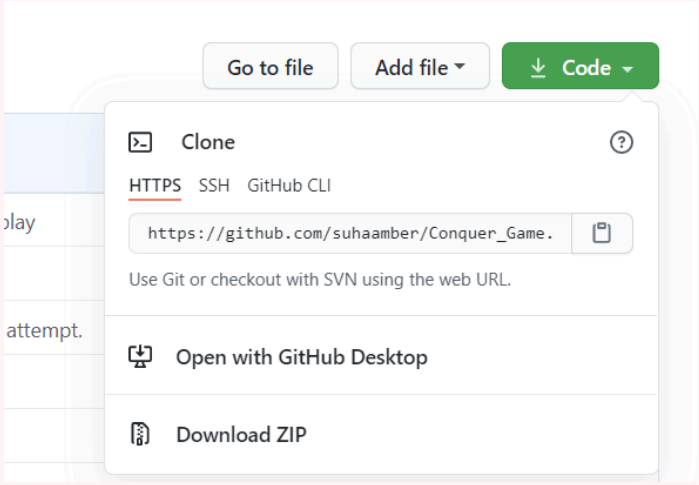
1. Navigate to the directory/folder, right-click and select `Git Bash here`



2. Type `git init` to create a repository
3. If Hidden Items is checked in the View Tab (under Show/Hide section), you will be able to see the .git folder that contains information about your repository including details of commits and remote repo address.



4. **You can now create files of your own in the folder OR clone a repo from Github**
5. `git clone https://github.com/username/repo_name.git`**: clone an existing repository from Github. You can get the link from the repository on Github. When cloning, do not create a new folder. The repository is cloned wherever you git bash.**



Dealing with the staging area:

1. `git add <filename1> [<filename2> <filename3> … ]` **will add files to the staging area. You can also stage entire folders/directories.**
2. `git add .` **: will add all modified files to the staging area.**
3. `git restore --staged <filename>`**: will remove files from the staging area (more on undoing things later).**

Committing:

1. `Git commit`**: commit all staged files. Opens an editor to enter a commit message.**
2. `Git commit -m "commit-message-here"`**: commit with inline comment.**

3. `Git commit -a`: commit all files that were at one point added to the staging area (includes modified and deleted files, new files ignored).

Inspecting your repository:

1. `git status` shows you the current status of the repository.

```
suhaa@LAPTOP-GDBMFSUC MINGW64 ~/Documents/My-First-Repository (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test.java.txt

nothing added to commit but untracked files present (use "git add" to track)
```

   a. Shows current branch
   b. Status of commits with respect to your remote repository
   c. Modified files (staged)
   d. Untracked files
   e. Status of merge (if manual change required)

2. `git log` shows you your commit history. You can use add-ons to find a specific set of commits.
   a. `git log --oneline`: summarizes commits in one line
   b. `git log --stat`: show changes made in each commit
   c. `git log <filename>`: commits made to the <filename>
   d. `git log --graph`: shows commits visually
   e. `git log --author="<user.name>"`: shows commits made by user
   f. `git log branch="<branch_name>"`: shows commits made in the given branch (git log branch=* displays commits made in all branches)
   g. `git log --patch`: shows the full diff of each commit
   h. `git log --grep "<commit_message>"`: shows commits that matches the <commit_message> regular expression in the commit message

Detailed explanation of commands here.

Press q to exit less (the scrolling program)

3. `git diff`: show differences between last commit and current uncommitted changes
   a. `git diff SHA1 SHA2`: Compare differences between two specific commits.
   b. `git diff branch1 branch2 filename`: compare differences between a file in two different files

4. The HEAD pointer: This special pointer points to a latest commit in your commit history. Use git log --oneline --graph to visualise the state of your repository.

```
suhaa@LAPTOP-GDBMFSUC MINGW64 ~/Documents/GenericPortfolio (master)
$ git log --oneline --graph
* 4e01939 (HEAD -> master, origin/master) Swapped jumbotron and social media ico
ns
*   71965d8 Merge branch 'createJumbotron'
|\
| * 435711a (origin/createJumbotron, createJumbotron) Bootstrap script added. Ju
mbotron content centered
| * cc876e0 Jumbotron added to index.html
* | 7c7caf3 Social media links added
|/
* 1590e51 Branch created to test offcanvas
* e45ff02 Design first section
* ff2e099 Created index files
```

The head currently points to the master branch (because the repository is checked out to the master branch). It is also the latest commit. Each commit has a unique SHA (Secure Hash Algorithm) which acts as an identifier for the commit. This SHA can be used to isolate a commit while using git log or git diff.

It may become a little tedious and vulnerable to error to use the SHA while identifying the commit you want to target. The HEAD pointer is a solution to this problem. It also comes with a set of operators:

- '~' (tilde)
  a. HEAD~ is the parent of the HEAD commit
  b. HEAD~~ is the grandparent of the HEAD commit
  c. HEAD~~~ is the great-grandparent and so on.
  d. HEAD~2 is the grandparent (alternate way of representing)
  e. HEAD~n where n=1,2,3,4... is the nth parent

Note: ~ always follows a straight line. It does not consider any branches.

- '^' (caret)
  a. HEAD^ is the first parent
  b. HEAD^^ is the first grandparent

c. HEAD^2 is the second parent (only applies to merge commits)

Rule of thumb: ~ goes back a number of generations and ^ navigates through parents.

Branching and merging:

1. Git branch

a. git branch: display branches

b. git branch branch_name: create a branch at the current commit with the name branch_name

c. git branch -d branch_name: deletes inactive branch (fails if active, or contains independent changes that are not merged.)

d. git branch -D branch_name: force delete branch

e. git checkout branch_name: HEAD points to active branch

2. Git merge

a. Make sure you're updated with your remote repo (if any) by using git pull.

b. Checkout to your **merge-receiving** branch. Further commits will happen on this branch.

c. `git merge branch-to-be-merged`: **merge branch-to-be-merged with current branch.**

3. Managing conflict

a. Merging fails before merging if there are changes in the working directory or staging area

b. If merging fails while merging, there is conflict with another developer's code. You need to manually choose which changes to keep. You have the following 2 choices:

i. `git merge --abort`: to abort the merge

```
<<<<<<< HEAD
this is some content to mess with
content to append
=======
totally different content to merge later
>>>>>>> new_branch_to_merge_later
```

ii. Merge conflicts are shown in your source code like this. You must open your code file and manually change your code.

Undoing things:

1. `git restore --staged <filename>`: **unstage a staged file**
2. `git restore <filename>`: **discard changes made after previous commit in the working directory (This modifies your working directory. You cannot recover these changes! Use wisely.)**
3. `git commit --amend`: **If you've made a mistake in your commit message or forgotten to add a file to the staging area before the commit (only applies to the latest commit), you can use this command to edit your commit. In case you want to edit your commit message, it will open your editor with your previous commit message that is open to editing. Example:**

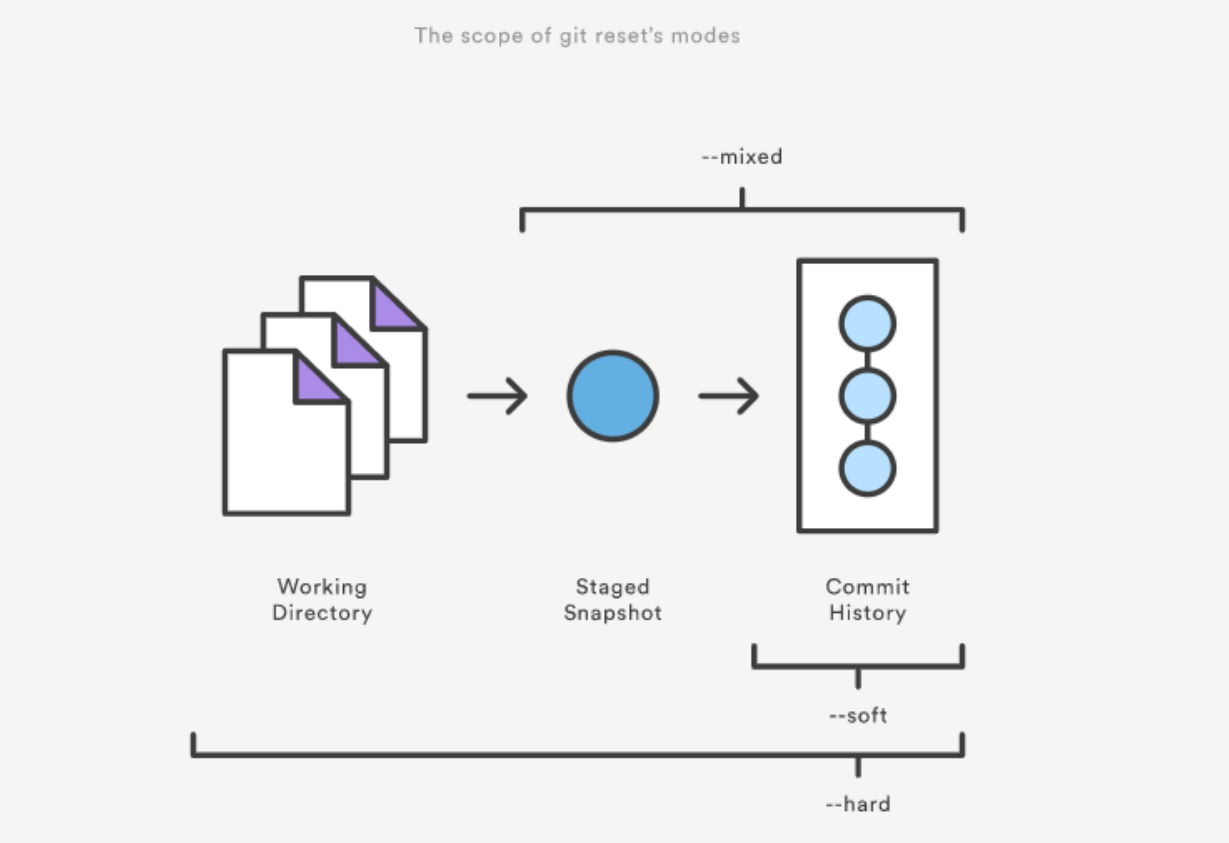
```
$ git commit -m 'Initial commit'
$ git add forgotten_file
$ git commit --amend
```

4. **git reset --[soft|mixed|hard] <commit SHA>**

a. `git reset --soft <commitSHA>`: **resets only your commit history to the state it was when the commit with SHA was created.**

b. `git reset --mixed <commitSHA>`: **resets your commit history and staging area to the state it was when the commit with SHA was created. This is the default mode if no flag is provided.**

c. `git reset --hard <commitSHA>`: **resets your commit history, staging area and working directory to the state it was when the commit with SHA was created. (This modifies your working directory. You cannot recover these changes! Use wisely.)**



Dealing with remote repositories (Github via HTTPS):

1. `git clone https://github.com/username/repo_name.git`: **clone an existing repository from Github. You can get the link from the repository on Github. When cloning, do not create a new folder. The repository is cloned wherever you git bash.**
2. `git clone` **automatically sets a remote connection to the repository you cloned from.**
3. `git remote add name-of-remote url`: **sets a remote connection from your repo to the URL with the name provided**
4. `git remote -v`: **shows you all remote connections in your repo**
5. `git remote rm name-of-remote`: **deletes remote connection provided**

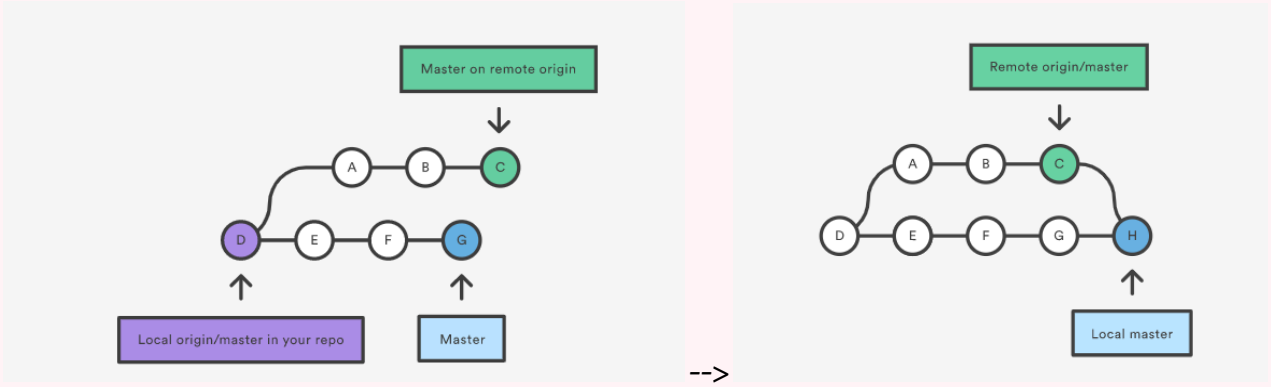
Pushing commits to your remote repository:

1. `git status` **can give you the status of your commits with respect to your remote.**

2. `git push -u remote-name branch-name`: **pushes branch-name into remote-name. -u is used to set up an upstream branch that tracks the remote branch (helping you keep track of which branches are ahead, and which behind your current status).**
3. `git push remote-name -all`: **pushes all branches to remote**

Fetch and pull commits from your remote repository

1. **git fetch remote-name: fetches commits from remote-name. (This only fetches updated commits but does not merge them. You can access them by checking out to future commits in the remote branch but cannot make any saved changes till you merge)**
2. **git fetch remote-name branch-name: fetches the given branch from remote.**
3. **git fetch --dry-run: simulates fetching. Gives you a demo-run to see how it would affect your repo.**
4. **git pull remote-name: pulls commits from remote-name. (Pull is a combination of fetch and merge. See below for visual demo.)**
5. **git pull remote-name branch-name: pulls commits from branch-name on remote-name.**



More resources:

[Learn Git with Bitbucket Cloud Tutorial]

[Git Documentation]

[All Git commands]

[Git Pro (eBook)]

[https://www.udacity.com/course/version-control-with-git--ud123]

[https://guides.github.com/activities/hello-world/]

[https://docs.github.com/en/github/getting-started-with-github/git-and-github-learning-resources]