1. **Introduction to git**

* Git is “distributed version control system”
* I.e.
  + It’s a system that records changes to our files over time
  + We can recall specific version of those files at any given time
  + Many people can easily collaborate on a project and have their own version of project files on their computer
* Git allows you to:
  + Store revisions in project history in one directory
  + Rewind to any version/revision in the project
  + Work on new features without messing up the main code base
  + Easily collaborate with other developers
* Github:
  + Online service that hosts our projects
  + Share our code with other developers
  + Developers can download the projects and then work on them
  + And they can re-upload their edits/changes and merge them with the main code base

**2 - Installing Git**

* Option 1:
  + Got to [https://git-scm.com/](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbElNSHNoUVlQeWF2TTFyMU9hdEUxYnlNdG01UXxBQ3Jtc0ttRzlFRl9lcVJzSVdkbHdSTk8wZjV4SkhfS0N1QkJwVTZWeng4NjhnRUlxZDg5YW4zMFlGeDUzYXZ3NUg5YmN3YnQ3SFNZWG40RmhTYnV5dGZTNDI5MVZUZ3N0N2tHMUpWS1FUR0RXNldkaHk1UjFfdw&q=https%3A%2F%2Fgit-scm.com%2F&v=MFtsLRphqDM) and download git for your platform(in this case windows)
  + Start installer and follow the installation instruction
* Option 2:
  + Download the “commander” the command line tool for windows from [http://cmder.net/](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqazdwbV9GVGg5UEt4aEw0YXZhV2JLdXhYRlpUd3xBQ3Jtc0treHlrQWhvcC16WFVIWHVsTFd5NmJSZmFmLXRzNl9uR2NpNU9EM1hnd2I2QnBwclZnTDZZLWhNZHVpUGVHS0FlVm1nSmlTdWxwYXZHcm1TdzloUHhVRjJlVTBKWnVNazc3NVRLWHVxX09ieVVXM25haw&q=http%3A%2F%2Fcmder.net%2F&v=MFtsLRphqDM)
  + If you install the full version you it comes with git
* Get text editor like VS-code
* Check if installed
  + On Cmd : git –version
* Set options for the git : email and user





Verify by:

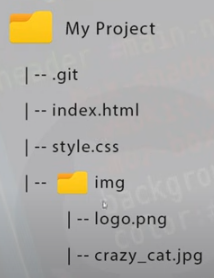




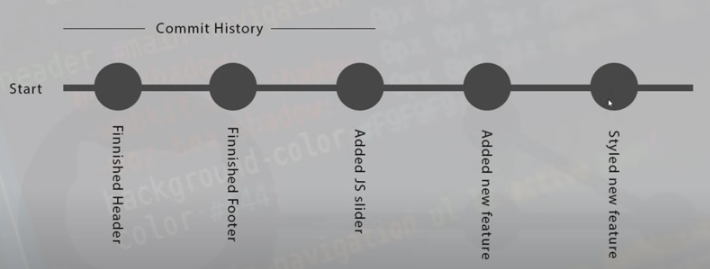
* Common commands:
  + # cd Docuements //changes directory/switch you to the specified directory*(“Documents”)*
  + # cd . . //takes you one directory back/up the directory tree
  + # ls //lists the content of the current directory
    - #dir (or on windows)
  + # mkdir test //creates new folder within current directory
  + # touch file.txt //creates new file with specified name and format within current directory
    - Do not work on windows
  + # code index.html //opens the specified file with code text editor
  + # rm index.html //deletes/removes the specified file
  + # rmdir test //deletes/removes the specified directory

# 3 - How Git Works

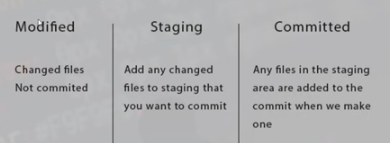
* **Repositories(Repo’s)**
  + A repo is a container for a project you want to track with git (eg. Website project)
  + i.e. a project folder that git is going to track the contents for us
  + contains “.git” folder in the root; which does the tracking on the project’s when there is a change to the files and subfolder inside the project directory.



* **Commits:**
  + Saving points(snapshots) for your projects at any given time, throughout project lifetime



* + **Allows** youto roll back to different earlier save points if needed
* **Stages of Commit**
  + Assume we are making changes to “index.html”, after the change is made for git to track it, we should save/commit it to the “man” branch (which is like Trank of the tree which flows in linear and stores all the project data). To commit you have to modify the file> select/nominate files changed for committing (Staging) then finally commit/save changes to main branch.

****

# 4 - Creating a Repository

* **Creating empty repo:** 
  + Create a project folder “website”
    - #*mkdir website*
  + Switch into the working directory: # cd website
  + Initialize a git repo, so that git can track any changes made in this project folder



* + Create a files and add content to them, for example “index.html” , when you do so git will immediately recognizes the newly created files and changes made; then starts tracking
* **Creating repo on existing repo:**
  + In case you have existing project that already have files and you want to make it a git repo
    - Switch to your project folder



* + - Then initialize the project with git

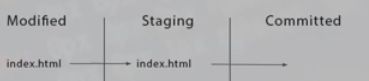


# 5 - Staging files

* Selecting and bringing modified file to staging area that makes it ready for commit.
  + Switch to your project directory: # cd website
  + # git status will show modified/(“untracked”) files in red
  + # git add index.html // adds the index.html file to the staging area
  + # git add . // adds all modified files to the staging area for commit
  + If you run “*git status*” again you will notice all modified files marked in red are changed to green(with “changes to be committed” tag), which means they are stagged
  + If you want to remove/unstage the file from staging area use : *git rm --cached <file>*

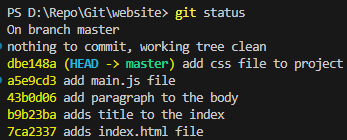
# 6 - Making Commits

* After modifying a file and staging them to stagging area and ready for commit, lets do the commiting



* Commit is save point or snapshot of the project at a given time
* Steps:
  + Run *git status* and make sure all the files are staged(not in red. They are gree) and ready to commit, if they are not, stage them via git add <file> or git add . .
  + Run *git commit -m “descriptive message at that particular saving”* //will commit the staged files
  + To view commit history run #git log which lists all the commit we have done so far
  + Or *git log - -oneline* to get all the commit at one place and summarized

# 7 - Undoing Things

* Undoing/rewinding changes after commit
* **Option 1: Checkout commits**
  + Safe and recommended way,
  + When we use checkout to revisit commit back in time, it is a read only and you **canno**t make modifications(or save or destroy commit history or alter in anyway or shape) while checking out particular snapshot.
  + Steps:
    - List all the commits via #git log - -oneline 
    - Use a commit id generated by git or the commit message you provided with at particular time to rewind to that particular commit.
      * 
      * Notice in the text editor all the changes and commits we made after this particular commit we checkout is gone. i.e. we are basically seeing the code in the state at the commit we checkout.
    - To return/get back to last commit(latest state of the code): reattaches back to original
      * 
      * Notice all the removed changes in previous step is returned, and any modification you made after rewinding with checkout will be lost
* **Option 2: Revert Commit**
  + Safe but can cause problem
  + Allow us to undo particular commit, and delete the commit out of the commit history and creates new commit message at the point we perform the revert. i.e. the affected commit is still in the history and new commit message and id will be added on the branch that marks the revert action; but the commit will undoes our work the commit targets.
  + Steps:
    - List all the commits via #git log - -oneline
    - Lets use revert to remove commit history with Id from the logged
      * # git revert 43b0d06
      * This will prompt you to provide a message for this revert committing, so that you can reference it latter with is message when needed.
    - To get out at this point press “Shift + : “ and type “wq”
      * This will undo the commit from commit history. If you log and view the commit history you will notice the revert is added as commit at the top with the same commit message as reverted commit but with different id.
      * i.e. it makes new commit which reverts undone any work on the targeted commit. So we still access the commit at the top with new revert commit.
* Option3: Reset Commit
  + Unsafe and can potentially destroy content in your repo
  + You can not undone the reset once run and destroyed your work
  + Permanently take us back in time to a particular commit and work from there again.
  + Steps:-Let remove the last three commits from earlier example by resetting to the 2nd commit
    - List all the commits via #git log - -oneline
    - 
      * Everything you done after the targeted commit will all be uncommited from the branch; so you work might be preserved in the editor but it is deleted form commit history and untracked by the git.
    - // the “- -hard” flag will destroy everything after the target commit and do not preserve anything in the editor also.

# 8 - Branches

* When you create a new git repo, by default it creates main/master branch for you. And when you make commits, you are committing to this master branch which represents the stable version of your code which is release or published.
* When you add new features to your code; you don’t directly work on this master branch; not to introduce bugs or mess-up the code.
* So when you add new feature, we create separate isolated environment that contain copy of the main branch to try out this new feature and when you satisfied you merge back this feature in to the master branch via “*merge commit*” which updates the master branch with that new feature.
* If the feature branch doesn’t work the way we expected or everything goes wrong we just delete it so that the main branch remains unaffected with the bug.
* Branches allow different developer to work on same project, but on different features at same time and merge back to the master branch when they are done.

**Creating New branch:**

* To create new branch, follow the following steps:
  + Make sure you are on the master branch witch contains the working code
    - when you run # git branch the prompt marks the master branch with *Astrex(\*)*
  + Create and name the new branch
    - # // creates a branch named “login\_feature”
    - # *git branch -a* // lists all the available branches, including your new branch



//notice the current branch you are on(master) is marked with Astrex(\*)

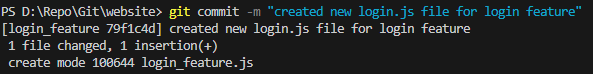
* + Switch to your newly created branch from master branch
    - To be on the branch you just created and work on that feature



* + - To verify you have successfully switched to desired branch run the # *git branch -a* command and your branch should be marked with astrix(\*)



* + Now you are safe to work on your new feature branch which contains the copy of master branch; but it is isolated form master branch
    - Lets create new “login.js” file and add content to it and run # git status , you will notice that this file Is untracked. So, stage then commit this file.



* + - After commit when you run “git status” you are prompted with “*on login\_feature branch*” instead of “*on master branch*” notification



* + - This action doesn’t affect the master branch, to verify this switch back to the master branch and notice the new added JS file is not available or added to the master yet.
      * //switches back to master
      * # *git status* or git log // will show you there is no untracked or committed file on master
      * *But when switch back to the login feature branch you will get all your additions*
* To create new branch and switch into it at same time run the following command
  + **
  + this will create a branchcalled *“check\_in”* and *“-b”* flag allows us to switch into the newly created branch immediate after the branch is created.(i.e. it will shorten the 2 steps into one)

**Deleting branch**

* What if things go wrong and we want to delete this new feature without merging to master branch.
* Steps:
  + Hence you can’t delete the branch you are working on you have to switch to master branch
    - # git branch -a //verify on which branch you are on currently, if you are not on the master branch, switch to it because you can delete the branch you are on .
    - # *git checkout master*
  + Now you are safe to delete the feature branch:
    - 
    - The “-d” flag deletes the feature branch after merging to the master branch, but since we haven’t merged it yet; git will suggest to use upper case “-D” flag instead which will delete the feature branch anyways.



* + - The “ -D” flag deletes the specified branch from the work tree
    - Check if you have deleted the target branch by running the “*git branch -a*”, you will notice the “*login\_feature*” branch is no longer exists in the list of branches.

# 9 - Merging Branches (& conflicts)

**Working on multiple branches at the same time**

* You can create multiple feature branches from the master branch at the same time and work on them separately with out to know about each other or about the master.
* For example let’s create two branches ‘a’ and ‘b’ and add content to them
  + Create feature branch ‘a’ =>***Method one*** first creating the branch and then checking it out.
    - Creates the branch,

// check via “git branch -a”

* + - Switch into the newly created branch:



* + - Create files, stage them and commit them
      * #mkdir feature\_a.js 🡪console.log(“feature\_a:);
      * # git add . 🡺git commit -m “added Js file”
    - 
    - Creates ‘*feature-a’* and switch to the new branch at same time so that we can work on it
  + Create new feature branch ‘b”, =>***method two*** creates & switch into the new branch with one git command
    - Switch back to the master branch, hence we switched to the branch “featue-a”



* + - Create and switch into new branch with one command

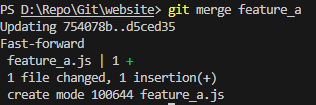
//verify via “git branch -a”

* + - Now add the files and commit changes to this feature branch like we did above
  + *Notice, when we switch between branches, we don’t see the files we created in the other branch. i.e. the content of one branch is not available in the other branch and they are isolated.*
* As we did above we can create multiple feature branch at same time and work on these new features without messing the master branches code.
* When we are done with these new features, we need to merge them back to the master branch so that we can publish them along the earlier codes before the branching.

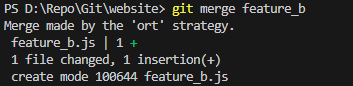
**Merging feature branches into master branch**

**Steps:**

* Switch into the branch you want to merge the feature branch into(in this case the “master”)
  + 
  + Check via “ git branch -a” to make sure you are on the main branch
* Merge the feature branch to the master branch:



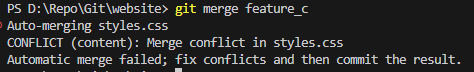
* + Merges the “feature\_a” branch into the master branch. This will fast forward the master branch where we created the feature\_a and commited the file
  + Now all the files in the “feature\_a” are available in the master branch
* Similarly, you can merge “feature\_b” into the master branch to include the files in branch “feature\_b” to the master branch.



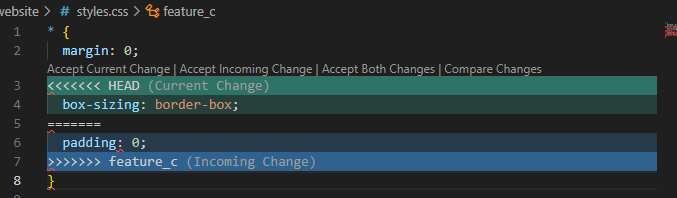
* + However when you do the second merge you are asked to provide merge message.
  + And notice the merge strategy for the first merge is “*Fast-Forward*” and for the second is “**ort**”.
  + The second merge strategy(“ort”) is different from the first(which fast-forward) because the master branch is not the same when the branching for “feature\_b” is made and added back (because earlier the “feature\_a” is added without “feature\_b” know about it)

**Confilicts**

* Imagine you are working separate feature branch and when you are done with your work and at the time you merge your feature into master; some other developer have made changes to the master code you didn’t know about after you make your branch (i.e your feature branch doesn’t include this change). At this time the conflict will arise and get doesn’t know which change to take.
* Demo:
  + You create the feature branch “c” and make changes to the styles.css file, stage and commit your feature branch.
  + Then switch to the master branch and make changes to the styles.css file, and commit it
  + Then merge the feature branch “c” back to the master, you will be prompted with a lot of messages,



* + - so to fix this do the following:



* + - * if you are working as a team talk with other developer and decide how the code should look and commit the change. So let us assume we agreed to keep all the changes made by both developer, so to accept the commit remove the comments on the line #3,5 and 7 and save the changes.
      * Then stage(add) the file and commit the agreed change with out commit message >> *shift +:* 🡺 type “wq” and press enter
      * Now check your commits with “ *git log - -oneline*”

# 10 - Introduction to GitHub

* ***Github*** is a service that lets us setup hosted repositories or known as remote repositories
* Is central online repository which multiple team-members could access , and clone copy of that repo on their own local computers work on their own features and push back to that remote repo when needed to update the code and merge it to the master branch on the remote repo.



* Eases the collaboration and working together effectivily.

**How to use github:**

* Create github account on the github website <https://github.com/>
* Then sign in
* Creating repo:
  + ***Option 1: if we already have local repository setup for our project***
    - Steps:
      * On github click the “new” button and give a name for that repo that matches your local repo name. say “*resume\_builder*”
      * Add description about your project, choose whether your repo will be available to *public* or *private*(paid option) , you can initialize the repo with *readMe* file, you can also include “.*gitignore*” file and also choose a license too.
      * After making your choices click the “***create repository***” button to create remote repository for your project on the github.
      * On the next page From the button choose one of the url path to the repository, we have 3 options(the “*https*”, “ssh” or “*Github CLI*”). for this example, copy the “https” option
      * Go to your local repository, then push the content of the local repository to the githubs remote repo.



* + - * + Check if the local repo is a git repository and everything is up to date by running “git status “command.

If it is not git repo run “git init -b main” in the repos root folder which initializes it with git. Hence in earlier versions of git when we initialize git repo with “git init” the main branch is named as “master” ; so we used the “-b main” flag to rename the master branch to “main”, to sync with github.

If you’re using Git 2.28.0 or a later version, you can set the name of the default branch using -b.



If you’re using Git 2.27.1 or an earlier version, you can set the name of the default branch using *git symbolic-ref*.



Also if you have untracked files when you run “git status”; stage and commit them.

* + - * + Push this local repo up to the “main” branch of remote repo using the “https” URL of your project on the github, (you can also push feature branches if you wish, with similar command except you should change the branch name from “main” to “<feature\_branch\_name>”)



On this step You might be asked to get authenticated by the github, and if everything goes right you will get feedback messages confirming the push actions and on which branch you have pushed(here it is master).

* + - * To verify the action, go to the github website and click on the entry of your project name under “Repositories” tab and you should see all your files pushed up there.
      * To push consequent changes in your local repo to the remote repo you will have to run after every modification you make, however, instead of typing(or copy-pasting) the remote https URL every time, you can create alias for this repo and use that alias whenever you want to push/fetch changes.
      * So to create alias; run



This adds(gives) remote repos URL the alias name called “origin”(*can be any name*). So now instead of typing the whole URL when we run push/fetch command, you can simply run the following command which is simple, short and concise:



Notice, the entire URL is replaced with the alias “origin”

* + ***Option 2: we don’t have no local repo for the project yet***
    - we create repo on the github and clone that repo in our local computer
    - Steps:
      * On github go to “your Repositories” > New> give repository name for your project and set other option you want to setup like descriptions, public or private, readMe file, etc… >> click “***Create Repository***” button.
      * Then clone this repository to your local computer:
        + From “<>Code” button copy the URL and go to your local directory where you want to clone this code to and run;



This will create a local git repo; which is copy of the remote

* + - * + Switch into the cloned empty project folder and start working:

# *cd project1*

* + - * Hence, we cloned it from remote repo, we don’t have to create alias for this repo, the origin alias is already setup for us by default, you can verify this by running  which state “origin” alias is set for both fetch and push operations.
      * So if we want to push up to the remote, you can just simply run:



//note the master branch is renamed to “main” in latest git hub versions

# 11 - Collaborating on GitHub

* So far, we have a local copy of our project on our computer and remote repo on the github, and we can push changes made on local repo to the remote, and also, we can pull updated code from remote repo when needed.
* Now let’s simulate how multiple developers collaborate via github;
* ***Steps***:
  + Always make pull request to make sure you have the updated version of the project in your local repo, just in case any other developer make changes.
    - Go to your project folder on your local computer open it in editor
    - Switch to master branch : # *git checkout main*
    - Then make a pull request,



This fetches all of the code from the remote main branch and then merges it to our local main branch and updates all the files to catch up with changes other people’s made. So that we are up to date with remote repo.

* + Create, switch into(checkout) and work on the new feature branch, so that you can preserve the state of the “main” branch. we don’t want to modify main branch, because we may mess the working codebase.



* + - Make sure you are on the feature branch before you start adding more code



* + - Then after making all the changes on your feature branch; stage it (# git add . ) then commit it (# git commit -m ”creates navbar ”)
  + Push the feature branch you are working on to the remote repo
    - If we merge our feature branch to main branch locally and push it to the remote repo, the local main branch will override the main branch on the remote repo.
    - So it is recommended (especially when you work as a team) to first push your feature branch as it is to remote and then compare the difference and then make a pull request to merge to main branch on the remote repo.
    - Push your feature branch to remote repo



This push up this feature branch to remote github repo

* + Go to the remote github repo and click the “( added when new features is pushed upto github), to compare the differences between these changes we make on our feature branch and the current state of the main branch and then make a pull request(merge it into the master branch) on this repository.
    - You can add explicit message for the changes you made on the next page after clicking the “*compare pull request*” button. Then click the “create pull request” button to merge feature branch into the main branch.
    - You can assign reviewers on the next page, This will send notification/ping to all collaborators.
    - If you see “***This branch has no conflicts with the base branch***” ,then you are safe to merge the pull request by clicking the “Merge pull request” button
      * You can add comment on this step also
      * then confirm merge by clicking “Confirm merge”, So that it will merge the feature branch to the master.
      * You will add another comment for the confirmation
    - Finally, you can see your changes under “Files changed” tab, and you can add comment line by line to the changed files. And also notice all the code in the feature branch ins added to the main branch.
    - Now you are safe to delete the feature branch, hence it is already merged to the main.
* Then on the subsequent work you repeat this steps all over again:
  + Pull the main branch to access the latest code and changes made by other developers.
    - #git checkout main
    - #git pull origin main
  + Create new feature branch, switch into it and work the feature:
    - 
  + When you are done, stage and commit your work( git add . , git commit -m “msg”)
  + don’t merge to local main branch b/c other also have to comment on it
  + push the feature branch to remote



* + go to github, click the “*compare & pull request*” for this feature branch add comments about the changes, “*create pull request*” >>assign reviewers, compare the difference with main branch content, under the “Files changed” tab and make comments if needed by pressing the “+” signs on the code line>> go back to the conversation >> and add comments about the change and this comment >>comment(btn)
    - go back to your local files add content(say image icons) >stage and commit them on the feature branch and push it to remote to notice the changes:
    - 
  + Go back to github and notice this change is propagated to remote repo on the same feature branch(you can check the changes under “files changed” tab
  + Then when you want the merge to happen go back to the “conversation” tab, add comment to identify this changes and then click “Merge pull request” >>”confirm merge”
    - This merge the change made locally to remote and all changes are added to remote
    - You can delete the branch finally if you want to. (not recommended, you can refer it in the feature)

<https://www.youtube.com/watch?v=MnUd31TvBoU>

# 12 - Forking (& Contributing)

* A way to fork a repo on the github to contribute to the open-source projects.
* You download a repo, improve it or make changes to it and push the changes back to remote repo. However when we try to push, we run into problem because we don’t have access rights.
* Forking creates a copy of a repository to our computer.
* Forkin steps:
  + Got to the repo you want to fork on github and click the fork button (forExample: [https://github.com/iamshaunjp/modern-javascript/](https://github.com/iamshaunjp/modern-javascript/fork))
  + Click the fork tab> when you do the fork will open news repositories in your github account, by default the repository will be same to the repo you forked from >> click “Create fork”
  + Clone the repo to your computer
    - Copy the url from your repo for the forked project and clone it on your pc



* + Switch in to it and make your contributions
    - Create feature branch for your work, make change stage it and commit it locally, you can also merge it to main if you want to
  + Push the branch to the remote repo of your account(not the account from where you fork)
    - #git push origin <branch name :main/feature”
  + Push your contribution to the original repo from which we forked
    - Go to your github account and the repo you have forked you forked to your account.
    - click the “New pull request” > then “create pull request” which pull into the main branch of the other account we forked from( the open source repo) > add comment about the contribution you have made and click “click pull request “ button again
    - Notice, you can not do the merging in the open-source repo, because you don’t own that account/repo. So the owner have to review and allow as to merge or merge changes himself.
  + The owner of the “open-source” account will log in is account and take the following action to merge the changes/contribution you made to the original repo.
    - Go to “pull request” tab and view the contribution from you, and if he likes it he add the comment and then clicks “Merge the pull request “ button to accept the changes to the original project > “confirm merge” (btn)