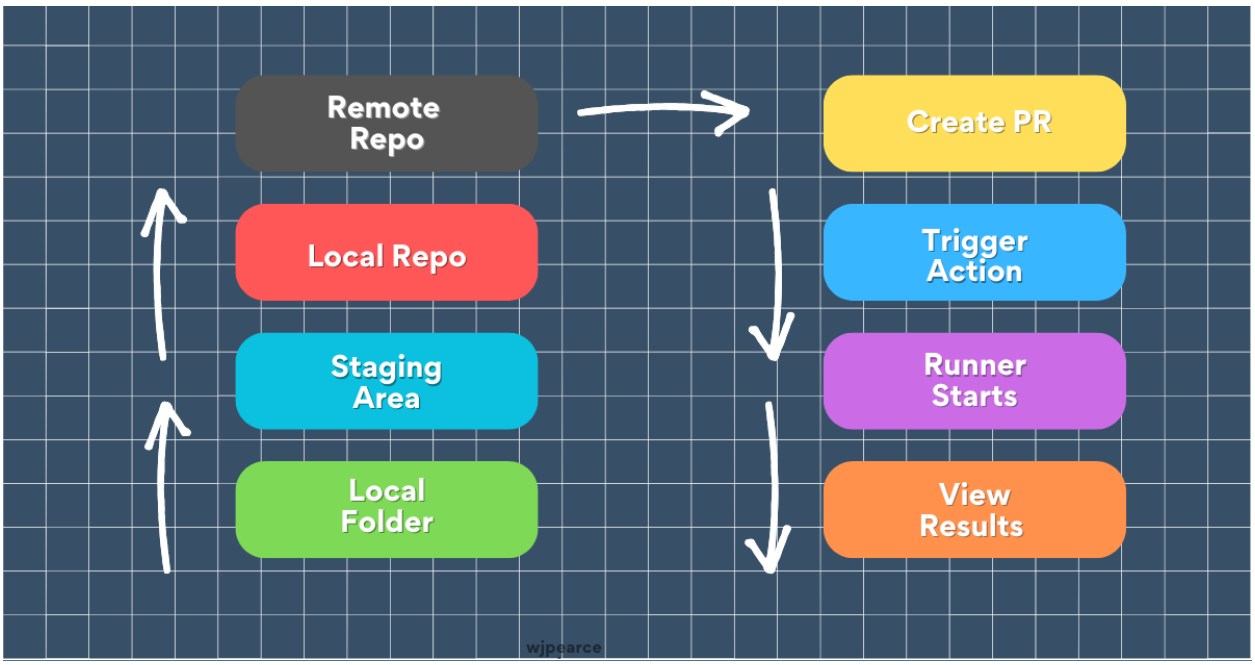
Project: Secure Your Code with GitHub Actions

By the end of this project you will know:

- What are Github Actions
- Why we use them
- How to set up a GitHub Action that uses an open-source tool (Gitleaks) to automatically scan the code in your repository for potential security issues.

This project encapsulates DevSecOps at its very core... *"How can we inject security into the Development Pipeline?"* Now of course, DevSecOps is a huge umbrella term that takes years to master but it's projects like this that are going to put you on a really good path for learning and get you thinking in all the right ways.

Here's a super high-level of what we are trying to achieve here from a Github point of view:



The year is almost over

There's still time for one more project you can complete over a weekend with just a few spare hours. If you want to know the basic of Github, I recommend checking out this chapter from my book - What is Github?

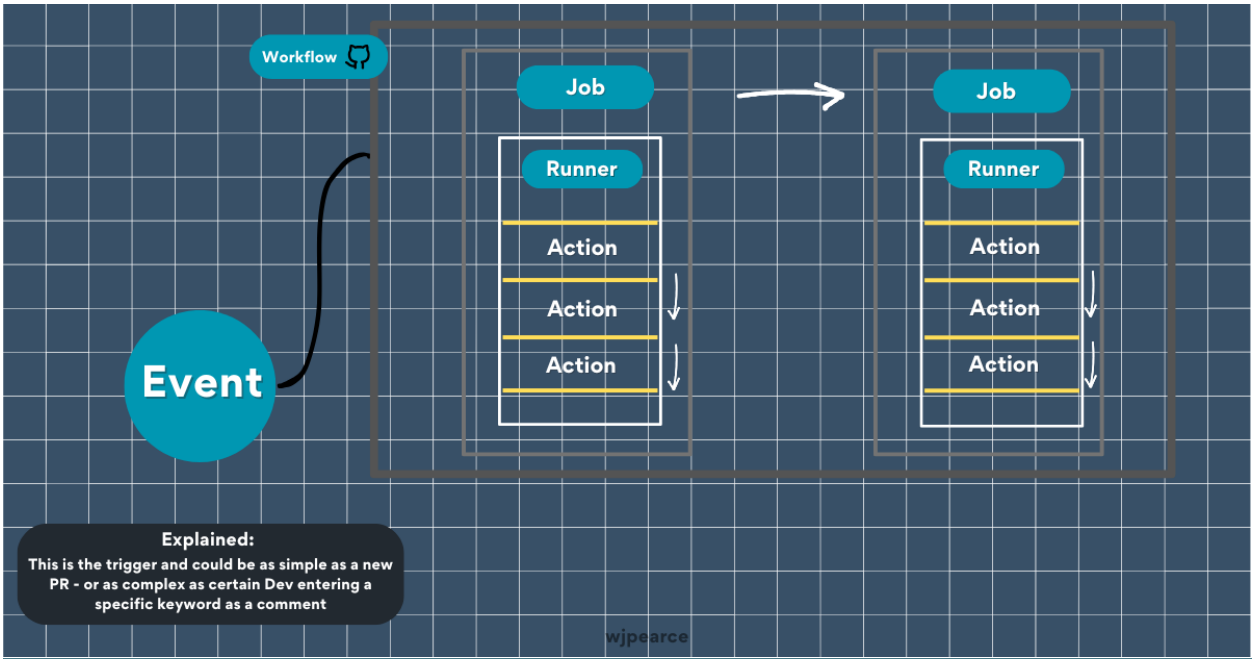
What are Github Actions?

GitHub Actions let you automate tasks directly within your GitHub repositories. Think of them as mini-robots (Runners) that perform specific jobs for you, like running tests, deploying code, or checking for vulnerabilities.

How do they work?

They use **workflows**, which are like recipes made up of individual steps (called **jobs**). These workflows run automatically when specific events happen - like when you push code or open a pull request - or on a schedule.

I've put together this diagram to help digest it:



Real-life Example

Let's say you're working on a website. GitHub Actions could:

- Test your website code every time you make changes.
- Deploy the updated version to the cloud if the tests pass.
- Run a security scan to make sure there are no vulnerabilities.

Why use GitHub Actions?

- Saves time by automating repetitive tasks.
- Reduces human error with consistent workflows.
- Helps integrate security into your workflow

Project Time

Okay, hopefully by now you have a decent grasp of what Github Actions are conceptually. Let's put that knowledge to good use and create our own Github Action.

Step One: Installing what you need

First, we need Git:

For most Ubuntu and Debian-based distro's, you want to run:

sudo apt-get install git

```
sudo apt-get install git
```

For Windows, the set-up wizard will do most of the work and guide you through the setup process, it's really simple! - **Download Here**

For macOS, Install homebrew, if you don't already have it, then:

brew install git

```
brew install git
```

Next, you need to sign up for a GitHub account which can be done here: github.com

---

Step Two: Repository set-up

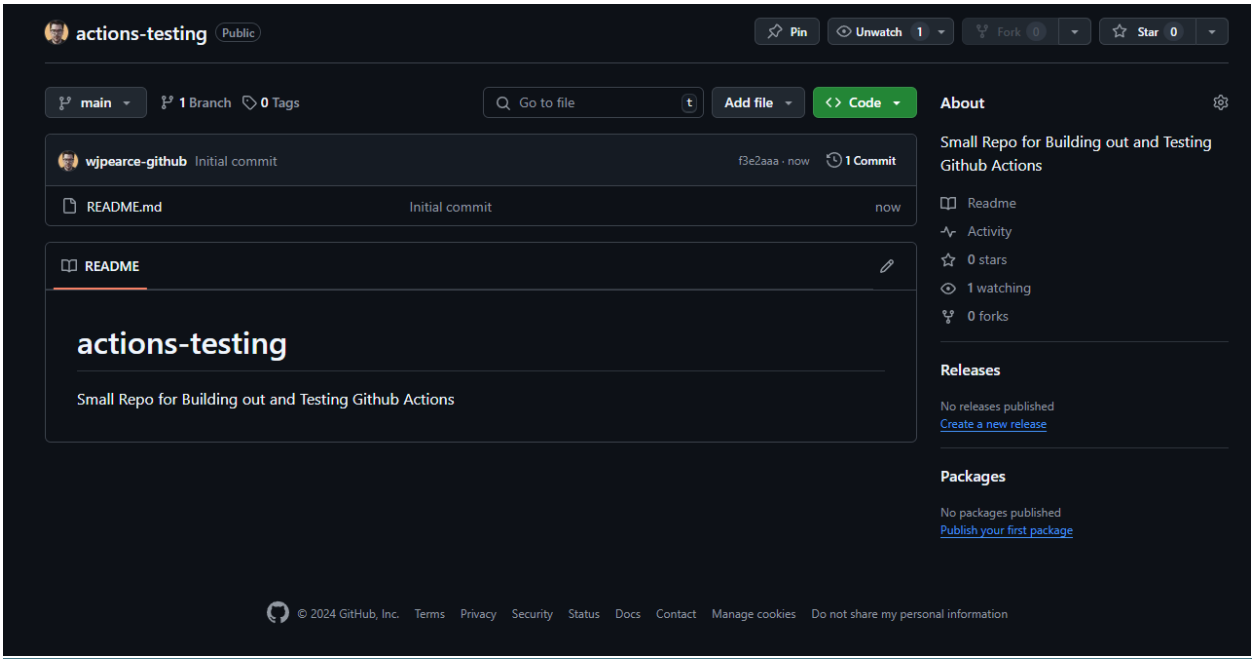
I have a feeling most of my readers will know how to do this already and I don't want to waste your time.

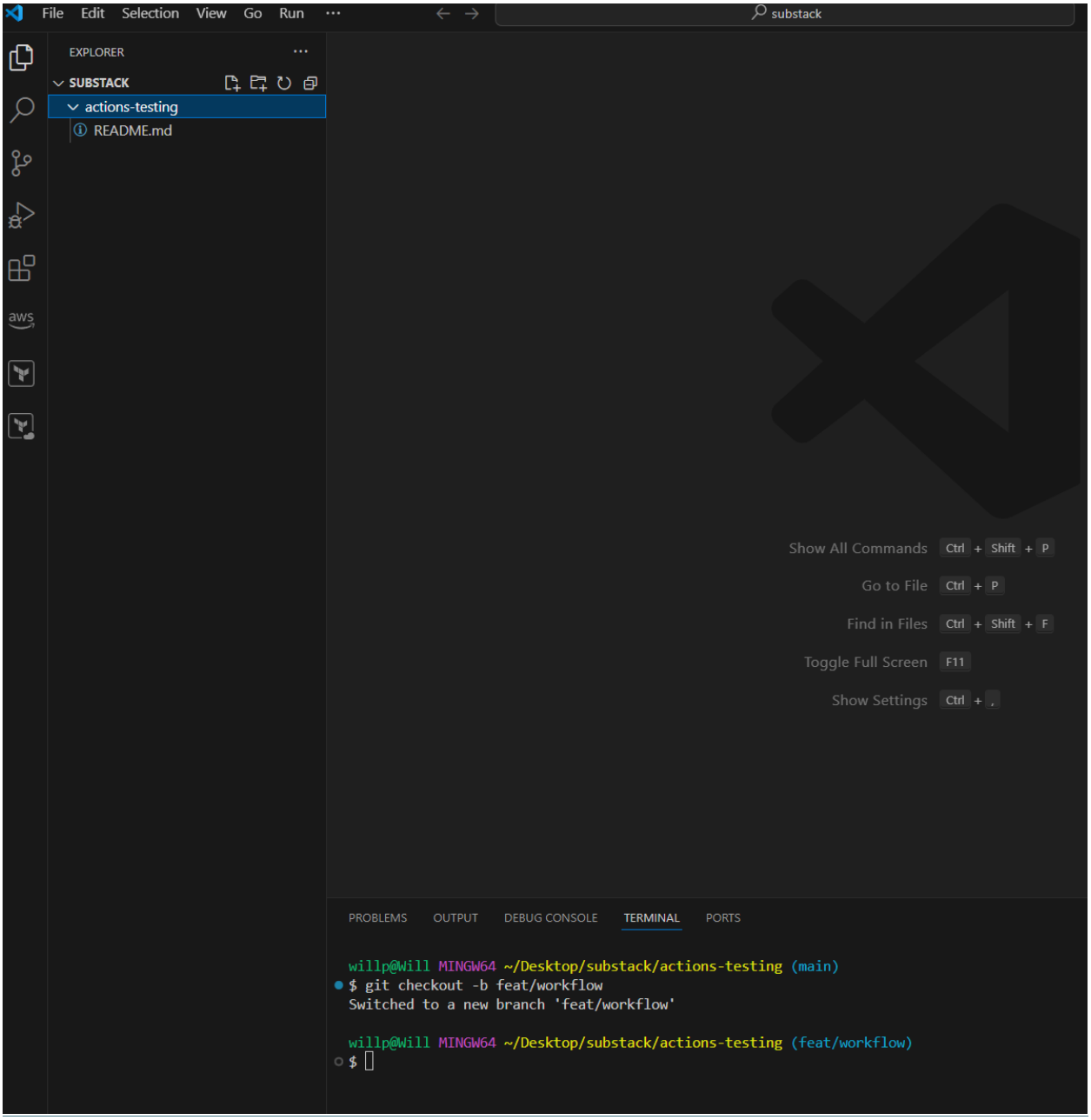
I'm going to link another article of mine here. It takes you right from creating a Github account to making your first PR

- Click me to get setup with Git and make your first PR

But effectively at this point you should have:

- A new Repo on Github
- README.MD File
- Cloned locally
- A new branch created ready to add files - Like this:





Here's where it gets interesting...

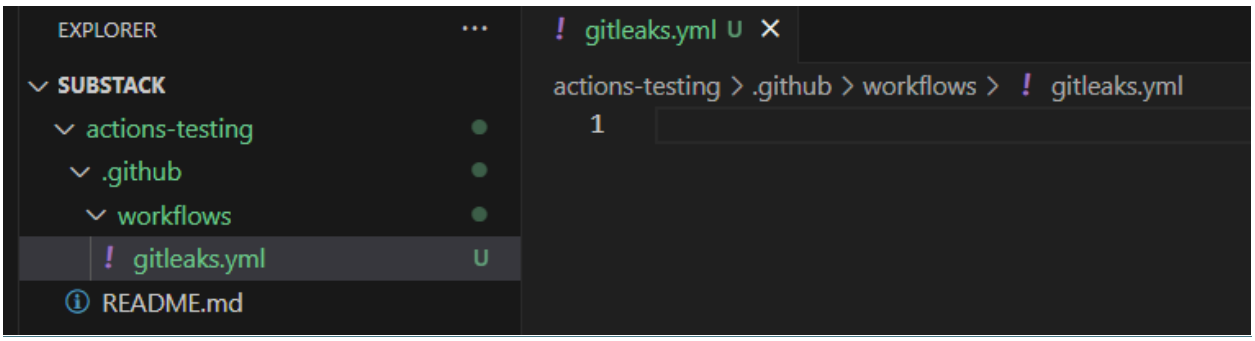
Step Three: Let's create our first Github Action - I'll include a link to my Github here ***(I've created a new one for substack projects)***

Github: Click Me

Step Three: Let's create our first Github Action

Back on your local device, setup two new folders and a file like this:

.github/workflows/gitleaks.yml



We now need to create our action in the gitleaks.yml file:

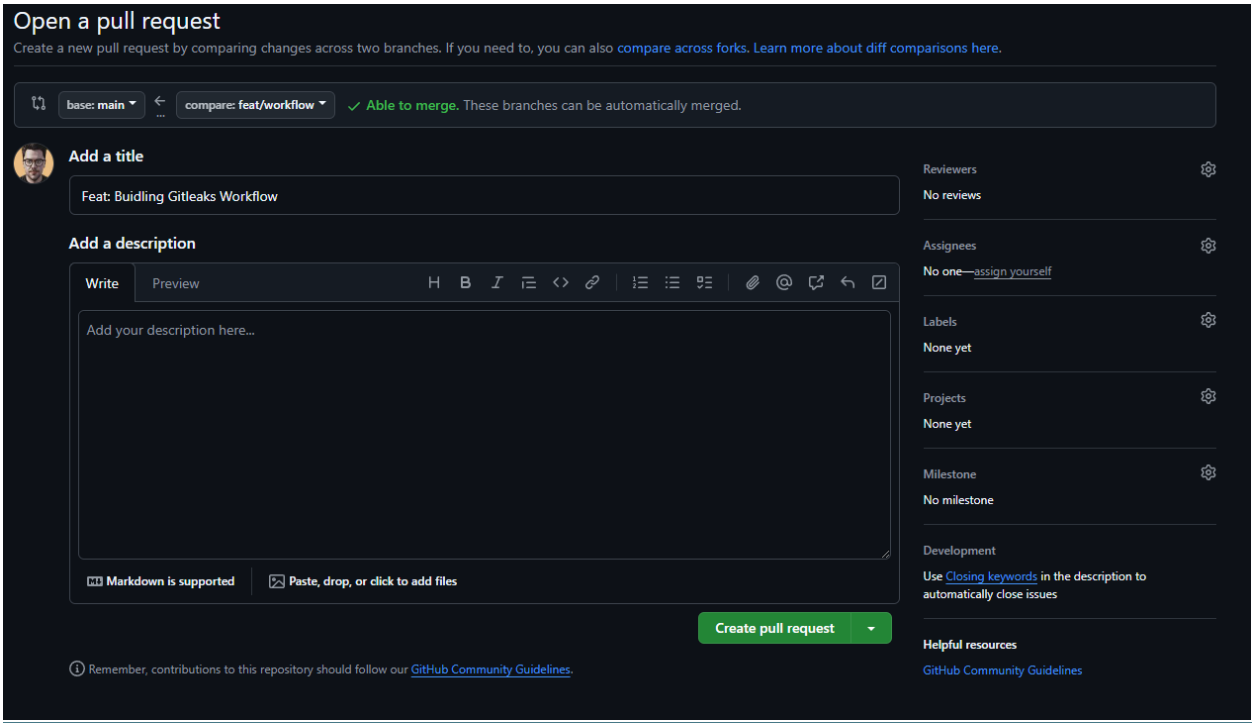
```
name: Gitleaks Scan # Workflow name.

on: # Events that trigger the workflow.
  pull_request: # Trigger on pull requests to 'main'.
    branches:
      - main
  push: # Trigger on pushes to 'main'.
    branches:
      - main
  workflow_dispatch: # Allow manual workflow runs.

jobs:
  scan:
    name: Run Gitleaks # Job name.
    runs-on: ubuntu-latest # Use Ubuntu for the job.

    steps:
    - name: Checkout code # Get the repository code.
      uses: actions/checkout@v3
      with:
        fetch-depth: 0 # Fetch full history for git diff.

    - name: Run Gitleaks Action # Scan for secrets using Gitleaks.
      uses: gitleaks/gitleaks-action@v2
      with:
        args: git --log-opts $(git merge-base origin/main HEAD)..HEAD --report-
format sarif --report-path gitleaks.sarif
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }} # Authenticate with GitHub.

    # Optional: Upload results to GitHub Security Tab.
    # - name: Upload Gitleaks SARIF Report
    #   uses: github/codeql-action/upload-sarif@v3
    #   with:
    #     sarif_file: gitleaks.sarif
```

I've commented on the code to help break it down at each stage. Feel free to re add back in the last block, it's cool for Github native security features but that's for another time.

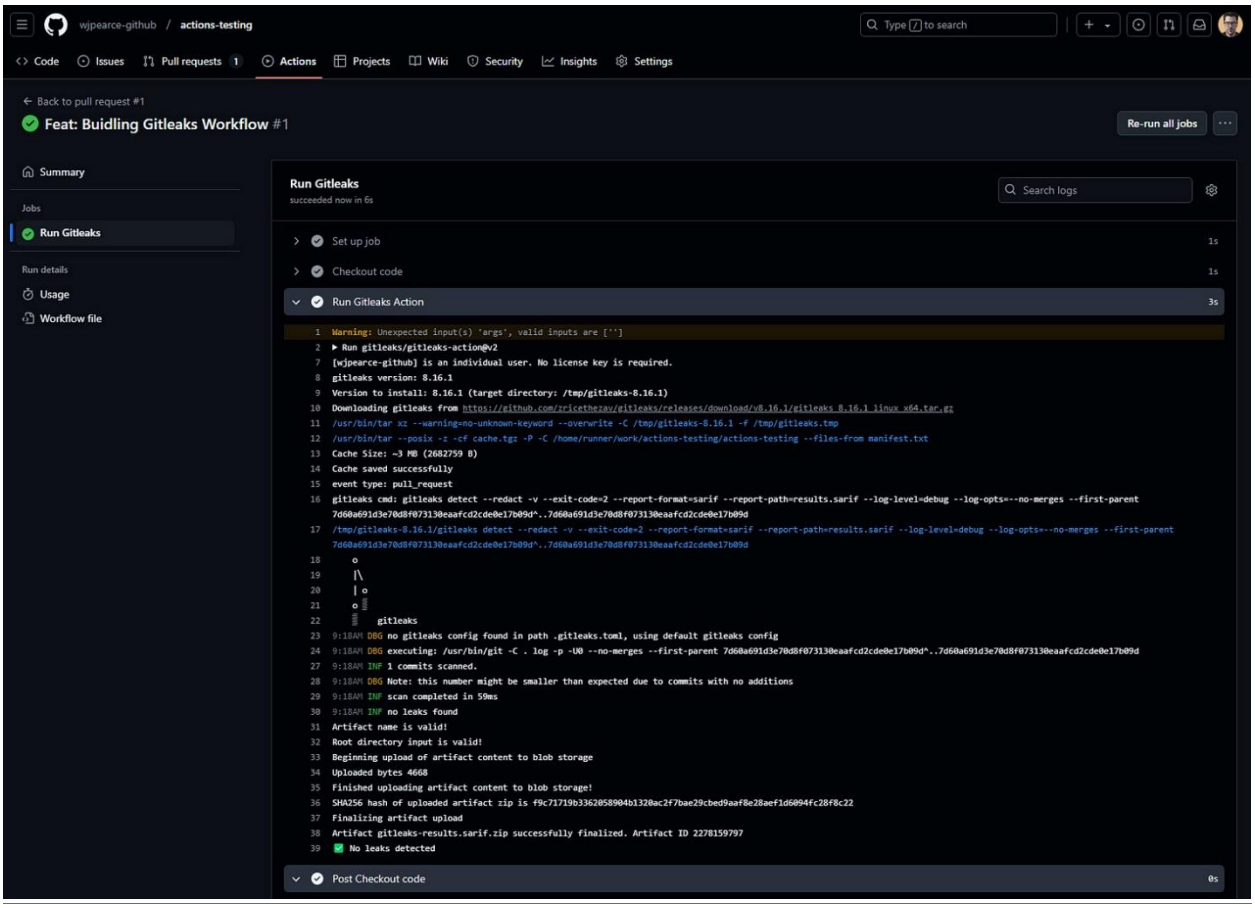
Save this, use git add, commit your code and then push - Like this:

```
git add .
git commit -am "Feat: Building Gitleaks Workflow"
git push --set-upstream origin feat/workflow
```

Back on Github we now want to create a pull request and check it's working as expected:



Once the PR is created this will trigger our Job, so navigate over to the Actions page and take a look.



As you can see, Gitleaks has been installed and run as expected - Now let's put it to work!

We can now merge this into our main branch.

Step Four: Finding Sensitive Data

Back on your local device, make sure everything is up to date and create a new branch

git checkout -b feat/secrets-test

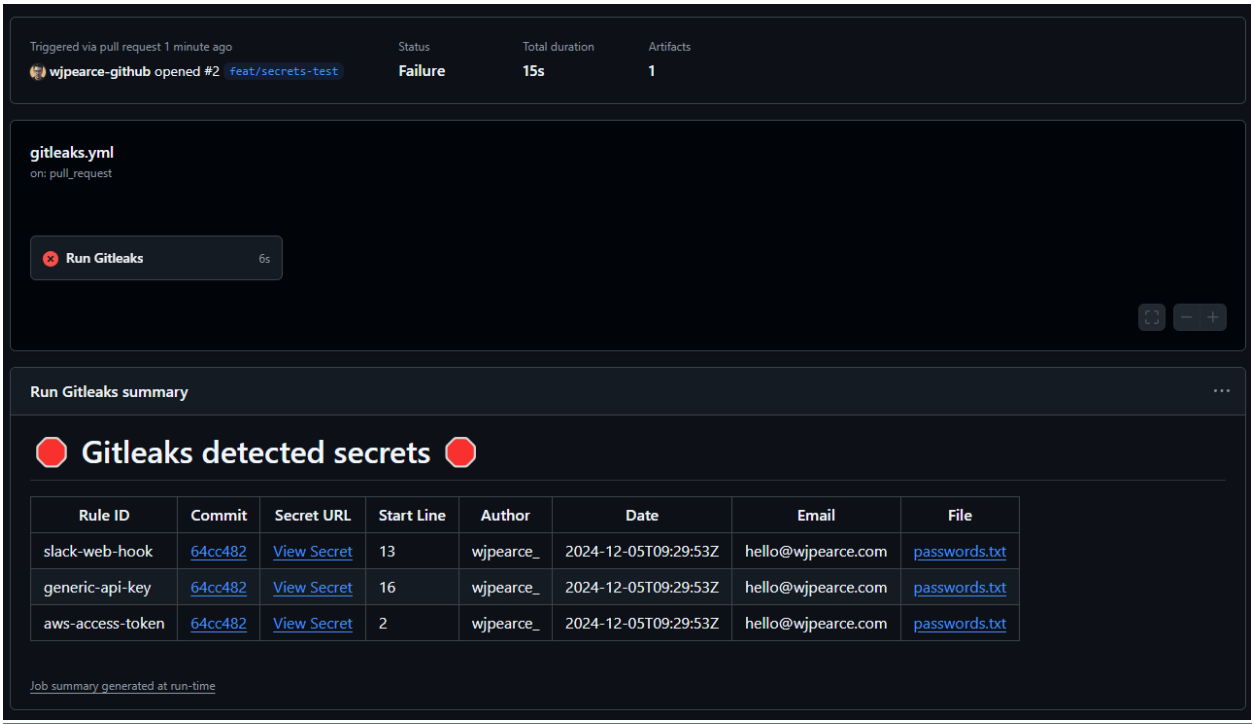
Add in a new file called passwords.txt and fill it with dummy sensitive data.

```
# Test Secrets
AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

# GitHub Token
GITHUB_TOKEN=ghp_12345EXAMPLetoken

# Database Credentials
DB_PASSWORD=supersecretpassword123
DB_USER=admin

# Slack Webhook URL
SLACK_WEBHOOK=https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXX
XXXXXXXXXXXX

# API Key
API_KEY=1234567890abcdef1234567890abcde
```

Save this file, push it up to your repo and create a new PR. Now let's watch this magic happen.

This PR creation will have triggered our Gitleaks action, so back on the actions tab, let's take a look.

We can see the action has worked but is in a failed status - This is good! This means Gitleaks has found our data, so head over to the summary tab:



Congratulations 🎉🎉

You've just created a repo, deployed an action and introduced secret scanning! That's huge and really is the fundamentals to a whole new learning path. Take this and run with it. Build upon it, explore how you could hide this data, and what happens if we remove the PR? Can we add more security scanning here?

You get the idea - This basic project is a fantastic first step into the world of DevSecOps.