Learning Objectives

Learners will be able to…

- **Define version control**
- **Connect version control actions and GitHub actions**
- **Define which VC step require which actions**

info

Make Sure You Know

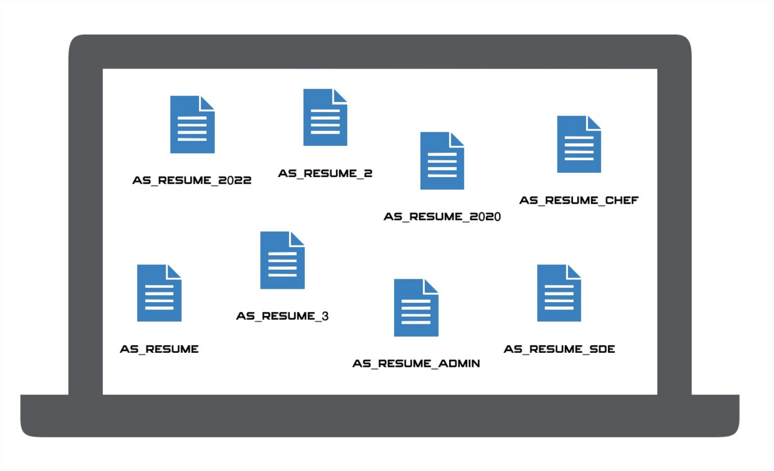
Basic `JavaScript`, `nodejs`, and how to navigate a `package.json` file.

Limitations

Learners should have a GitHub account, as this assignment will use `npm` as a package management tool and GitHub.

Introduction

We often save files to save changes. However, saving a file **overwrites the file's previous version**. It's typical to make a copy of the file to work on since we can only undo so many times.



In doing this, we've created a basic **version control system** by making several copies of a file for backup or editing.

The version control system plays important role in Continuous integration. as it allows to remove manual, human intervention on all steps in favor of consistent, automatic intervention.

We'll use **Github Actions** as an example of setting up CI step. Most of other systems are similar as they allow us to define an execution pipeline for each of our CI/CD steps.

Github gives us a certain amount of free minutes to run our pipelines.

But not only pipelines are important, we can also use **Github Hooks** to perform automated actions each time we push code to a branch.

Version Control

Version control is **a system for tracking changes in software**. It enables developers to identify and monitor each version of their software as it is developed, making it easier to maintain and upgrade their code as needed.

Version control systems use a **snapshot** approach to store versions, taking periodic snapshots of all the different pieces of software so that any changes made can be **tracked over time**.

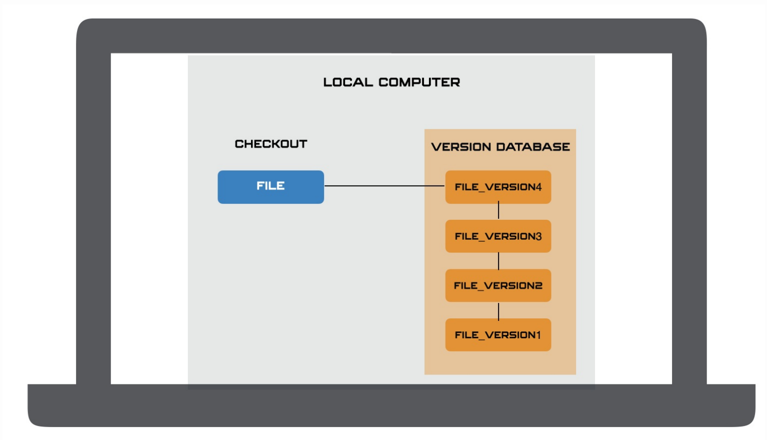


Diagram of a local version control system

With version control, software developers can also quickly go **back and forth between different versions**, identify differences between them, collaborate with other developers on shared projects, and even roll back to a prior state if necessary.

This enables teams to work on the same project simultaneously, ensuring that all changes are properly tracked and accounted for.

Checkpoint

GitHub

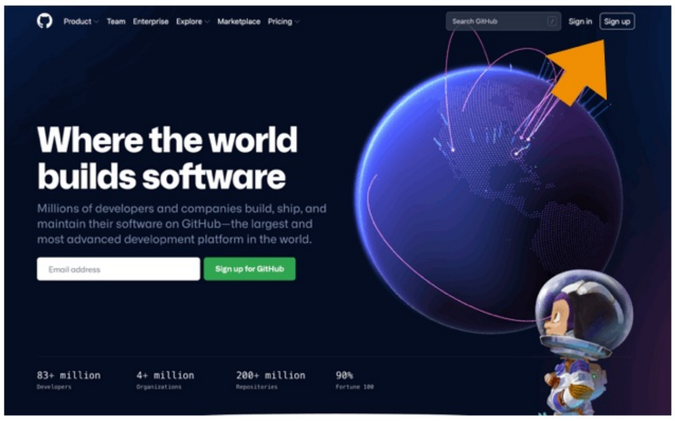
GitHub is an online, collaborative, cloud-based storage service where developers can share and work on code. It offers free and paid hosting for git repositories that lets us push our files up to a remote host without having to host it ourselves.

To be effective users of **GitHub**, we'll explore:

- How to set up a **remote repository** by creating a repository on GitHub and then connecting it to our local repository.
- How to work with GitHub Actions.
- How to **push** changes so we can add our local project files **to the remote repository**.

GitHub Account

We'll need a GitHub account to continue with this lesson. If you don't have one already, you can click on **this link** to create one.


Product
Team
Enterprise
Explore
Marketplace
Pricing
Search GitHub
Sign in
Sign up
Where the world builds software
Millions of developers and companies build, ship, and maintain their software on GitHub—the largest and most advanced development platform in the world.
Email address
Sign up for GitHub
83+ million Developers
4+ million Organizations
200+ million Repositories
90% Fortune 100

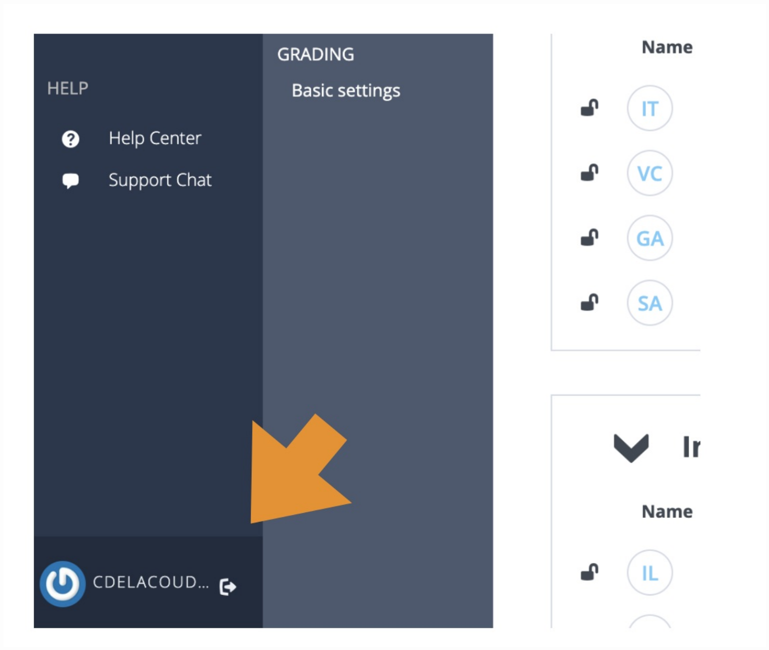
Screenshot of the GitHub.com home page with an arrow pointing to the sign-up button.

Checkpoint

Connecting Codio to GitHub

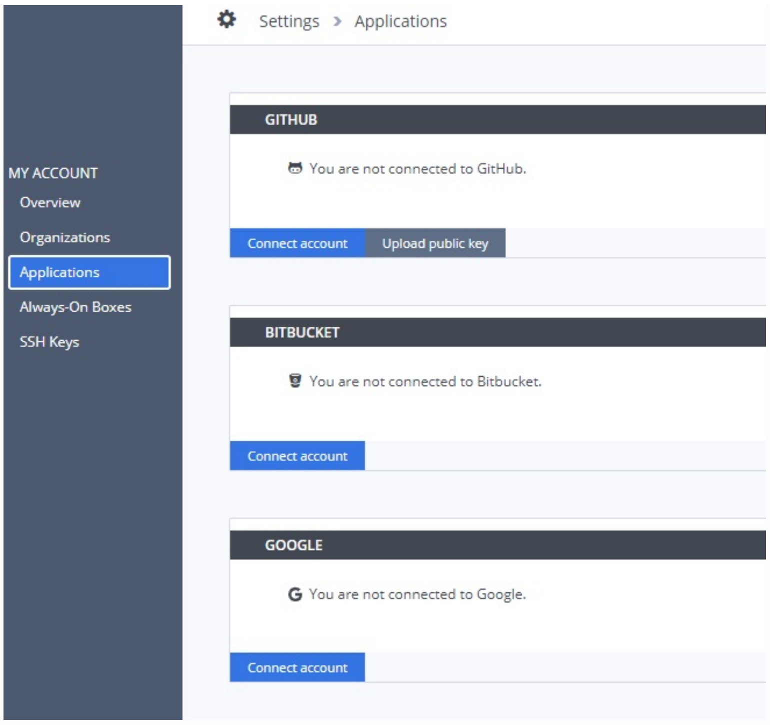
We need to **connect** our GitHub account to our Codio account so that the two can communicate. We only have to do this one time.

- **In your Codio account, click on your username at the bottom of the panel on the left of the screen.**



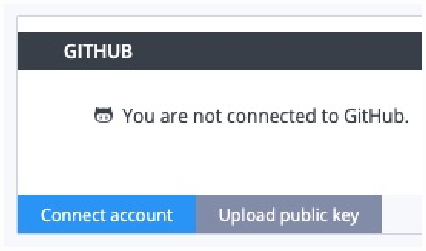
Screenshot of a username icon in the lower left of the Codio panel

- **Click on Applications**



Screenshot of the applications tab from a Codio account panel

- **Under GitHub, click on Connect account**



Screenshot of the connect account button in Codio under the GitHub application

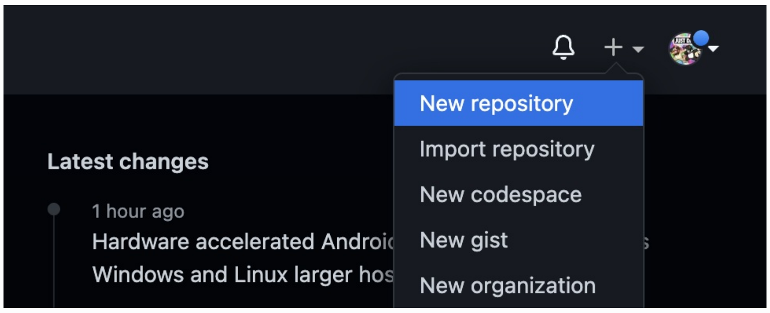
- **We'll be using an SSH connection, so you'll also need to click on Upload public key**

Checkpoint

Creating a GitHub Repository

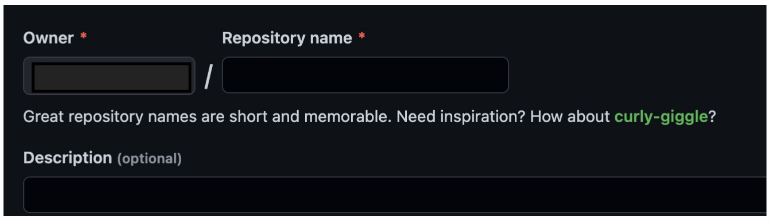
Let's create a repository on GitHub.

- **Click on the + sign in the upper right corner of the screen, and then choose *New Repository*.**



The add new repository option on github.com

- **Type a name for your new repository.**
    - You can also add a description if you like, to help others understand your project.



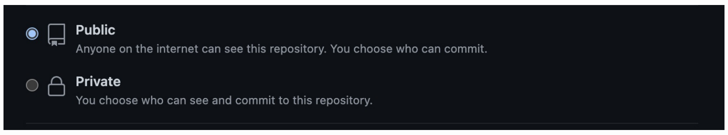
Owner * [redacted] / Repository name *

Great repository names are short and memorable. Need inspiration? How about curly-giggle?

Description (optional)

Image of areas to specify the owner and name of a new github repository.

- **If GitHub doesn't set your repository to be *public* by default, change it to be *public*.**
    - You shouldn't worry about anyone making changes to your code just because it's in a public repository. Instead, this means that other can see your work.
    - They can fork the repository to make a copy for themselves, but they cannot change your code.



[✓ Public]
Anyone on the internet can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

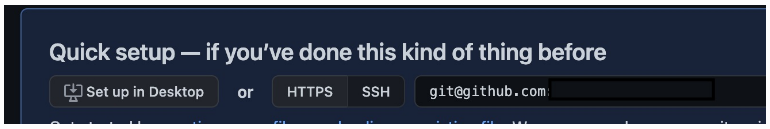
Public or private visibility options for new repository settings

- **Click the green `Create Repository` button to create the repository.**

When the repository is done, GitHub will give it an address. **Be sure to select the SSH address, beginning with `git@github.com:`**.

- **Once you've chosen the SSH address, you can copy it by clicking the button on the right.**

We will need this address to link your Codio project to your GitHub repository.


Quick setup — if you've done this kind of thing before

git@github.com [redacted]

Screenshot of ssh address needed for Codio

Checkpoint

Connecting Local to Remote Repositories

In the previous assignment, we created a remote repository on GitHub. Now it's time to create a **local repository**. In our case, this will be in our Codio assignment workspace.

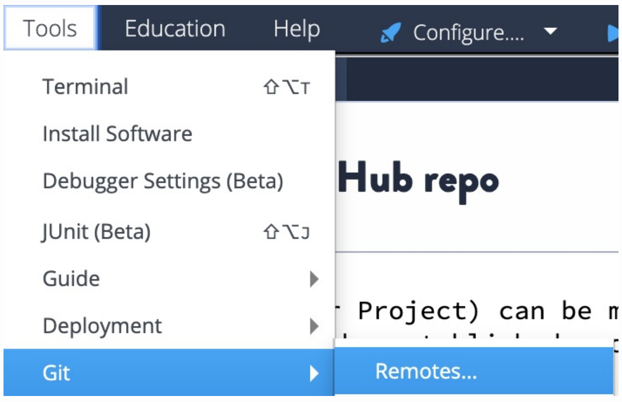
Adding our Remote Repository

You should have clicked the icon to copy the **SSH address** of our newly created repo on GitHub.

- **If not, click on the tab that says `GitHub` and copy the address that starts with `git@github.com:`.**

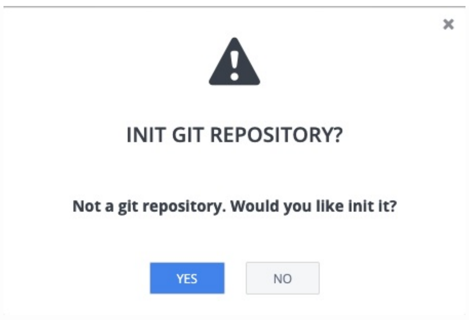
  This will be our remote repository. Once you are back in Codio…

- **Click `Tools` in the menu bar. Hover over `Git` and then select `Remotes...`.**



We've already made the remote repository, but a remote repo depends on the information in a local repo. This is why Codio displays a notice about **initializing our repository**. This action creates a local repo on Codio.

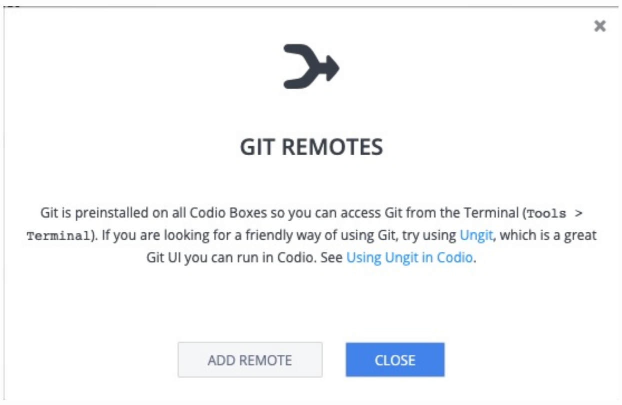
- **Select `YES` to create your local repository on Codio.**


INIT GIT REPOSITORY?

Not a git repository. Would you like init it?

The next page we see is a list of connected remote repositories for this project. Our list is currently empty because our work in Codio is not linked to any remote repositories.

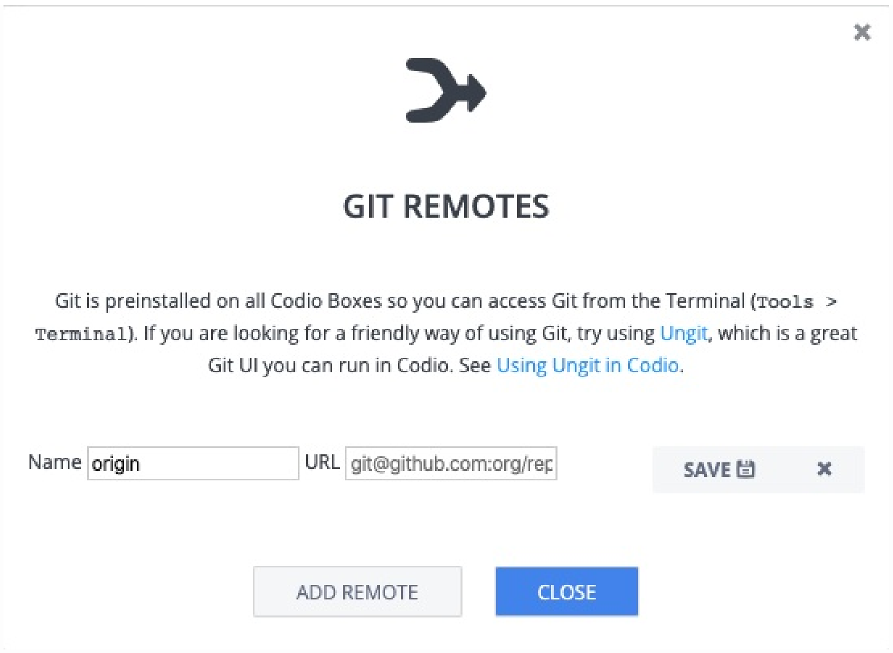
- **Click on the `ADD REMOTE` button.**


GIT REMOTES

Git is preinstalled on all Codio Boxes so you can access Git from the Terminal (`Tools > Terminal`). If you are looking for a friendly way of using Git, try using Ungit, which is a great Git UI you can run in Codio. See Using Ungit in Codio.

Codio then requests a name and URL for the remote.

- **In the name field type `origin`**
- **In the URL field, paste the ++SSH address++ for our GitHub repository.**
- **Click `SAVE`**



Git is preinstalled on all Codio Boxes so you can access Git from the Terminal (`Tools > Terminal`). If you are looking for a friendly way of using Git, try using Ungit, which is a great Git UI you can run in Codio. See Using Ungit in Codio.

Name origin URL git@github.com:org/rep

We should now see a list of remotes associated with the directory. The remote name `origin` is listed in bold as well as the SSH URL for the remote.

- **Click CLOSE**

Checkpoint

Pushing to GitHub

Pushing is how we send commits from our local repository to a remote repository. We don't have to push each time we commit changes. Many developers will make changes throughout the day, but they won't push until the day is over.

important

## IMPORTANT

Code doesn't get saved until we use the `git push` command. If we want to be sure that our code is always safe, we should send our code to GitHub with the `push` command often.

Let's create a local file and push it to our Github repo using the `git push` command.

```
echo "Hello Github!" > test.txt
git add test.txt
git commit -m "test file"
```

- **Try to push our project to the GitHub repository.**

```
git push
```

Because this is our first time pushing to this remote repository, we will need to use a special command.

The exact command is easy to forget. Thankfully, when we use `git push`, Git will tell us what else we need to add to complete the command.

- **Copy the correct command from GitHub, paste it into the terminal, and press `return`.**

```
git push --set-upstream origin master
```

Again, since this is the first push, git will ask if you want to store the SSH keys. This is the last time you'll have to answer this question.

- **Type `yes` and press the `Return` key.**

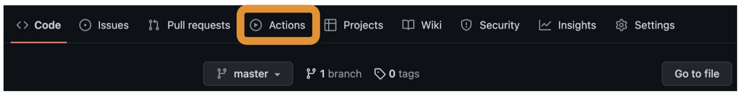
Go back to the tab with the GitHub website on it and refresh the page. You should see the files for our project.

Now that our local repo is connected to GitHub and we can start creating our first GitHub Action.

Checkpoint

GitHub Actions

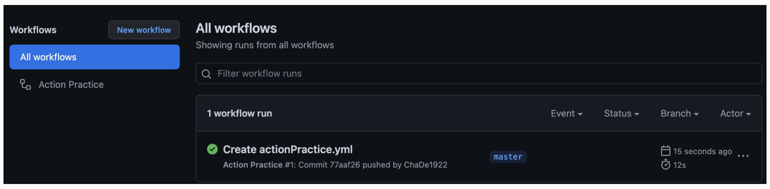
Many programmers use **GitHub Actions** to help them take care of common tasks. They do work in our repositories by making one or more text files that contain actions. The name for these is **workflows**.



The GitHub Actions tab in a GitHub repository.

Workflows can take care of common build tasks, such as continuous delivery and integration. This means that when the master branch changes, we can use an action to compress images, test our code, and push the site to our hosting platform.

We can also have **jobs** that run at certain times or that control what happens when someone interacts with the GitHub repository itself. If someone comments on a pull request, we can get an email, and if someone stars our project, we can run an action.



The All Workflows tab under the GitHub Actions tab.

Actions can run on Linux, macOS, or Windows, as well as on virtual machines or containers. That means we can test our code in different environments. We can even test workflows on more than one platform at the same time.

Every action creates a detailed log that can be used to troubleshoot deploys as they are happening. We will also be able to store **secrets** in the repository on GitHub. This will make it safer and easier for everyone on the team to use.

Actions have a wide community of developers who have already made hundreds of **actions**, examples, and **workflows** so we don't have to start from nothing.

Checkpoint

Workflows

The **workflow** is the engine that drives GitHub actions. In our repositories, we build actions by putting together a set of **workflow files**.

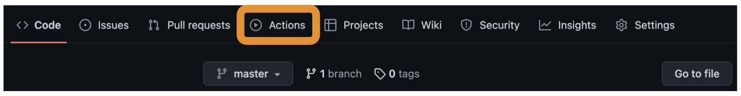
To make a **workflow**, we make a series of **YAML files**, which are like `.txt` files. These files live inside the `.github/workflows`.

Each of these **workflows** can have several jobs that do different things, and each workflow begins when a specific event happens.

Let's work with an existing template to create our first workflow.

- **Navigate our project repository on GitHub from the last assignment.**

- **Locate the `Actions` tab in the repository's toolbar.**



The actions tab in a GitHub repository.

We can see that there are no workflows for this repository. GitHub does suggest a workflow, `Simple Workflow`, based on our existing files. We will use this template and adjust it for our own workflow.

- **Click the `Configure` button to edit this workflow.**

After clicking the button, GitHub does a few things for us:
– Creates a `.github/workflows` directory
– Creates a `.yaml` file named `blank.yaml`
– Fills the file with some template text and appropriate formatting

- **Rename the file from `blank.yml` to `actionPractice.yml`.**

This template contains example actions as well as comments throughout the document describing what each command does.

- **Take some time to explore this template and familiarize yourself with the syntax.**

At the top of the document, this entire workflow is currently named `CI`.

#

```
name: CI
```

- **Rename this workflow from `CI` to `Action Practice`.**

There are a few key lines and sections to be aware of when working with this file.
- `on:`
- Describes the action that will trigger the workflow.
- This template workflow is triggered by:
- a `push:` to the master branch, `branches: [ "master" ]`, or
- a `pull:` from the master branch, `branches: [ "master" ]`
- `workflow_dispatch:`
- This line allows us to run this workflow from the **Actions** tab in GitHub.

- `jobs:`
  - The **jobs** section is the heart and soul of the workflow.
  - This section defines the tasks that this workflow will run when triggered.
  - The **jobs** section is made up of one or more tasks that can run one after the other or at the same time. This workflow has **one** job named `build:`.
- `runs-on:`
  - Defines the runner the job will execute with. Our template runs with the latest version of Ubuntu Linux, `ubuntu-latest`.
- `steps:`
  - Steps are a list of tasks that will be done in order as part of the job.
  - `uses:`
    - This line gives our job access to our repository, by checking out our repo under the variable `$GITHUB_WORKSPACE`
  - `name:`
    - Defines the name of the task.
  - `run:`
    - This section defines each command that will run as part of this task

We can see that the first step in the `build:` job is named `Run a one-line script`. This step `runs:` the command `echo Hello, world!`

- **Rename this step from `Run a one-line script` to `Start workflow message`.**

- **Remove the command that currently runs for this step and replace it with the code snippet below.**

```
echo "Now running build workflow."
```

We can also see that the second step in the `build:` job is named `Run a multi-line script`. Notice that this step includes a `|` after the `runs:` indicator.

This step `runs:` the commands:
`- echo Add other actions to build`
`- echo test, and deploy your project.`

- **Rename this step from `Run a multi-line script` to `Ending workflow message`.**

- **Remove the command that currently runs for this step and replace it with the code snippet below.**

```
echo "Now ending build workflow."
echo "Have a great day!"
```

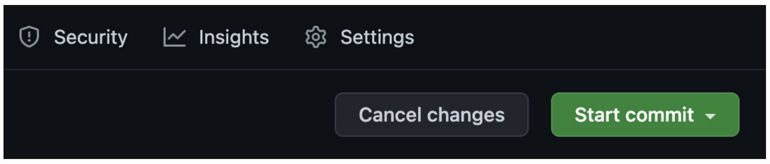
important

## Indentation

The indentation between the lines is important. When we see a dash `-`, it means we're making a list and all of the items below are included in that list.

We're going to add this to our repository now. In the upper-right portion of the screen, we should see a button that says **Start Commit**.

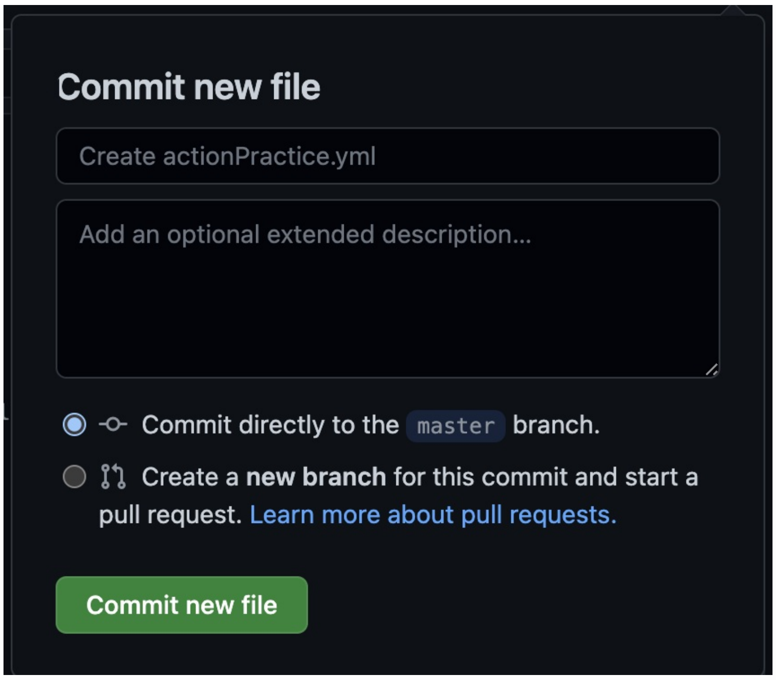
- **Click `Start Commit` to begin the commit process**



The start commit button in a GitHub repository.

This opens a window for us to write our commit message as well as add any extra descriptions or define which branch to commit this file to.

- **Write a commit message and click `Commit New File`**


Commit new file

Create actionPractice.yml

Add an optional extended description...

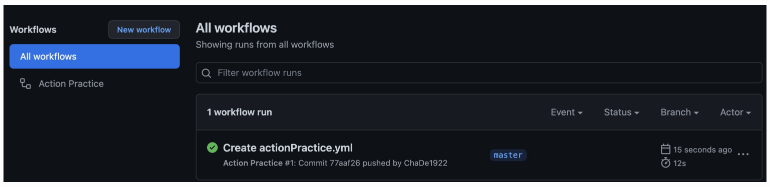
Commit directly to the `master` branch.

Create a **new branch** for this commit and start a pull request. Learn more about pull requests.

Commit new file

When we do this, it will send a **push** to our repository, which will trigger our Action.

If we click on the **Actions** tab, we can see that **Action Practice** can now be found under **All Workflows**.


Workflows | New workflow

All workflows

Action Practice

All workflows

Showing runs from all workflows

Filter workflow runs

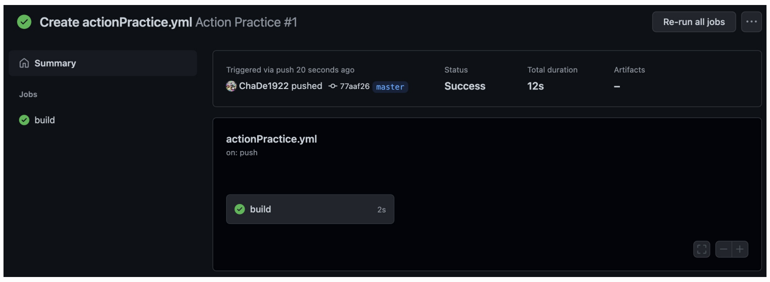
1 workflow run | Event | Status | Branch | Actor

Create actionPractice.yml

Action Practice #1: Commit 77aaf26 pushed by ChaDe1922 `master` 15 seconds ago 12s

The Action Practice workflow listed under All Workflows in a GitHub Repository

We can see that it’s running the action, and if we click on the action, we can see what **job** our action doing right now. We can click on our workflow to see that we are currently running the `build` job.


Create actionPractice.yml Action Practice #1

Re-run all jobs

Summary

Jobs

build

Triggered via push 20 seconds ago: ChaDe1922 pushed 77aaf26 `master`

Status: Success

Total duration: 12s

Artifacts: –

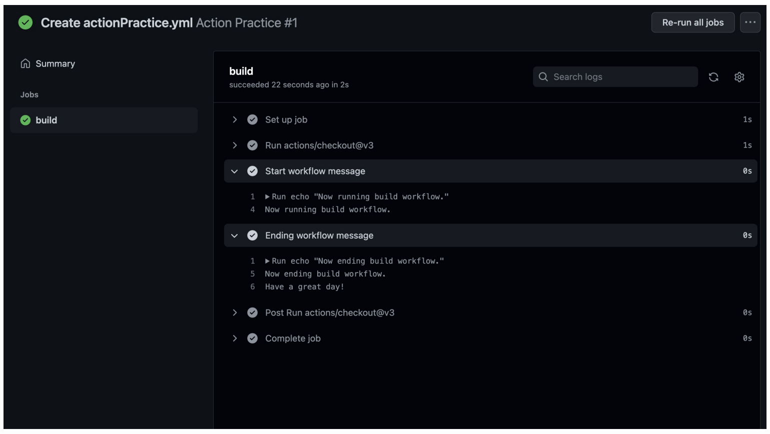
actionPractice.yml

on: push

build 2s

The build job listed under the Action Practice workflow on GitHub.com

If we click on this job. We can see each task that this job has carried out, the results of that task, and the success status of each step.



Create actionPractice.yml Action Practice #1

build
succeeded 22 seconds ago in 2s

```
Set up job                                   1s
Run actions/checkout@v3                      1s
Start workflow message                       0s
  ▶Run echo "Now running build workflow."
  Now running build workflow.
Ending workflow message                      0s
  ▶Run echo "Now ending build workflow."
  Now ending build workflow.
  Have a great day!
Post Run actions/checkout@v3                 0s
Complete job                                 0s
```

Tasks listed under the build job listed under the Action Practice workflow on GitHub.com

This is just one example of how we can use existing templates to customize our own GitHub Actions.

Checkpoint

Events

Now that we've used a workflow, let's look at some of the ways they can be triggered. There are many different **events** that can be used to do this when we're working with an action.

- **Workflow Events**
    - This means something like a `push`, `pull` or `fork` that signals a change to the repository.
- **Webhook Events**
    - These refer to an action that has occurred on GitHub. This could be someone marking your repository as a favorite, or changes to the Wiki page occur.
- **Schedule Events**
    - These events enable us to use cron syntax to specify when and how often an action occurs.
- **External Events**
    - We can also have an event triggered by an **external process** outside of GitHub called a **repository dispatch event**. This is what we'd use if we needed an external service to interact with our GitHub repository.

There are also a few specifiers we can use to further define how events are to run in our workflow.
- **Single or Multiple**
- **Types**
- **Editor Helper**

Let's go back and edit our action to understand these a bit better.

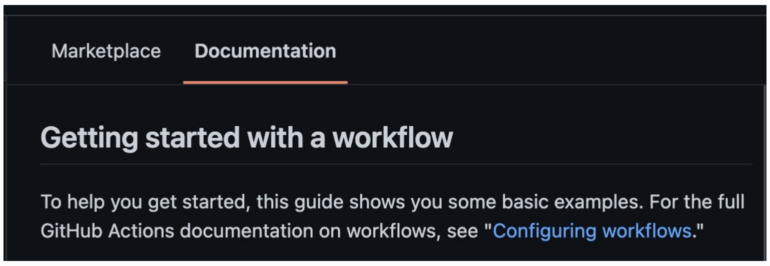
- **Navigate to the `Actions Practice` Action we created in the last exercise.**

- **Click the three dots on the right of the action and select `View workflow file`.**

- **Click the pencil in the upper-right area of the workflow window to edit this file.**

This will open our workflow file, allowing us to edit.

- **On the right side of the screen, click the tab labeled `Documentation`.**



Getting started with a workflow

To help you get started, this guide shows you some basic examples. For the full GitHub Actions documentation on workflows, see "Configuring workflows."

Screenshot of the workflow documentation tab

The workflow documentation provides us with simple examples to help us get started with creating our custom actions.

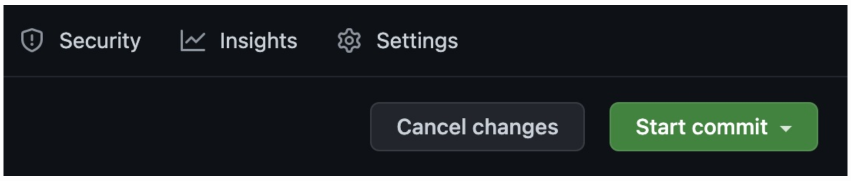
Currently, in the `on:` section, we can see that our action is **triggered by** either a `push` or a `pull_request` event to the master branch of our repository.

- **Add the following line to issue a completion message after the action.**

```
on:
  push:
    branches: [ "master" ]
  pull_request:
    branches: [ "master" ]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Output a message
        run: echo "Action Complete!"
```

- **Commit this change to the repo by clicking the `Start commit` button, creating a commit message, and clicking `Commit New File`.**

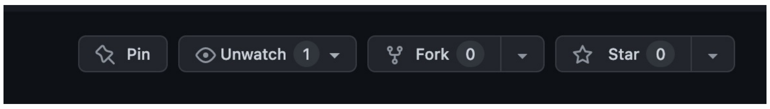


If we navigate to the `Actions` tab at this time, we should see that our action has been triggered due to the commit creating a `push` to the repository. Let's `fork` our repository to see that trigger in action.

- **Navigate to the `Code` tab of our repository toolbar.**

- **In the upper-right area of the screen, click the `Fork` button and**

select `Create a New Fork`.

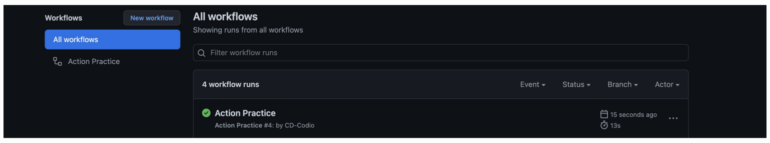


Screenshot of the fork button in a GitHub repository.

- **Customize the name of this forked branch and click `Create Fork` to copy our repo.**

Let’s check our Actions tab to confirm that our `fork` trigger was successful.

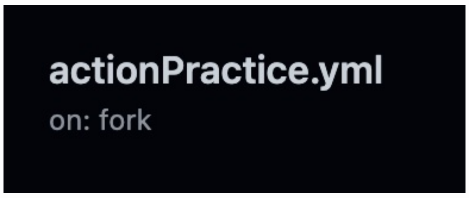
- **Navigate to the `Actions` tab of the repo toolbar.**



Screenshot of the actions screen in the repository

We should see that our Action has now been triggered by the `fork`. We can confirm this by clicking on the job and checking just below the `.yml` filename.

- **Click on the `Action Practice` workflow run to confirm the `fork` trigger.**



actionPractice.yml
on: fork

There are a lot of options for Action triggers to choose from. The **“Events That Trigger Workflows”** page is valuable resource for all of the different event commands that are available for us to use to trigger our workflows. It includes a list of valid commands, syntax, and examples to help us encorporate triggers into our workflow.

Checkpoint

Create github action

Github Actions are stored in `.yaml` files, which are used to store configuration settings, letting us run code on any Github event, such as a push or pull request.

Take a look at the `example.yaml` file to the left.

In this example, we use Github Actions to build a Style Checker for Github.

We start by defining the `name` of the action as well as the event that triggers this action

```
name: Check style
on: push
```

Next, define the job section that contains the commands we want to run when triggered.

```
jobs:
  build:
  runs-on: ubuntu-latest
```

This example just includes one job, but jobs are independent and can be run simultaneously.

For example, if we are preparing builds for many platforms, we can use multiple roles to generate builds for Linux, Windows, and Macintosh. Because our program is cross-platform and cross-browser, we can use only one for now.

Now, we're going to define the steps section, which is a list of steps/tasks that the job will run.

We, first, checkout a copy of your Github repository at the version we want.

```
  steps:
    - name: Checkout
    uses: actions/checkout@v3
    with:
      fetch-depth: 0
```

Now, we can add our step for our style checking command.

```
- name: check style
run: npm style
```

By using GitHub Actions, we can easily automate portions of our CI/CD software development process.

Checkpoint