Git & GitHub Tutorial | Visualized Git Course for Beginner & Professional Developers in 2024

72 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=S7XPTANSDL4
https://www.youtube.com/watch?v=S7XpTAnSDL4

SUMMARY

This crash course on Git and GitHub provides a comprehensive overview of essential version control concepts and practical skills necessary for developers. It emphasizes the importance of Git in managing code changes, collaborating with teams, and resolving issues effectively, ultimately preparing developers to handle real-world coding challenges.

- *Git is a distributed version control system that tracks code changes and allows multiple developers to collaborate seamlessly.*
- *Key commands include `git add`, `git commit`, `git push`, `git pull`, and `git merge`, which facilitate tracking changes, committing updates, and synchronizing with remote repositories.*
- *Branching allows developers to work on features independently without affecting the main codebase, and merging integrates those changes back into the main branch.*
- *Merge conflicts occur when multiple developers edit the same lines of code, requiring manual resolution to determine which changes to keep.*
- *Advanced Git commands such as `git reset`, `git revert`, and `git stash` help manage mistakes, revert changes, and temporarily save work in progress.*
- *Using a GUI like WebStorm simplifies Git operations, making it easier to perform complex tasks without memorizing commands.*
- *The course includes a reference guide for quick command lookup and encourages practice to build confidence in using Git effectively.*

imagine you're working on a coding project and you make a mistake that breaks everything your boss would most likely fire you if you were even able to get a job in the first place without git you'd have no easy way to go back and undo the changes you're toasted git is the industry standard most companies team and open-source projects use git so naturally every job description mentions it learning it isn't just a nice to have it's your get good or get out moment

it's a must for any serious developer wanting to land the job so hi there and welcome to a quick non-nonsense git and GitHub crash course unlike a typical git tutorial which only scratches the surface and leaves you pushing straight to production like many other interns we go beyond the basics and dive into the real stuff no one really talks about like how to fix a broken production on a Friday by the end of this course you'll

not only know how to track code changes and collaborate with your team but also professionally resolve merge conflicts fix real life production issues Write Clean commits so your team doesn't have to question your life choices recover from major mistakes with reset revert and checkout so when something breaks production and

it will you'll know exactly how to fix it and save the day use git through a GUI so you don't have to memorize tons of commands and even

Master Advanced G tricks like cherry-picking and stashing because you're here to get good and not just get by basically with this crash course you'll become your company's goto get guy the person everyone turns to when things go south and there's a lot we'll cover in this video and I don't expect you to return to it anytime you need to refer to a command for that reason I've put together the ultimate git reference guide packed with Advanced tips and

tricks real world use cases and commands to help you level up your git skills you can download it for free by clicking the link in the description and use it as a companion to this course and your career let's Dive Right In so what is git and why is it so popular git is a distributed Version Control System sounds fancy right well let's break it down the Version Control part helps you track and manage code changes over time

while distributed means that every developer's computer has a complete copy of the code base including its entire history of changes and information about who changed what and when allowing you to get blame someone hopefully people won't blame you but do you really need it can you code without using it well of course you can but then your workflow would look something like this you start coding your project in a folder named my project and as you make progress you

worry about losing your work so you create copies my project V1 V2 V3 and so on then your colleague asks you for the latest version use zip up my project V3 and email it over they made some changes and sent it back as my project V3 Johns changes. zip meanwhile you've continued to work so now you have my project V4 you then need to manually compare John's changes with your V4 and create a V5 incorporating everyone's work and then a

week later you realize you accidentally removed a crucial feature in V2 you dig through your old folders trying to figure out what changed between versions now imagine doing this with 10 developers each working on different features it's a recipe for chaos lost work and countless hours wasted on version management system instead of actual coding git solves all of these problems and more it tracks every change automatically allows multiple people to work on the same project seamlessly and

lets you easily navigate through your Project's history no more final version V2 final really final zip files git does all of this for you but in a much more powerful and organized way to get started you need git installed whether on Windows Mac or Linux it's just two clicks away Google download get and get it for your operating system once get is installed open up your terminal nowadays I prefer using a terminal built into my IDE I'm using webstorm and honestly I

prefer it over vs code because everything just works seamlessly and webstorm get support is extraordinary you can do so many things you'll learn about in this video everything from creating branches committing pulling changes merging and even resolving conflicts all without ever leaving the IDE you can view your G history stash

changes and even review pull requests directly inside webstorm it makes working with Git smooth and intuitive after walking you through the essential git commands in the terminal I'll show

you how we can perform the same tasks more intuitively and efficiently using webstorm if you're interested in trying it out I've included a link in the description where you can download webstorm for free just create an account click download and go through the onboarding once it's set up create a new project and you're ready to start coding with all of these powerful git features built in first things first let's check whether you've installed git properly run git D- version and you'll get back

the version that is installed on your device next you need to configure G to work with your name and email this is just to track who made the changes in the project so your colleagues know who to blame here's the command get config -- Global user.name and then in single quotes put in your name once you do that you can repeat the same command but this time instead of changing user.name will change user. email and here you can

enter your email press enter and that's it you're all set up now let's talk about repositories a repo or a repository is where G tracks everything in your project think of it like a folder that stores all the versions of your code simply put if a folder is being tracked by git we call it a repo now let's create a new repository in your terminal type get in it and press enter as you can see git has just

initialized in your repository on top of the success message we can also see a warning in previous times the default name of a branch has been Master but nowadays you'll see main used much more frequently as the name for the primary Branch so let's immediately fix it by configuring the initial Branch name you can copy this Command right here and at the end you can just say main now considering that we have just changed the initial configuration

settings we have to create a new folder create a new one called something like mastering git open it within your editor and then rerun git in it as you can see here and here now we're in the main branch that means that git has initialized an empty repository you won't see any changes yet in your folder but a hidden dogit folder has been created inside your directory you don't need to touch this folder git handles everything inside from commit history

branches you'll make remote repos and more most of the time git will already become pre-initialized by the framework or library that you use to set up your project with that's how integrated git is into every developer's life so now that we have this main right here what does that exactly mean well Main is the default Branch name of your repo created by git every time you initialize git this Branch will be automatically created for you I'll teach you more

about get branches soon but for now know that a branch is nothing but a parallel version of your project all right let's add some files and track changes I'll create a new file called hello.js and you can see how smart webstorm is it automatically asks me whether I want to add it to get but for now I'll cancel that because I want to explain everything manually let's make it simply run a console.log that prints hello get alongside this file let's

create another new file and I'll call it readme.md in here we can do something similar and say hello get and now run git status git will tell you that you're currently on the main branch that there are no commits yet and that there are two untracked files one of which is a markdown document so to track use get add readme.md after adding a file we need to commit it committing in git is like taking a

snapshot of your project at a certain point think of it as creating a whole new copy of your folder and telling git to remember when you did it at what time so in the future if anything happens you'll time travel to this folder with the commit name you specify to get and see what you had in there it's essential to commit your changes regularly Reg regular commits help you keep track of your progress and make it easier to

revert to previous versions if you break something you can commit by running git commit dasm which stands for message and then in single quoted strings you can add that message for example add readme.md file there we go congrats you just created a checkpoint in your Project's history now let's try running git status again to see what it shows as you can notice that other file hello.js is still there it's not tracked we asked git to

track only the readme file to track this file or other files that you may create will have to run a similar command it'd be too much work to commit each file individually thankfully we have a command that commits all the files we've created or modified that git is not tracking yet to see this in action let's create another file test.js and let's add a simple conso log that's simply cons logs a string of test now to track both files and commit them

in a single commit action we can do that by running git add dot the dot after git add tells git to add all files created modified or deleted to the git tracking next as usual we can specify the commit name for this tracked version by using git commit DM add hello and test files there we go so now now we can see that all of these files are tracked and since I'm using webstorm it also has a

hidden Ida folder so it added it to tracking as well which I'm okay with well done now to see the history of all commits we've created we can use a new command git log and there we have it our git history it contains a commit ID or a hash automatically created by git the author we specified when using it config a Tim stamp and the commit message we provided great but how do we switch to an older commit and restore it let's say

the commit add hello and test files introduces some buggy code and we want to restore our project to a previous version without these files our brain would immediately suggest deleting those files entirely or clearing up their code and if you do that you'll most likely break your production because other files depend on those files so instead of deleting them manually to restore to the first version where we had only committed the readme file we can use a

new command first you have to copy the commit hash yours is going to be different from mine so make sure to copy yours I'll get this one first that says add readme MD file and I'll press copy then you have to exit this git log by pressing the Q letter on the keyboard and then you can use a command get checkout and then you can provide a hash of a specific commit or a branch you want to check out to now press enter

okay something happened first of all our two files are gone detached head experimental changes what's happening well in git there is a concept of a head which refers to the pointer pointing to the latest commit you've created when we created our second commit our head shifted from read me commit to the latest add hello and test files commit but when we ran get checkout command we moved the head to the previous older commit that's why we got this detached

head warning it's a state where the head pointer no longer points to the latest Branch commit and the rest of this message tells you that you can create a new Branch off of this commit but don't worry your files are still somewhere when you use a git checkout command You're simply viewing the repository State as it was at the time of a specific commit like right now we're viewing a snapshot of your codebase at a previous moment in time when we only had

a readme.md file the beauty of this is that all the logs and files whether created or modified remain untouched the get checkout command won't delete any logs or history so you can safely explore past states without worry but what if you actually want to discard changes made after that commit maybe you want to quickly roll back to a stable state after an issue hits production tidy up messy commits to look more professional or undo a bad push you

regret making perhaps you've been experimenting with a refactor that didn't pan out or you need to recover from a messy merge conflict thankfully git provides a few commands that will help you in these scenarios and I'll teach you how all of that works very soon so just keep watching and we'll dive into these more advanced commands that are really going to help you well fix a broken production now to go back to our current state which is often

called the head State you simply have to run get checkout Main and there we go previous head position was at the hash of this checkout and now you switch the branch to main you can see the same thing happen right here on the bottom right or the top left depending where you're branching is and if you made any changes while in the detached head State and you want to discard them you can do the same thing

get checkout dasf that means force and then get back to main in this case we're good we're already on Main and that's it you already know more about G than most developers do of course we'll dive deeper into advanced use cases and tips and tricks soon but now let's talk about GitHub and how it differs from git git is a tool you use to track changes whereas GitHub is a cloud platform that allows you to store your git

repositories online and collaborate with others to push your local project to GitHub you'll need to link your repository to a remote but what's a remote well there are two types of repositories local repository is a version of a project that exists on your own machine laptop or whatever else you use where you do your developer work when you initialize a repo using get in it you create a local repo in your folder changes you make there are

private until you push them to a remote repository so a remote repo is a version of a project stored on a server like GitHub gitlab or bid bucket it's used to share code between collaborators and keep project versions in sync

across different users computers when collaborating with a team you'll have two kinds of repost everyone in the team will have a local repository on their machine and there will also be this one common remote repo from which everyone

will sync their local repository versions now head over to github.com and create an account if you don't already have one once you're in press the little plus icon on the top right and select new repository enter a repository name such as mastering git choose whether you want to make it public or private leave the add readme file checkbox unticked and click create repository this is a remote repository

here you can see your repository's origin copy it when you clone a repository from GitHub git automatically names the remote repository as origin by default it's basically an alias for the remote repositories URL now our goal is to link our local repository to the remote origin if you haven't yet switched the default Master Branch name to main you can do that by running git Branch DM Main and this will change the branch name to main which is a standard

practice nowadays and now we are ready to link our local repo to a remote origin you have to run a command get remote add origin and then you have to paste the link to the origin that you just copied and press enter and a good thing to know is that you can have multiple remote repositories you just have to rearend the command and change the origin name to something else of course that's the name of your choice

and then you can also update the new URL but in most cases you'll be fine with just one remote repo finally to push your local commits to GitHub use git push Das origin Main and remember we used origin here to refer to the remote repository instead of typing the full URL so press enter and there we go this worked if anything with Git goes wrong typically it goes wrong at this point when you're trying to push to a remote

repo so if you don't see what I'm seeing right here and instead you got some error typically all of these errors are very easily resolvable I would just recommend copying the error message pasting it in Google and then fixing it right there and then but in this case we're good and now if you go back to your GitHub repository and reload boom your code is now online for the world or your team to see and okay okay you might

have already known this for some of you that's about as far as you've gone with Git create a repo push your changes and call it a day but git has so much more to offer especially when you're working within a team so now let's take things up a notch and dive into branching and merging this is where get truly shines branches in git allow you to create different versions of your project like making a copy of a project at a specific

moment in time whatever changes you make in this copied version won't affect the original the main project or Branch stays untouched while you experiment modify or add new features in the copied Branch if everything works out you can later merge your changes back into the original project if not no no worries the original remains safe and unchanged when working in a team using separate branches for different features or bug

fixes is essential it allows you and your team to work independently on different parts of the code without causing conflicts or errors ensuring everyone can focus on their own tasks at the start you'll have one default branch called main to create a new Branch run G branch and then type A Branch name this will create a new branch and if you want to switch to this newly created Branch then run get checkout and

then enter the branch name you want to check out to and there we go switch to Branch Branch name now if you want to go back to main just run get checkout main there we go and here's a little Pro tip there is a shortcut to create a new branch and immediately move to it to do that run git checkout with a dasb flag and then enter a branch name such as feature branch of course this Branch

name and feature Branch are just dummy names make sure your branch name is short and explains which changes will you be making on that Branch for more tips on how to properly name your branches you can download the git reference guide so let's create and move to this feature branch in one command there we go and what I'm about to say now next is very important so keep it in mind when you create a new Branch it'll

be based on the branch you're currently on so if you're on the main branch and run the command the new Branch will contain the code from the main branch at that point in time however if you're in a different branch with different code the new Branch will inherit that code instead so to ensure you're creating the new Branch from the correct starting point you you should either first switch to the branch you want to base the new

one on or run this command get Branch then you can enter a new Branch name and then the next thing can be the source Branch so if you do it this way and replace the new Branch name and the source Branch with the names of actual branches then it'll create a new Branch from another specific Branch so if you run this command you can directly create and switch to a Branch based on any other Branch without needing to check

out to it first for now I'll remove that and let's say that we want to go into our code and implement this feature we're working on let's say that in our case the only feature we want to do is to modify the read me so below hello G I'll say I'm adding this from feature Das Branch there we go feature implemented if only it was this easy and you can see that our IDE in immediately highlighted this read me file in blue

indicating that it has some changes now we need to add it commit it and push it this time instead of saying git add readme MD let's just use the git add dot which is a command that you'll use much more often next we need to commit the changes with Git commit dasm and then we have to add a commit message so this is the perfect time to learn how to write a proper commit message a quality commit

message is written in the imperative mood a grammatical mood that sounds like you're giving a command like improve mobile responsiveness or add AB testing when writing your commit message make it answer this question if apply to the codebase this commit will and then fill in the blank like this commit will upgrade the packages or this commit will fix thread

location and why do we do this well because it answers the question what will happen when I merge the branch containing this commit it will add AB testing for example be direct and eliminate filler words for example let's use modify read me in this case it's short sweet and in an imperative mood and press enter there we go we've just made get aware of our commit now that you know how to write better commits

let's take a moment and check out our remote repository what do you think will it have the latest commit we made Let's reload it and it's the same it doesn't contain our newly created feature branch do you know why it's because the changes we made are in the local repository which has not yet been synced with the remote repo to see those changes first you'll have to publish your local branch and you can do that using the git push

-- set- Upstream origin and then the name of the branch in this case feature Dash branch and press enter there we go an upstream branch is a remote branch that your local branch tracks when you set an upstream Branch using set Upstream you're essentially linking your local branch to a branch on a remote repo through this command you push a local feature Branch to the or origin remote repository and then you

set the Upstream Branch for your local feature Branch to track origin feature Branch alternatively you can also use git push dasu origin feature Das Branch or the name of your branch of course both set Upstream and- you establish a tracking relationship between your local branch and the remote Branch this way in the future if you want to push something from your local branch to your remote

Branch you simply have to run get push that's it at this moment for us everything is up toate but as you make future changes you don't have to rerun set upstream or dasu you only have to add it commit it and push it that's it and if somebody else made changes to your remote Branch either directly or by merging some other changes into it you have to make your local branch up toate with the remote branch and you do that

by using the git pool command there we go it's already up to date in this case this command fetches changes from the remote repo and merges them into your local repo for that Branch there are also two more advanced and highly practical commands that help you move changes between your remote and local repos and I'll explain those in detail later on so keep watching now if you go back to GitHub check this out feature Branch had recent pushes a few minutes

ago and if you go to branches you can see your feature Branch right here so if you click on it you can see the state of your code base on that branch and you can see the modify read me commit that we made but back on the main branch it hasn't been modified it simply says hello get still so once you have made all of the changes for this branch and implemented the feature you wanted to

add and tested it you'll want to merge it back into the main branch to do that you must open up a pull request a pull request lets you share your changes with your team for review and feedback once approved and merged your changes become a part of the main branch keeping the codebase stable and organized to open up a PLL request you can click this button right here compare and pull request or you can click this pull request tab

right here select new P request request select the branch you want to merge from such as the feature branch and then the branch you want to merge to here you can see all the changes you implemented in this case we have added one line of text if the changes look good you can click create P request if needed modify the title and describe what your PR will do in more detail and then select create one final time there we go now your team

lead or senior Dev can review your PR and give you feedback and if everything looks good and there are no merge conflicts they can merge it in this case reload the browser and let's merge it ourselves yes we're sure confirm merge this is where the magic happens your feature will now seamlessly get merged into the main branch easy right so if we go back to code you can see that we're now on the main branch and we can see

the change from the future Branch so why does it still say that we have two branches well no worries once you merge a PR from a specific Branch GitHub and git still leave it here for you but they tell you that you're one commit behind Main and zero commits ahead which means that everything this Branch was supposed to do in this single commit has been done and it has no additional code that you might want to merge so it's safe to

say that we can delete it now if you go back to code you can see that there is only one branch that's the main branch and and it contains the newly merged changes easy right this is how teams collaborate without breaking each other's code by doing this GitHub is essentially executing a git operation in the background a command that you can do on your own by running git merge and then choose a branch that you want to

merge but I typically prefer doing this using github's BRS now if we switch back to our local repo and navigate to the main branch do you think we'll be able to see the changes we just merged well let's try do you remember the command to switch back to main well it's get checkout and then main there we go we switch to Main and it says it's up to date with origin main so if we had to read me wait this

doesn't look good it doesn't contain this new line that we added right here and that's because the merged happened on the remote repository which makes sense right so how do we bring those changes into our local repo simple we need to pull them into this Branch by running get pull there we go it pulled changes from a remote repo and we can see them right here in our local one the command we just ran get pull is just a shorthand

for git pool origin main but by default GitHub pools from the remote origin from the same Branch you're currently on so just get pool is fine so let's repeat things here is the typical workflow you'll follow most of the time first clone the repo second create a new Branch from the main or another Branch third make your changes fourth push the branch to the remote repository sixth open up a pull request seventh merge the

changes eighth pull the merge changes into your local main branch and then repeat from step two you can also find these steps alongside some additional explanations within the GID reference guide so great job learning about branching I hope this was intuitive but next we'll focus on a scary topic which is resolving merge conflicts something that not a lot of people teach you but you and I will dive deep into it sometimes when two or more developers

edit the same lines of code git gets confused this is called a merge conflict when this happens G will ask you to manually choose which or whose changes to keep but why don't the merge conflicts happen when you're working on your project alone a merge conflict is a situation where git is uncertain about which part to merge but why would get be unsure to show you what I mean in practice let me create two branches

first I'll create a branch called Dev JSM so let me run git Branch Dev D JSM and PR press enter this will only create a branch but not switch to it now let's immediately create and switch to another Branch we can do that with a git checkout dasb and then the branch name command I'll call this one Dev Adrien and you can of course use your own name and press enter now I'll

make some changes to the codebase from the dev Adrien Branch specifically I'll go to read me and I'll change hello git to welcome to git I'll also add another line and say this is coming from Dev Adrien like this now let's add the modified file to git by using git add dot we can write a meaningful commit message by doing git commit DM and let's try to be descriptive of what we did

it's always easier to track the changes that way so I'll say modify read me by changing The Heading and adding a new line press enter and at this point everything is looking good but the branch is of course not yet visible on GitHub since it only exists on our local system to publish the branch to the remote run git push dasu origin and then the name of the branch Dev Adrien and press enter the U

says the branch is an upstream branch and the origin as you know is the name of the remote repo if you go back to to GitHub you can see that Dev Adrian has recent changes so let's open up a port request there we go modify read me by changing the heading and adding a new line and create a new P request great the P request is there now for just a second imagine you're not yourself you

are your friend so let's actually move to your friend's Branch by running git checkout and we called it Dev JSM remember this Branch was created off of the main branch at the time when other you didn't yet add those additional changes so let's say that your friend also wants to modify the read me instead of saying hello git they're going to say something like heyo everyone welcome to my git guide and

they'll add an additional line and say yo Johnny here and they're going to commit their changes by running git ad Dot get commit dasm and of course they don't know how to write quality commit messages as you do so they're going to write something like today I woke up and drank some coffee then I sat at the table and added a few lines of code super long but doesn't really tell us anything right of course we have to push

that Branch by running git push dasu origin de- JSM or whatever name of the branch you ch chose and now our friend is going to come to GitHub and they'll want to open up their pquest so let's do that right away their title is so descriptive it doesn't even fit in one line and now we have two PRS currently opened and even though you opened up your PR first life sometimes isn't fair so let's say that whoever is supposed to

review this PR just didn't want to even see the code that this guy wrote so they immediately merged it the key takeaway here is that they merged it before yours got merged that's great for them but now if you go to your PR and try to merge it you'll see something like this this branch has conflicts that must be resolved merging is blocked now it is highly likely that whoever is reviewing this won't even start reviewing it until

you resolve the conflicts so they'll most likely write something like Adrien please fix these merge conflicts so what just happened why do you have a merge conflict and most importantly how can you resolve it the merge conflict happened because both Dev Adrien and Dev JSM made changes to the same file specifically to the same line in the same readme.md file when the reviewer approved and merged the dev JSM Branch

it created a new commit that made changes to the same line of code that Dev Adrien has already modified therefore when you try try to merge your branch with the main branch git could not automatically resolve the conflict and required you to resolve it manually in other words git was just not sure which version of the code to merge since two different versions existed on two different branches when git encounters such a conflict it expects you as the

user to resolve the differences and decide which version to keep and then manually update the code to reflect the changes made in both branches this is a typical scenario that happens when both you and your teammate are working on the same code simultanously on the same code at the same time resulting in modifications to the same line of code and finally causing a merge conflict it is something that no matter how much you try to avoid it always happens so you're

likely to encounter those merge conflicts from time to time so how can you resolve them well there is a standard process that you need to follow with enough practice it'll become second nature here is what you have to do first check out to the main branch by running git checkout main next pull the latest changes from the remote main branch or in other words the changes that your friend got merged Before You by running git pool there we go we can see his

stupid comments right here now your local main branch and remote main branch are identical so you can safely check out to your own Branch get checkout Dev Adrien stop here and let me ask you a question when you created a p request which branch were you attempting to merge the changes into well of course the main branch so what are we trying to accomplish here if you're not sure let me show it to you in action run the

command git merge main that's correct you have just merged the main branch into your branch although the initial objective was opposite it was to merge your branch into the main branch but that process didn't work due to the merge conflict so we first must resolve the issue and to do that we are merging the main branch into our Branch to identify the problem the command we just used get merge and then the branch name

is used to merge that Branch name that you specified right here such as Main in our case into the branch we're currently on and now that you've run this command you can see see that there merge conflicts in readme.md automatic merge failed fix conflicts and then commit the results most code editors have some kind of a

versioning system right here on the left sidebar by pressing command zero I can open up web storms one here you'll be

able to see a list of all merage conflicts in this case we have only one and it is in the readme.md file let me show you how to read it arrows pointing to the left side that say head refer to the changes coming from your branch and arrows pointing to the right side that say main refers to the changes coming from the main branch due to the merge command that you executed to resolve them you can manually choose

what you want to keep or remove by removing those lines and clearing up the code that you don't want there but the usual way is to click the resolve button on webstorm it looks like this but the options should be similar across all code editors typically you have three options you can choose whether you want to accept your code whether you want to accept their code or whether you want to do something in between of course

wouldn't life be so simple if you could just choose one of those two options but typically that's not what you want to do in most cases you'll want to keep some of your code and some of their code so we'll have to go with the third option this interface will differ depending on the code editor you're using sometimes it might be in line so you see the red and green lines there irly within a single file but I very much like the way

in which webst is doing this on the left side we can see our changes on the right side we can see their changes and in the middle we can see the result so you can just choose which one you want to merge in this case they wrote something that's not so useful so we want to keep ours and now that gets transferred over to the middle one welcome to get but let's say that this code is an important

implementation of some feature within the their a branch of course in this case it's just a line of text but I think you can imagine how Johnny can really write something useful so let's say that we also need to merge this and at the same time we have also added an important feature from our own Branch so we can add it right here as well so the final result will look something like this modified heading Johnny's feature

your feature and then the feature that was previously there let's remove this stop change from Johnny because we preferred ours better all changes have been finished and we can click apply and there we go here is our new readme file containing both Johnny's changes and our changes without conflicts now let's open up the terminal run G add dot to add our newly modified file say git commit DM resolve merge

conflicts and run git push you already know the drill right now return to GitHub and check the pr if you scroll down you'll see that there's no more merge conflicts because they've been resolved by a resolve merge conflict commit now it would make sense to tag your reviewer and say something like merge conflicts resolved please check then they'll be able to go through a code base request some changes if necessary

and finally merge your pull requests to the main branch now if you go back to the code you can see that there's here's a new commit to the read me 3 minutes ago and our heading has now been cleaned up it doesn't include

Johnny's nonsense anymore but it does include their features it includes your features and it includes features previously added to main amazing work now you know how to resolve merge conflicts one of the scariest things to do as a developer and

I Know It All looks intimidating at first but with enough practice it'll become second nature trust me so far you've learned the basics and some intermediate git skills that you'll use in your day-to-day work but let's be honest sometimes things can go wrong and you mistakenly break production that's when git becomes your best friend only if you know how to use it to its full potential of course and this is where what I call git savior commands come in

handy these commands let you manage changes undo mistakes and tweak your commit history think of them as your backup plan when things get messy you already tried and tested one such command which allows you to check out to a past commit do you know which one is it yeah it's GI checkout and then a commit hash but as I've told you that command doesn't discard or delete anything it's just a way to view the history of specific commits there might

be the case where you want to check out to a particular commit and then delete everything that comes after it it that's the job for get reset imagine you want to remove some commits and revert to a previous commit with the possibility of choosing whether you want to keep or discard the changes in the working directory that's a bit hard to understand right so let me say that again suppose you made 10 commits and want to check out the third commit in

history you want to delete all commits after the third like fourth fifth 6th 10th and so on maybe you even wrote some bad code in those commits and to keep things simple you want to remove these bad commits but still want to keep the changes that you implemented I hope that makes sense so delete these commits but keep all the changes you made from the fourth to the 10th Commit This is where we use git reset it allows us to remove

the traces of commits in history but gives us the changes we made in those 2B discarded commits so we can decide what to do we can keep those changes form a better commit out of them or delete them entirely it's up to you let me quickly add a consal log to this hello.js file on top of the first one I'll add another one that says something like consol log hello GitHub there we go now we have two

then make a commit by running git add dot get commit DM add GitHub conso log to hello.js s next to properly illustrate how we'll use git reset I'll add one more conso log this conso log will say something like hello from and then put the name of your branch Dev Adrian in my case since we haven't switched from before we're still in this branch and I also want to commit this

imagine this is a bigger feature that you implemented so I'll run git add dot git commit dasm and we'll say something something like add hello console.log to hello.js and committed now you committed you stand up from your desk go for a walk or go about your day then you come back and then you start focusing on some additional functionalities within the same Branch let's say that additional functionality is a another console log

but this time this console log will actually break things so we can say this is bad code okay and now we can do another commit of course without knowing that this commit breaks the code so we can say get add dotg commit DM and we can say add another console log now if you run get log you'll see many commits keep in mind that git log here includes all commits that have happened before as well everything from the latest console

logs to previous ones to resolving our merge conflicts even John's changes right here there's a lot of stuff that's here everything from when we first started working on this project now before I tell you how to remove this bad code using git reset you must understand that there are a couple of different ways to run the git reset command soft reset simply moves the specified commit in history but keeps changes staged in a working directory that means that

whatever changes were made after that commit will be in stage mode and Stage changes are those that we add to G tracking system by running the G ad. command so before running that they're untracked and then once you run git add then you can consider those changes tracked or staged so if you want to do that you can run git reset D- soft and then add a commit hash the second way to use git reset is called a mixed reset

and for that one you don't have to pass any flag as it's the default one mixed reset moves to the specified commit in history unstaged the changes and keeps them in the working directory thus all changes made after the specified commit will be in your working directory but they won't be staged you have to manually stage them if you want to by using get add Dot and finally there's a hard reset it moves to the specified

commit in history and discards all changes in the working directory and staging area all those changes made after the selected commit will be discarded entirely and you won't see any Trace you can run that command by running get reset Das Dash hard and then add a commit hash now let's actually go with the mix res set by copying one of the commit hashes from here but which one do we want to copy we can scroll to the top

to see the newest ones and we have this add another conso log which we know breaks her code so we want to avoid this so instead we can go all the way to add GitHub console lock to hello.js so let me copy this commit hash and press Q to exit get log next we can run get reset and then paste this commit hash and press enter unstated changes after reset hello JS so

take a quick look at the file explorer you'll see that the hello.js file is blue which means that we have some changes in that file and if you pay attention to this green line here this means that these are the two additional changes which are currently unstaged so let me put that in other words the changes for the two commits that we have added are right now here and are unstaged and we can verify that by

running git log and you can see that there are no additional commits after the add GitHub console log there's no add hello Dev Adrian and there's no add bat code we can exit that for a second and we can also run get status and you'll see that hello JS has been modified and specifically it has been modified with the file changes that came from the two commits that we have reset so now it's completely up to you whether

you want to keep those changes modify them and then again stage and commit them in this case I will simply remove them from the code like they never happened and we can go to get log and see that there's no more add buggy code or ADD conso log and see that those two additional commits happening after the GitHub conso log are completely gone all of your mistakes completely deleted I hope this makes sense so you can take a

moment and try the other two variations of the git reset on your own and see how they differ once you do that we can move to the next Advanced git command git revert let's say you've deployed a feature that broke production and you want to undo its effects without losing the commit history you want the locks to be there but you want to revert to an old Commit This is the exact situation where you want to use get revert it's ideal when

you have nothing to hide and you want to maintain a clear record of changes that you did it's almost the opposite of reset let me show you how it works I'll add another console log that says something like trying out revert okay and I will commit it by running git commit DM add revert console log oops I forgot to do a get ad before that my mistake let me do get ad Dot and I'll use the up Arrow two times to get back

to the previous command and commit it now let's say you want to revert to the previous version of your codebase that didn't contain this conso log but you want to keep it in the logs you can run get log you can find the commit hash of the previous commit without containing the revert conso log copy it and then run get revert paste the commit hash and then we get something that looks like this which is somewhat familiar to us it's like a

mini merge conflict it's trying to figure out what we want to keep and what we want to remove in this case I'll manually remove lines of code that we don't need so we don't need these ones and we don't need the trang out revert because we want to abort or revert that one you can save it and once you save it we want to add those changes to staging by saying get add Dot and then you can

run get revert continue and this command will successfully finalize the revert as you can see we're getting there this message is saying that we will revert this commit but we just need to provide it a commit message or we can exit this window by pressing colon QA and then exclamation mark and then press enter there we go so one file has changed there's there's one insertion and two deletions and we are back to where we were similar to get reset right but now

if you check out get log you can see that a new commit revert add GitHub conson log to hello has been added to our G history as I said similar to reset but depends whether you want to hide your tracks or you want to show everybody that you messed up and you fixed it later on both have their own use cases now let me show you g stack another Super useful git command let's say you're in the middle of developing a

feature but an urgent bug caused by your teammate comes up and your boss wants you to work on it first you haven't finished your feature and it's not ready to commit yet but you still have to keep your active changes somewhere so you can get back to them later on and you can start working with this bug soon and that's exactly what get stash does it lets you temporarily save your uncommitted changes both staged and un

staged without actually committing them let's say that I am in the process of implementing an important feature and then I have multiple lines of code which I don't want to lose that's going to look something like this you can also write some of your own very important code here so now how do we save this code so we can use it later on so we can focus on something else for the time being it's super useful you

just run get stash as you can see your important code is now completely gone and we got a message here saying that saved working directory on dev Adrien so now you can go ahead and implement this urgent fix that your boss requested you to do you can then run get add get commit and say save the day of course this is not a realistic commit message but in this case since you implemented the Urgent Fix You indeed did save the

day and you can even push that code if you want to so I can run get push great the day is saved but now you want to get back to implementing that other feature you were working on before your day was ruined by a bug to get our code back simply run get stash apply and then enter a stash name but how do we get to a stash name you can run a command get stash list to see a list of all stashes

currently we have stash at curly braces zero so let's do just that get stash apply stash at curly braces zero and press enter and you can see that our code got back right here since the implemented fix was on similar lines as our existing code we have yet another merge conflict but by now you should be an expert in resolving those so simply remove the lines around it and keep both

the Urgent fix and and your work in progress and that's it the day is saved and you can continue working on your feature I don't want this to happen but I'm sure that someday soon you'll encounter an issue in your code that requires you to use one of these three very handy git commands G stash will definitely be the first one so when the time comes you'll know how to use it great work although commands like get

add commit push merge checkout stash and others may seem hard to remember even though they're simple once you get a hang of them there is an alternative method as developers we always discover more efficient techniques for accomplishing repetitive tasks so let me introduce you to a completely different approach for executing these git commands before I teach you these very powerful techniques do remember that it is still super important to understand the behind the scenes of how these

commands work and that's what you've learned so far which makes you ready to utilize the power and simplicity of this new approach I'm referring to using git through a GUI a graphical user interface it doesn't require memorizing commands but instead offers a more Visual and userfriendly approach to Version Control allowing you to execute G commands through simple buttons panels and menus many editors and IDE offer G integration through

agui but their functionality it is limited compared to web Storm's features so I highly recommend downloading webstorm it's free and packed with powerful git Integrations but if you're prefering not to no worries you can still follow along with the video and check if your editor supports some of the features that I'll be showing in webstorm we can start from scratch by creating a completely new folder and create a new readme.md file within

it there we can say something like T

testing git through a GUI a graphical user interface first things first let's turn this folder into a git tracked repository on the top left or the bottom right you can click Version Control and then create a git repository open up the current folder and immediately this folder will be turned into a git repository this is the same as running the git init command and immediately you you can see that

right now you're in the main branch I actually prefer having this on the bottom right so I'll press command shift B then I'll search for toolbar and I'll hide it if you hide the toolbar then its details will appear at the bottom right so right now you can see all of the branch details here now that we have initialized the get repo let me show you how we can commit using a GUI you can go to the bottom right corner and then

simply click commit it'll immediately lead you to the this menu showing you all the unversioned files you can select all of them and then add a commit message I'll say initialize the repo and I will commit the changes it is as simple as that now if you click on the main branch once again you'll see that you also have the possibility to create a new Branch so let me show you how to do that you just

press new branch and enter a branch name let's do something something like new Dash branch and it even asks you whether you want to check out to it immediately for now I'll untick that and click create so since we haven't moved to it immediately how can we move to that Branch now well you can see this little popup telling us that a branch New Branch was created so we can click here and here you can see the new Branch but

an even easier way to switch to it is to click at the bottom right or the top left and then simply select the branch you want to move to and click check out and that's it you are on your newly created Branch I love how webstorm conveniently organizes these branches into recent local and later on will even have remote so you always know at which versions of your branches you're working on but of course that begs the question

how can we push these branches to remote we usually use the command get push dasu origin and then the branch name but with webstorm you can do it in a single click just click push click Define remote and you'll have to enter the URL of your remote origin to get it I'll create a new repo and I'll call it webstorm undor git and click create repository and I'll simply copy the URL

back in the code I'll paste it click okay and click push check this out all of your code has been pushed to the new branch and now a new origin New Branch also exists and it even asks you whether you want to immediately create a p request I won't do that yet first I want to teach you how we can add a few more commits to our read me so I'll add a new line and then simply say commit one then

you can either go to this commit icon on the left side and then select it and then mod if the commit message and simply press commit or let's add another one commit to you can also go to the bottom right and simply click commit there which is going to lead you here and you can add a commit too this time let's check out the second

option which allows you to immediately commit and push it's going to ask you to which

branch you want to push in this case we want to push it to the new Branch so let's simply click push and and in a matter of seconds all of those commits are right there of course if you committed some changes earlier like I will do in this case commit three add it and then add commit three and then later on you want to push them you can go to the bottom right select push from here

and repeat the same steps it is super convenient now what about pulling some changes let's say that somebody else comes to your repo and inserts an additional message like a commit 4 right here and updates the readme how would you get access to that change directly within your repo well typically you would have to run git pool but by using webstorm you can just go here and press update project it's going to ask you whether you want to

merge incoming changes into the current Branch or whether you want to rebase which will rewrite the commit history by rebasing your current Branch onto another Branch effectively moving it to a new commit in this case we don't need to rebase we'll just merge it into our current Branch so just press okay and in a matter of seconds the commit forward that another person pushed is right here within our local repository what about viewing history well let me show you

something really cool look for the git icon at the bottom left this one here if you click it and expand it you'll see a ton of different git functionalities you can see the exact changes that were made in in this repo who created them and when all within this Nic looking graphical user interface and within here you can do everything and more that you can do with typical get commands all by choosing one option from a menu that is in English no

obscure commands if I open up the repo for JS mastery. Pro platform it is super convenient seeing exactly what is happening within our projects we can see what changes were made who made them when and whether the tests have passed check this out you can even see the merges that were happening this makes it super easy to go back to previous versions of your code base if needed in addition to this if you press here and

then you press console you can see all the commands that webstorm is running for you of course you know some of these base ones but some of these more complex one well it's going to be easier to do it using a GUI now how can we use webstorm to merge one branch into another typically we would have to create it and then run G merge command to merge it into another branch in this case select a main branch and press this Arrow right

here it's going to allow you to merge main into New Branch or vice versa similarly if you click the arrow on this Branch you can easily create a new Branch from this Branch or you can updated in this case let's go to Main and let's merge main into the new branch in in this case it says it's already up to date so we're good but what about pull requests one of web Storm's coolest features is that it

allows you to do all sorts of tasks directly within the IDE you don't need to use GitHub desktop or even GitHub to perform different actions everything you need is right there so let's talk about creating a pull request on the

left side you can find a p requests tab for you it might be a bit of a different icon in this case it's a GitHub icon for me within here you can do everything PR related so let's open up a new P request

in this case we want to merge the new Branch into the main branch but before we do that we have to first create a remote origin version of the main branch so let's do that first I'll head over to main by checking out to it and I'll simply push it push to origin main it is as simple as that we can immediately now use this create p request feature to move us to this pool request menu we

want to merge from Branch New Branch or you can choose origin New Branch to origin main exactly what we wanted to do you can add a title of this for request such as Implement for commits because that's what we added into the readme description is not needed and directly from here you can add reviewers assignes and labels in this case I'll just click create for request and there we go it is right here if you

go to GitHub and click P requests you can see the new pull request indeed has been opened the next cool thing is that you can see exactly the changes that were implemented for each specific commit so if you go here you can see that first at some time they added a commit one then later on they added commit two and so on but now we're interested in all commits so now you can review the changes that were made for

each file separately first we want to dive into the readme.md file so rightclick it and then it'll open up a diff a diff or a difference refers to the changes of a specific file on two sides of the coin or should I say on two different branches one is the branch that we're trying to merge and the other one is the branch we're trying to merge the changes to you can view those things either split like this or

unified in a single editor and based on the green or red color you can see the changes that were made in this case I'm happy with the changes and I'll click submit and if you're a reviewer you can immediately add a review directly within webstorm you don't have to go to GitHub just by pressing a plus here you can add a comment something like fix this line and you can even start a review there we go finally you can

submit your pull request review and you can add one final comment looks good to me great finally if everything looks good you can press the three dots next to the request review button and then say merge review merge merge pool request one it'll Implement four commits that's the title of the pool request and we can say merge believe it or not this is it so now on Main you can simply update the

project and all of these commits will come directly to your code base I mean just how convenient and intuitive and efficient this is at start you might feel like you are kind of cheating the system or that you're a bit of a less of a developer by using this help that webstorm provides but don't feel that way any tool that can make you more efficient is welcome and what matters is that you understand what webstorm is doing behind

the scenes and you do because you came to this point of the video I think that in just a couple of minutes I've showed you all of the primary functionalities that before using the terminal I spent more than an hour to explain

and we've accomplished all of that in more in just a few minutes but there are so many more things that you can do with webstorm one of these things is a fetch if you see

this icon on top right you can just click it and it'll fetch all the latest changes another thing is how easily you can delete branches we no longer need this one since we merged the changes so you can just press delete there we go that was it you can also compare different branches Mark branches as favorite or even cherry pick from a specific commit if I go right here to get at the bottom left and go back to

log we can see all of these different commits that I added cherry picking is a very Advanced technique but in webstorm it is as simple as clicking this cherry pick icon you click it and it'll allow you to pick the changes from this commit you can press merge and then you can see which changes you want to accept this is similar to reverting or resetting to a specific commit but it gives you even more power to pick and choose which

features you want to keep and which ones you want to remove you can also rename different branches or even revert to specific commits directly from within this view typically you would have to get the commit hash figure out what you want to do here you just press revert commit and that's about it we cannot do it now because we have unate changes but that's also pretty easy to resolve once you have webstorm you just go here and

you can easily resolve all the conflicts and then commit and when a technology is so good that it feels like Magic based on how many things is it doing for you and it's doing them well well it almost feels like magic for us developers so whichever complicated feature you want to use you can use it within webstorm with ease these were just a few examples of how we can use the webstorm GUI to perform many different git operations

without needing to rely on a single git command and that's only one of the reasons why I love webstorm while you can accomplish some of these tasks in other editors not all of them are possible as there's a clear difference between an editor and an IDE right so go ahead and explore other options try breaking some things fixing them reverting resetting cherry picking you can only get better at something by trying it out yourself I'll also leave a

reference on how you can use all of these features within the G reference in the description of this video but with that in mind we covered a lot about git if you're still with me and you have successfully completed these git actions with a solid understanding and even tested them out yourself congratulations you can confidently add this skill to your resume LinkedIn profile or anywhere you're applying for a Dev rooll and and don't worry if you're not feeling fully

confident yet or are concerned about remembering everything git can be tricky at first but with practice it'll become second nature especially once you get used to these graphical user interfaces but still it's easy to get stuck and lose your mind about something especially when you have to pull some very obscure commands out of your pocket so for those situations don't forget to download the cheat sheet in the description so you can have a handy reference whenever you need it in the future and

great job with sticking till the end of this essential crash course thank you so much for watching and I'll see you next time have a wonderful day