**GitHub Actions – The Complete Guide**

**Course Overview**

* **Course Name:** GitHub Actions – The Complete Guide
* **Instructor/Platform:** Maximilian Schwarzmüller
* **Course Link:** https://www.udemy.com/course/github-actions-the-complete-guide
* **Estimated Completion Time:** 10.5 Hours
* **Start Date:** 4/8/2025
* **End Date (Goal):** 4/22/2025
* **Main Learning Objectives:**
  + 10 Lessons per day
  + Create this reference document
  + Be able to use Github & Github Actions
  + Be able to add this to my resume

**Modules & Lessons**

**Section 1: Git & Github Crash Course**

* **Lesson 1:** Welcome to the Course **✓**
  + Learn all the core essentials
* **Lesson 2:** What is GitHub Actions **✓**
  + Notes:
    - A Workflow Automation Service offered by the GitHub Company
    - It automates all kinds of repository-related processes and actions
    - A repository is a bucket that contains code for applications, packages, or websites
    - Two main areas of processes that can be automated
      * CI/CD (Continuous Integration Continuous Delivery/Deployment)
      * Code and Repository Management

**Lesson 3:** Git, GitHub, & GitHub Actions **✓**

* + Notes:
    - What is Git
      * Git is a free version control system tool
      * You can download and install on your machine
      * Git manages source code by allowing you to create snapshots (commits) of that source code.
    - What is GitHub
      * The name includes the word “Git” which is a standalone independent tool
      * GitHub is a company that offers cloud Git Repositories and services
      * You have local repositories and remote repositories. These two relate to pushes and pulls
      * Also offer code management and collaborative development features. This relates to Issues & Pull Requests.
    - What is GitHub Actions
      * Automation Services provided by GitHub

**Lesson 4:** About this Course **✓**

* + Getting the most out of the course
    - Watch the videos at your pace
    - Watch in order
    - Watch the same videos multiple times
    - Practice what you learn, pausing and learning on your own
    - Learn and grow together with Q&A, Discord based community
* **Lesson 5:** Join the Discord Community **✓**
  + Notes:
    - Single page about joining the Discord Community.
* **Lesson 6:** Course Code Attachments & Slides **✓**
  + Notes:
    - Single page about where to find code and slides
  + Resources:
    - <https://github.com/academind/github-actions-course-resources>
* **Lesson 7:** Course Setup **✓**
  + Notes:
    - Single page about using an IDE
    - Use VS Code

**Section 2: Git & Github Crash Course**

* **Lesson 8: Module Introduction** **✓**
  + Notes:
    - This section is optional
    - How to work with Git and GitHub
* **Lesson 9: Getting Started with Git ✓**
  + Notes:
    - You must install Git on your system
      * I installed Git on the PC to support Visual Studio Code access to Git but not GitHub Desktop. I would rather all my development be on my VM Development Server on the PowerEdge T30. Eventually, I will ensure that this server backs up nightly to the NAS.
* **⇨ Lesson 10: Configuring Git** **✓**
  + Notes:
    - After installing Git, you really need to configure it. Most importantly, you need to set your email and username.
    - There are more options at https://git-scm.com/docs/git-config
  + Key Commands:
    - git config --global user.email [you@example.com](mailto:you@example.com)
    - git config --global user.name "Your Name"
  + Resources:
    - https://git-scm.com/docs/git-config
* **Lesson 11: Project Setup** **✓**
  + Notes:
    - We downloaded a provided resource file and moved it to the /home/nick/Development directory/
    - We open the index.html in VSCode
* **Lesson 12: Working with (Local) Git Repositories** **✓**
  + Notes:
    - You have to create a repository
  + Key Commands:
    - git add index.html fails because we are trying to add a file to an uninitialized directory.
    - This is the result that shows it wasn’t initialized.
    - fatal: not a git repository (or any of the parent directories): .git
    - git init results in
    - Initialized empty Git repository in /home/nick/Development/.git/
    - This will initialize a repository including subfolders
    - It also creates a hidden .git folder
* **Lesson 13: Staging Files & Creating Commits** **✓**
  + Notes:
    - Here we are going to learn how to commit to gitlab
  + Key Commands:
    - git add html we could also run git add . which would have added all files in this directory. We can also separate files by name with a white space or add a subfolder with a /
    - git status shows which files are staged
    - Now we can do a git commit

We can also use the -m flag. to leave a short message between the quotes.

* **Lesson 14: Multiple Commits & Checking Out Snapshots** **✓**
  + Notes:
    - We will look at what happens when we do multiple commits and how to manage those commits.
    - First he did a get status to show where he was in the stack.
      * In parenthesis there is a pointer managed by Git to show which state is currently loaded.
      * Once you have multiple commits, you can move between commits with the “git checkout <id>” command. The other commits wont be deleted, you just load another state.
      * To demonstrate, he changed the index.html file and committed it.
      * He introduced the “git checkout" command and checked out the previous commit. He made a point of showing the HTML code in the example file was now missing the last changes.
      * He also pointed out the pointer says HEAD but not MAIN.
      * Also don’t forget the changes aren’t lost. To get back to the master branch you can just checkout the MASTER (MAIN)
  + Key Commands:
    - git status

This shows us that the top of the commit stack is marked with HEAD->MASTER (His said MAIN)

* + - git checkout <id> (determined by “git log”)   
      git checkout master (you need to remember what the main branch is called. By default, now they use MASTER)
  + Resources:
* **Lesson 15: Reverting Changes with “get revert” ✓**
  + Notes:
    - How do we undo a commit? With git revert
  + Key Commands:
    - git revert <id> will revert to the previous commit as the head
  + Resources:
* **Lesson 16: Resetting Code with “git reset” ✓**
  + Notes:
    - Sometimes you really want to get rid of a commit. The git reset –hard will do this
    - It used the id of the commit you want to go back to
    - It rewrites history and you are losing data
    - We are talking about the .folders and how they don’t belong to the code project
  + Key Commands:
    - git reset --hard <id>
  + Resources:
* **Lesson 17: Staging Multiple Files & Ignorning with gitignore✓**
  + Notes:
    - We brought over some new files
      * Index.html, logo.png and styles.css
    - We can add these files to git so that we can commit them via
      * Git add index.html logo.png styles.css
      * Git add .
      * Git add \*
    - The problem with the first is will become unwieldy.
    - The problem with the second and the third is there isn’t a way to skip some files
    - A .gitignore file will list files I want to ignore
      * I create a ignore.me and and .gitignore
  + Key Commands:
    - So we can add the files with
    - git add index.html styles.css logo.png
    - or
    - git add . to add the all the files
    - but since he had a .vscode file, it would add that as well. This is where .gitgnore comes in.
    - Files in the .gitignore file are not included in the add or the corresponding commit. The format is filenames one line at a time. This causes vscode to gray our the file
  + Resources: Extrafiles.zip
* **Lesson 18: Understanding Branches ✓**
  + Notes: Branches are related to commits
  + You start collecting commits
  + You get one “Master” branch whenever you initialize a directory
  + You can create additional branches to say fix a bug, or work on a feature.
  + In this example we are going to look at the 3 lines of code in the <body>
    - We want to make some changes without disturbing the original code.
  + Key Commands:
    - git branch <name> will create a new branch called “name” using the latest commit
    - git merge <name> will cause all the commits of one branch into another
* **Lesson 19: Working with Branches ✓**
  + Notes:
    - We are going to create a new branch called “feature-restructure”
  + Key Commands:
    - git branch feature-restructure
    - git branch will list all existing branches. It also shows which branch is currently active.
    - git checkout feature-restructure will change to the other branch
    - You can delete a branch with git branch- D feature-restructure
    - You can create a branch and check it out at the same time with git checkout -b <name>
* **Lesson 20: Merging Branches**
  + Notes:
    - Now lets see how we can work with commits in the different branches
    - Now we want to add a change to the master branch such as a ! after the H1 text.   
      git checkout master to ensure we are in the master  
      git add . to add the change to the master branch  
      git commit -m “Change to main branch” to commit
    - Now we can change back and forth and notice that the changes in the branch and different than the changes in main.
    - Let’s merge them together. Change to the main branch  
      git checkout master  
      git merge feature-restructure will merge feature-restructure into master
    - We were able to show that the master branch is now the combination of the two, and we are going to delete the feature-restructure branch
    - git branch -D feature-restructure
  + Key Commands:
    - git merge feature-restructure
  + Resources:
    - 04 Updated Files.zip
* **Lesson 21: Github Introduction**
  + Notes:
    - We used Git locally but if your machine crashes, if you need to work with others, or if you switch machines.
  + Key Commands:
  + Resources:
* **Lesson 22: Creating a Github Account**
  + Notes:
    - Key feature of GitHub is the remote repositories
      * Cloud storage buckets which can be connected to your local account.
    - I had an account from a previous attempt at this course and other things
      * Username: **nick-gnu-1**
      * Password: **In 1password**
    - The next thing he does is start to create a repositiory
      * I already have 15
      * We created a new repository called github-crash-course. It prepends nagray/ to it so that it is unique.
        + Public with all other settings by default.
  + Key Commands:
  + Resources:
* **Lesson 23: Creating a (Remote) Github Account**
  + Notes:
  + Key Commands:
  + Resources:
* **⇨ Lesson 24: Connecting Local & Remote Repositories**
  + Notes:
    - We are going to connect the remote repository “github-crash-course” to our local repository via the “git remote add” command.
    - Thereafter we use “git push” commands to the remote so they are stored there.
    - And you can download commits which you don’t have via “git pull”
    - The command “git remote add” is not going to be enough. We have to give it more information.
      * A local name. “origin” is the most common name (local identifier)
      * You must provide a URL to the remote repository
    - This worked!
  + Key Commands:
    - **git remote add origin git@github.com:nick-gnu-1/github-crash-course.git**
  + Resources:
* **Lesson 25: Pushing Commits & Understanding Permissions**
  + Notes:
    - We want to push out code to the remote repository. We have the “git push” command.
    - If we do this “git push” without some setup we get an error

*“Fatal: The current branch master has no upstream branch.To push the current branch and set the remote as up stream, use*

*git push -–set-upstream origin main”*

* + - This means the current local branch does not know which branch in the remote repository belongs to this local branch.
    - For a one time push you can “git push origin master” and it will create that local branch on the remote with the same name. but if you execute this you get an error. It doesn’t know who you are or your permissions
    - Note that we are on the “master” branch.
    - If I do that one time push, I will still get an error
    - Remote: Permission to nagray/github-crash-course.git denied to nagray
    - To authenticate, you can say  
      “git remote set-url origin https://nagray@github.com:nagray/github-crash-course.git  
      fatal: unable to access ‘https//github.com/nagray/github-crash-course.git/’: The request URL returned error: 403”
    - This should make sense because this is an unauthenticated request. You must be an authorized, authenticated user to make changes to a repository on GitHub.
    - So we add the command git set-url origin https://nagray@ github-crash-course.git”
      * If we use this, VSC prompts for a password. But it is not really a password, it is a PAT (Personal Access Token)
      * This is created on the GitHub web page under Settings⇨Developer⇨Personal Access Tokens⇨Tokens (Classic) ⇨Generate New Token
        + Give it a name/Expiration/and check repo
        + Save the token (1Password) since you will never see it again
      * Paste this token in for password. In order for it to get saved you have to add “git credential-manager” It was quite a bit of work to get this going. After downloading it. I had to
        + sudo dpkg -i gcm-linux\_amd64.2.6.1.deb
        + /usr/local/share/gcm-core/git-credential-manager configure
        + git config --global credential.credentialStore plaintext
  + Key Commands:
    - git push
    - git push –set-upstream origin master
    - git remote set-url origin https://nagray@github.com:nagray/github-crash-course.git
  + Resources:
* **Lesson 26: Github & Branches**
  + Notes:
    - We are going to group some parts of the page into a section and create a new local branch to put it in called “feat-section” checking it out as we create it with “git checkout -b feat-section”
    - We made the change and saved it. Next we added the file to the local repository and then committed it with “git add .” and “git commit -m “feat-branch”
    - Now lets push it to the remote repository.
    - We can go to the GitHub repository and select either one. You can also browse the code and make changes to the repository.
    - We will merge this code from the branch. We could do it from inside of GitHub, but instead, we will do it this time from the git locally.
    - So, “git checkout master” and “git merge feat-branch” and push it to the remote repository
    - We can delete the branch on GitHub by going to “view all branches” and deleting file” it will still exist on the local branch, which we can delete using “get branch -D feat-section”
  + Key Commands:
    - git checkout -b feat-section  
      git add .  
      git commit -m “feat-branch”  
      git checkout master  
      git merge feat-section  
      git push  
      git branch -D feat-section
  + Resources:
* **Lesson 27: Readme Files & Pulling Changes**
  + Notes:
    - We created a README.md on GitHub. The .md is the extension for the markdown language. The ReadMe file is required to be save with an MD extension.
    - Note that the file was created on the GitHub site and doesn’t exist locally yet.
      * Of course, we are going to do a “git pull”
  + Key Commands:
    - git pull
  + Resources:
* **Lesson 28: Cloning Repositories**
  + Notes:
    - Started this section by talking about the tabs across the top  
      (Code, Issues, Pull Requests, Actions, Projects, Wiki, Security, Insights, and Settings)
      * Actions is for GitHub Actions
      * Issues and Pull Requests will be covered later
      * Project lets you set up Project Management features.
      * Wiki – Sets up a wiki for this repository.
      * Security – Lets us change security settings
      * Insights – Gives us analytics
      * Settings – Allows us to set all kinds of settings.
    - We signed out of nick-gnu-1, but saved the “github-crash-course” URL and went back to GitHub and looked at GitHub as an anonymous user.
      * We can still see the code, because it is a public repository.
      * We have lost access to Settings.
      * We can still browse multiple branches if they exist
    - We created a second user to understand some key concepts
      * I created
        + A gmail account [nick-gnu-2@gmail.com](mailto:nick-gnu-2@gmail.com)
        + A unix user on Ubuntu-Development-Desktop called nick-gnu-2
        + a user called nick-gnu-2 on GitHub
      * I added the git config variables user.name and user.email to the account.
    - To help as a second user, you must start by getting a copy of the code
      * Click the code button. This will allow you to download the code in different ways.
      * Since the user nick-gnu-2 already has git installed, we can copy the URL and “clone” the repository to the local account.
      * We clone the repository using the “get clone” command
        + If you do this, you don’t have to create a link. It is automatically setup for you (so you can just update with “git pull”
        + Also, you can include another variable at the end of the command. This will be the name of the destination folder

We used git-demo, just to see it work

* + - * + Notice that the .vscode folder is missing. Like we expected
      * To show that it really is linked you can run the “git remote” command. You will see “origin”
      * Using origin as an argument, you can see exactly where this is linked to with “git remote get-url origin” Note that you can’t see the original creator, which makes sense.
  + Key Commands:
    - git config --global user.email nick-gnu-2@gmail.com
    - git config --global user.name " nick-gnu-2"
    - git clone <https://github.com/nick-gnu-1/github-crash-course.git> git-demo
    - get remote
    - get remote get-url origin
  + Resources:
* **Lesson 29: Making Changes as A Different User (Non-Owner)**
  + Notes:
    - We are still logged into the Ubuntu-Development-Desktop as nick2 we create a new branch called feat-color
    - We have changed the styles.vss file. By modifying the color of h1.
    - We add and commit locally, and attempt to push the change
    - Of course this didn’t work because we don’t have permissions. But it didn't work try to save it until I gave it my username and password. But then it failed because I didn’t have a token for nick2 yet. Once I generated and new token and tried again, then it gave me permission to denied 403 Error, which means access forbidden.
  + Key Commands:
    - git branch -b feat-color
    - git add .
    - git commit -m “changed color”
  + Resources:
* **Lesson 30: Github Issues & Collaborators**
  + Notes:
    - There are two main ways of fixing this
      * We could fork the repository
      * Or if we cloned it, we could reach out to the creator of the Repository and ask for permission to push to this repository.
      * One way to reach out to the owner is to create an issue**.**
    - **Issues are not meant to contact the owner in general. It is meant as a method to report bugs or suggestions.**
    - The owner can either accept or reject our suggestion. There is a third option that the owner may ask us to make that change ourselves.
    - Click on “New Issue” button and submit the issue
    - As the owner, you will see a number appear next to the issue in this repository.
    - Click on the issue to expand it.
    - If we want to allow this person to push this issue. I go to Settings-> Collaborators and add other accounts that should have access rights to this repository. Push the Add People button. You will have to know their username.
    - The other person receives an email with a link that takes them to the accept invitation page. The top of the screen will show a message that they now have push access to the repository.
    - As the owner I might want to restrict access within that repository. Under settings-> branches we can set up a branch protection rule
    - Be aware that there are two different kind of settings, One for the account and one for a repository
      * To get to the repository settings select the specific repository by clicking on the cat icon and when you are in the repository it is on the main bar
      * To get to the system settings, click on the user's icon, and about 2/3 of the way down, you will see Settings
    - Select add rule to the, and type in the name of the branch. The most common is to protect the main/master branch. Then select require a pull request before merging and require approvals.
    - Now to push you just need to set up the link from one (local) account to another (remote) account with “git remote set-url origin https://nagray2@github.com/nagray/github-crash-course.git”
    - Don’t forget that you had to generate and use an access token for the new account
    - And push the branch.
  + Key Commands:
  + git remote set-url origin <https://nagray2@github.com/nagray/github-crash-course.git>
  + git push origin feat-color
  + Resources:
* **Lesson 31: Working with Pull Requests**
  + Notes:
    - We know that as a non-owner user of a repository, we can’t push directly to the main branch since it has branch protection turned on
    - So what we want to do is create a “pull-request”. It is a request to the owner to pull our changes into the main branch
    - From the non-owner click on “Pull requests”, click “New pull request”. Select the main branch, and then the branch that should be merged
    - Click create pull request, add a title and description. In the description you can include the issue by typing a #1 and github will suggest a link
    - Since we did this all on github. There are changes not on our local machine. A git pull will update it on the local machine the owner and another will update in the collaborator.
  + Key Commands:
  + Resources:
* **Lesson 32: Understanding & Using Forks**
  + Notes:
    - We saw how poll requests work and how we can collaborate if we add people as collaborators.
    - Lets remove nagray2 from collaboration access via settings=>Collaborators=>trash can
    - You normally wont be added as a collaborator but you can still help with a “fork” What you are doing is cloning that repository on GitHub
    - Since it will be your own repository on GitHub, you can clone that to your local machine, make changes, push it to your cloned and owned GitHub repository, and you can actually create poll requests across repositories on GitHub
  + Key Commands:
  + Resources:
* **Lesson 33: Module Summary**
  + Notes:
  + Key Commands:
  + Resources:

**Section 3: GitHub Actions – Basic Building Blocks and Components**

**Section 4: Workflows & Events – Deep Dive**

**Section 5: Job Artifacts and Outputs**

**Section 6: Using Environment Variables & Secrets**

**Section 7: Controlling Workflow & Job Execution**

**Section 8: Jobs & Docker Containers**

**Section 9: Building & Using Custom Actions**

**Section 10: Security & Permissions**

**Section 11: Wrap Up**

**Action Items**

* **Exercises:**
* **Projects:**
* **Discussions/Group Work:**

**Progress Tracker**

| **Module** | **Lesson** | **Status (Not Started/In Progress/Completed)** | **Notes** |
| --- | --- | --- | --- |
| 1 | Lesson 1 |  |  |
| 1 | Lesson 2 |  |  |
| 2 | Lesson 1 |  |  |

**Final Thoughts**

* **Key Learnings:**
* **Next Steps:**
* **Additional Key Commands:**