
A StirTrek 2025 Presentation
Lights,
Camera,
Github Actions!
A Non-Comprehensive Guide to Getting Your Code on the Big Screen
Start

Welcome to Github Actions.

GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline. You can create workflows that build and test every pull request to your repository, or deploy merged pull requests to production.

GitHub Actions goes beyond just DevOps and lets you run workflows when other events happen in your repository. For example, you can run a workflow to automatically add the appropriate labels whenever someone creates a new issue in your repository.

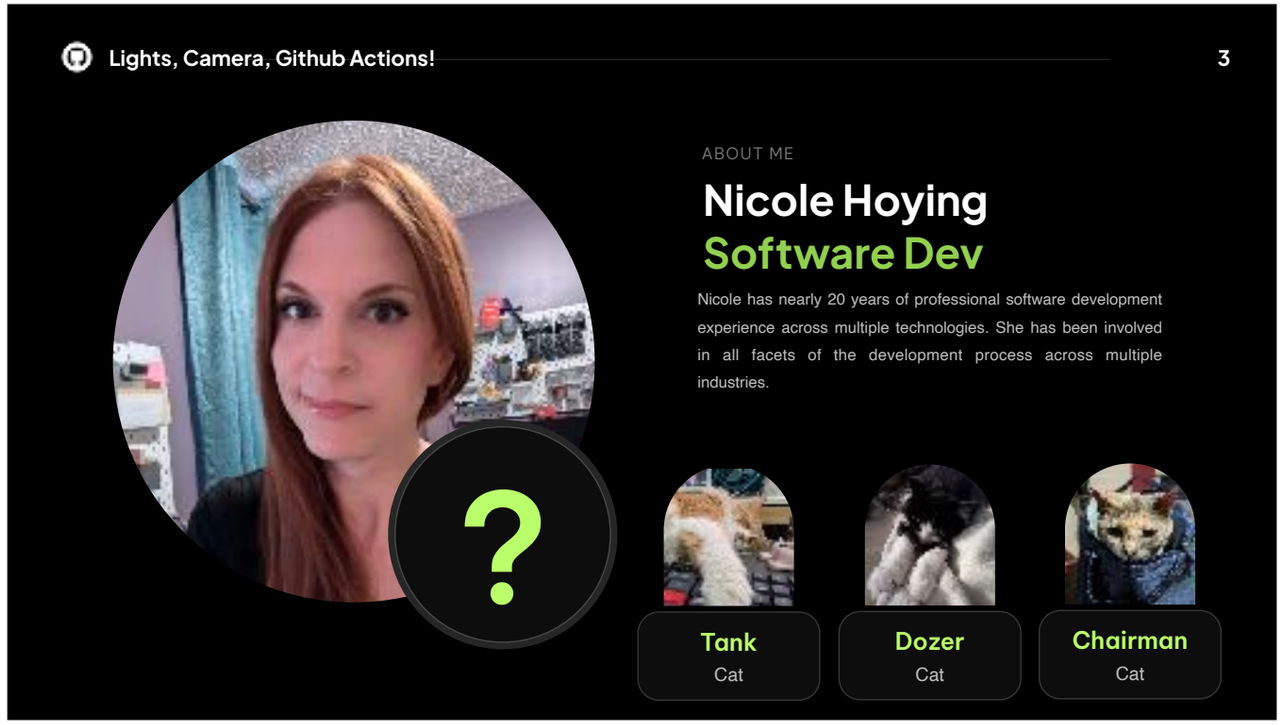
GitHub provides Linux, Windows, and macOS virtual machines to run your workflows, or you can host your own self-hosted runners in your own data center or cloud infrastructure.

https://docs.github.com/en/actions/about-github-actions/understanding-github-actions

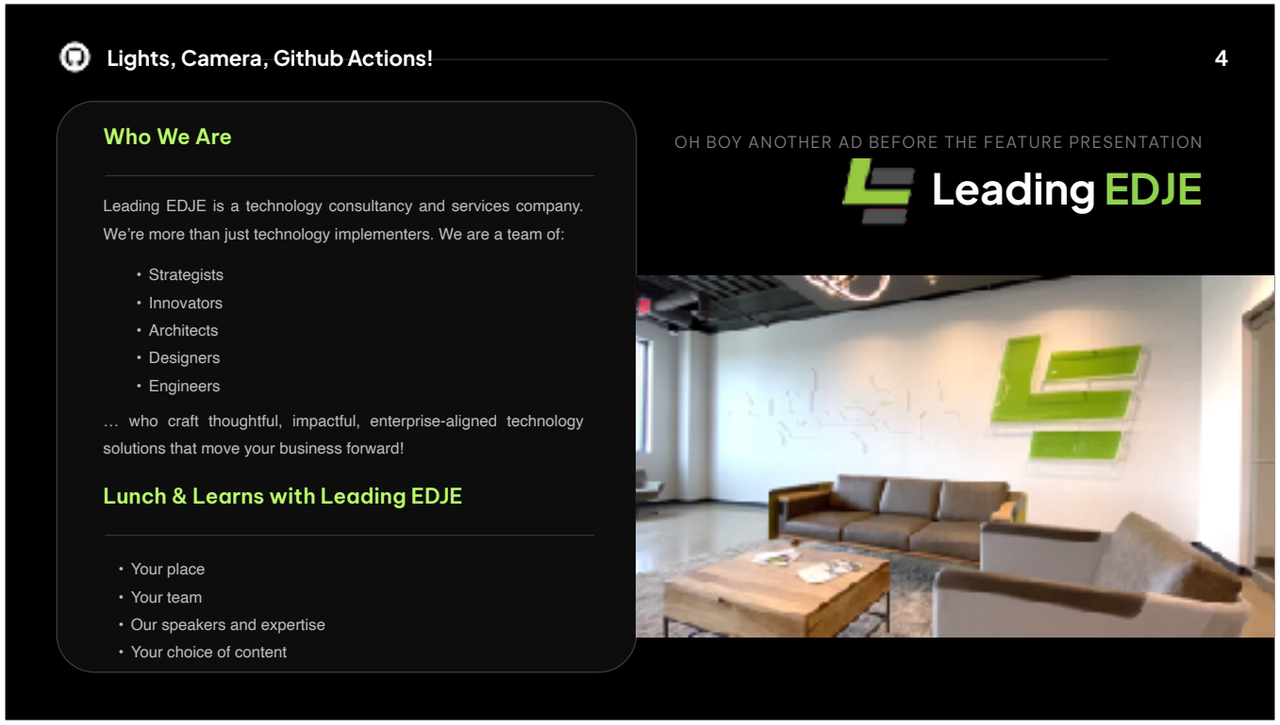
Talk about Github, Gihub Actions.

Ask the audience if they've used any CI/CD tools before like Azure DevOps, AWS Code Deploy, Bamboo, or GitLabs

We've got a lot to get through here because unlike Hollywood, Stir Trek frowns upon me splitting this out into 2 separate presentations so we can get that sweet, sweet sequel money.

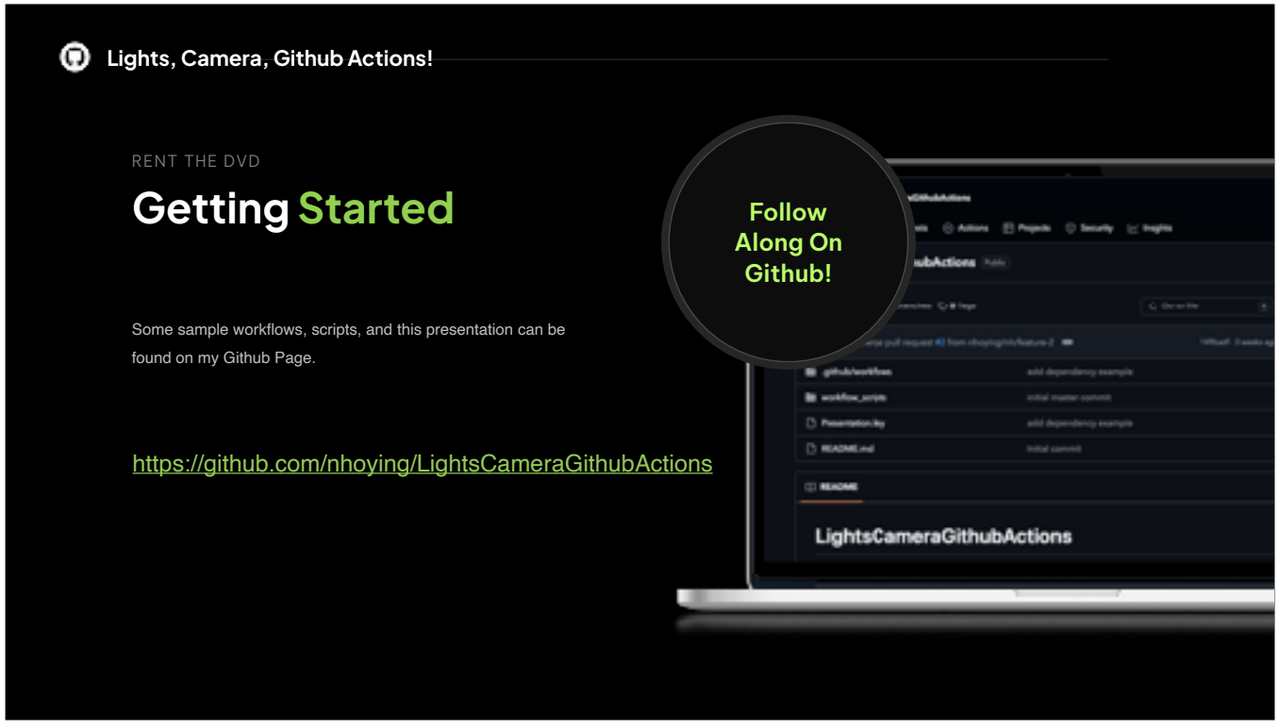


Talk about yourself, your history with CI/CD and GitHub actions. Explain how your are the unwitting protagonist.I Emphasize you haven’t been doing this very long but it’s pretty easy to pick up.



And because you can't start a movie these days without a bunch of ads beforehand, let me take a minute to talk to you about my company, LeadingEDJE.

At Leading EDJE, we believe technology is the most powerful tool for transforming businesses and unlocking human potential. As strategists, innovators, and partners, we don't just solve problems — we craft bespoke solutions that positively disrupt your business.



post this in the discord on your phone when you get here

Getting Started

- Make sure your repository supports actions.
  - Free Tier needs to be a public repository.
  - Make sure it's enabled in the settings.
- Workflows only show up in the Actions tab when they exist in the default branch.
  - Stub out a workflow and then merge it to the default branch before you actually start writing any real code.

Getting started is pretty easy!

Also mention that if you are using self hosted runners, you need to ensure the runner group is setup.
Mention how you can stub out a workflow to run on push but you don't personally like it.

SETTING THE STAGE

Laying It All Out

Stubbing out your first workflow

- The Three (Four) Top Level Components
  - name
    - This is what gets displayed in the UI
  - permissions
    - Optional. This defines the permissions for the workflow
  - on
    - The trigger condition
  - jobs
    - The individual jobs that are handled in the workflow

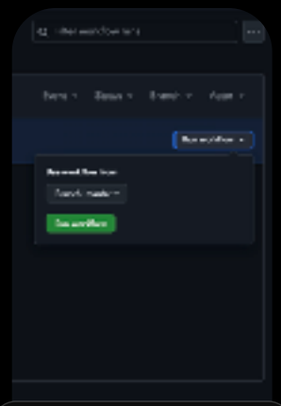
Bringing Your Vision to Life.

Stubbing out your workflow allows you to develop your action inside a feature branch without constantly having to check in to your default or trunk branch.

So lets set our stage. We want to create our first stubbed out workflow. We just need all of the top level components to allow us in the door.
Make sure to define stub. Just enough of a workflow to pass the syntactic checks.

Talk about if you define one permission, it sets every other one to false. Permissions are applied to the Github_token (if you are storing and using other tokens they are not affected) The github_token is what is used for the default credentials for every action step. (Certain actions allow you to override that token with a different one that you can provide)

Permission	Allows an action using GITHUB_TOKEN to
actions	Work with GitHub Actions. For example, actions: write permits an action to cancel a workflow run. For more information, see "Permissions required for GitHub Apps."
attestations	Work with artifact attestations. For example, attestations: write permits an action to generate an artifact attestation for a build. For more information, see "Using artifact attestations to establish provenance for builds"
checks	Work with check runs and check suites. For example, checks: write permits an action to create a check run. For more information, see "Permissions required for GitHub Apps."
contents	Work with the contents of the repository. For example, contents: read permits an action to list the commits, and contents: write allows the action to create a release. For more information, see "Permissions required for GitHub Apps."
deployments	Work with deployments. For example, deployments: write permits an action to create a new deployment. For more information, see "Permissions required for GitHub Apps."
discussions	Work with GitHub Discussions. For example, discussions: write permits an action to close or delete a discussion. For more information, see "Using the GraphQL API for Discussions."
id-token	Fetch an OpenID Connect (OIDC) token. This requires id-token: write. For more information, see "About security hardening with OpenID Connect"
issues	Work with issues. For example, issues: write permits an action to add a comment to an issue. For more information, see "Permissions required for GitHub Apps."
packages	Work with GitHub Packages. For example, packages: write permits an action to upload and publish packages on GitHub Packages. For more information, see "About permissions for GitHub Packages."
pages	Work with GitHub Pages. For example, pages: write permits an action to request a GitHub Pages build. For more information, see "Permissions required for GitHub Apps."
pull-requests	Work with pull requests. For example, pull-requests: write permits an action to add a label to a pull request. For more information, see "Permissions required for GitHub Apps."
repository-projects	Work with GitHub projects (classic). For example, repository-projects: write permits an action to add a column to a project (classic). For more information, see "Permissions required for GitHub Apps."
security-events	Work with GitHub code scanning and Dependabot alerts. For example, security-events: read permits an action to list the Dependabot alerts for the repository, and security-events: write allows an action to update the status of a code scanning alert. For more information, see "Repository permissions for 'Code scanning alerts'" and "Repository permissions for 'Dependabot alerts'" in "Permissions required for GitHub Apps."
statuses	Work with commit statuses. For example, statuses:read permits an action to list the commit statuses for a given reference. For more information, see "Permissions required for GitHub Apps."



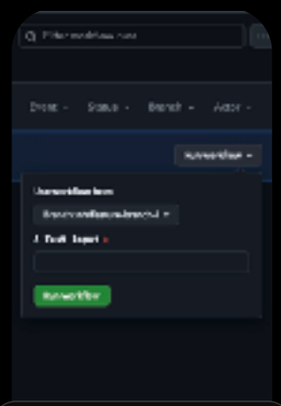
master
No Input

SETTING THE STAGE

Staging Your Workflow

Don't interfere with your team! Make sure you always develop inside a feature branch. There's almost no reason not to.

```
on:
  workflow_dispatch:
    inputs:
      test-input:
        description: A test input
        required: true
        type: string
```



feature
Has Input

Talk about how you can change the branch on the workflow. Illustrate how this change added an input and the input loads automatically when you switch the branch

TRIGGERS

Starting the Action

The on: keyword is used to define the trigger for a workflow. There are many different events that can fire off a workflow and each one has different caveats that you need to be aware of.

workflow_dispatch

Allows for a workflow to be triggered manually from the Actions tab of your repository.

push

Executes the workflow when you push a commit or tag to the repository

pull_request

Triggers the workflow when an activity happens on a pull request in the workflow's repository. Can be filtered by activity type.

Triggers are used to start the action in your workflow. They don't have anything as dramatic as say, having your aunt and uncle murdered by stormtroopers who are looking for some missing droids but they are equally important in getting your action rolling.

workflow dispatch triggers manually
push happens on ANY push action. You can use filters on this. Sometimes you may want to skip on push and you can use [skip_ci] to prevent the trigger from running
pull request will probably need additional filtering because it would trigger on ANY pr activity.

TRIGGERS

Starting the Action

The on: keyword is used to define the trigger for a workflow. There are many different events that can fire off a workflow and each one has different caveats that you need to be aware of.

schedule

Executes on a cron-based schedule. You can't use inputs with this but you can set outputs or env variables using github.event.schedule metadata

workflow_call

Allows for the workflow to be triggered from another workflow. Needed for Re-usable Workflows.

schedule can be used with cron
workflow call for when you want this to be a reusable workflow

IT'S ALL CONNECTED

Understanding Jobs

- A workflow job is a collection of actions running inside the same context.
- A job must specify a name, runner, and collection of steps.
- Steps within the job will all run in sequence on the same runner instance with the same context.
  - If a file is needed outside the job context in a different job, we can upload an artifact and download it later.

Now that we've determined our workflow's name and figured out how we're going to kick it off, we're ready to script out what our workflow is supposed to actually supposed to do.

Jobs are the core building block of a workflow.

Github has 3 built in runner types. Ubuntu, Windows, and MacOs.

Free gets 2000 minutes per month
Enterprise gets 50,000 minutes

Talk about how I burned through all of the client's credits testing the MacOS runners.

Talk about self hosted runners

IT'S ALL CONNECTED

Organizing Your Dependencies

needs: [job_name]

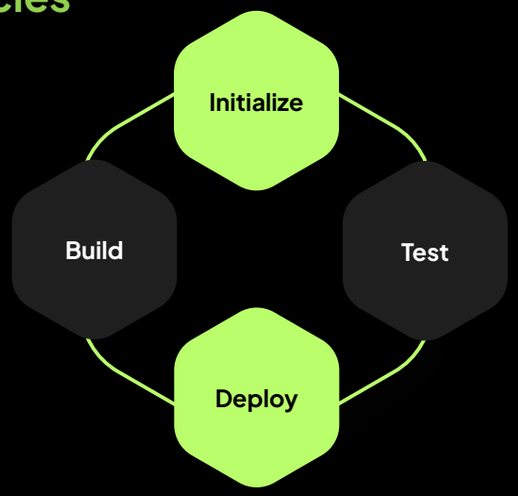
Use the needs definition to set pre-requisites for your jobs. If the dependency fails, it will not run the next job in the chain.

if: ${{ always() }}

Use in conjunction with needs so that the job will still run even if one or more of the prerequisites fail.

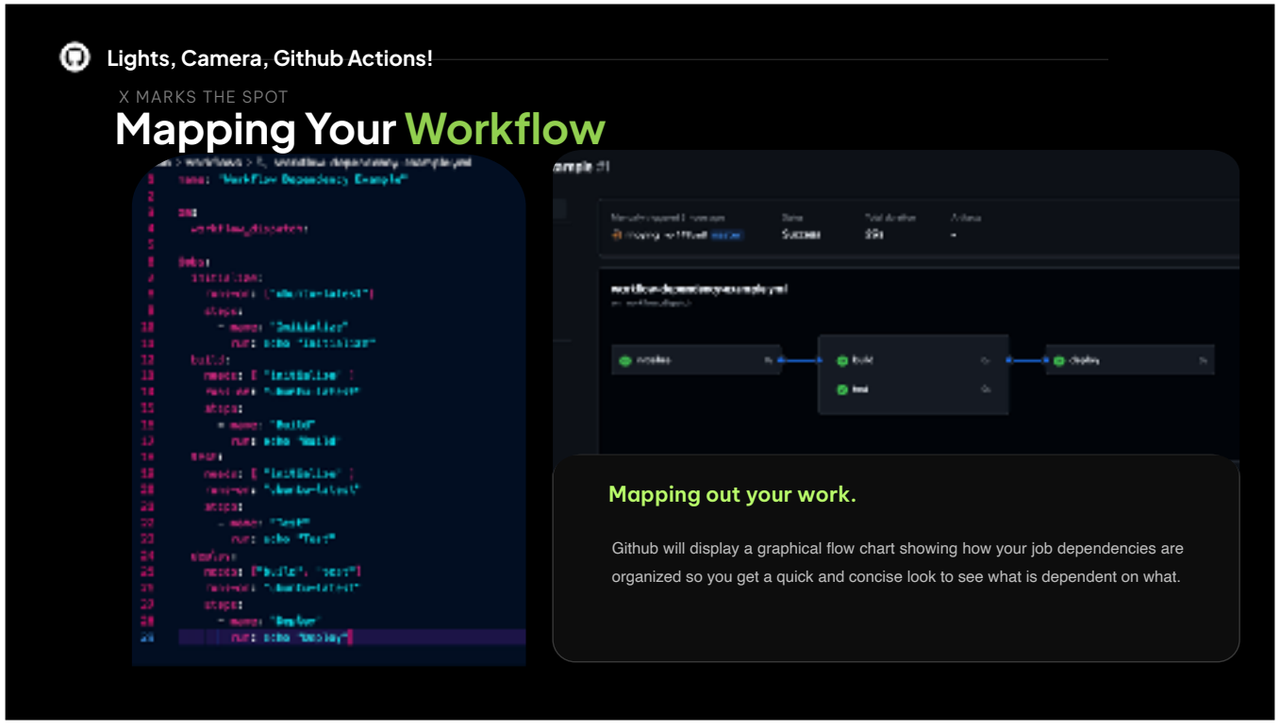
${{ success() | failure() }}

Runs if the job fails or succeeds. You can use the or operator to ensure that it will run in either state as a replacement for always().



Every movie needs an order to its acts for it to make sense to the audience. Github actions is no different. You can use these to structure your jobs so they run in the proper order.

Make sure to mention how if always can cause weird infinite looping

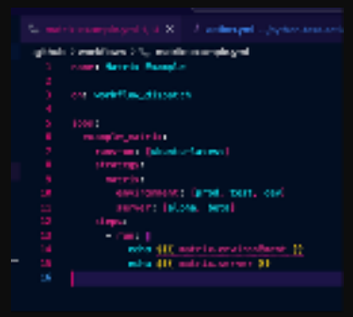


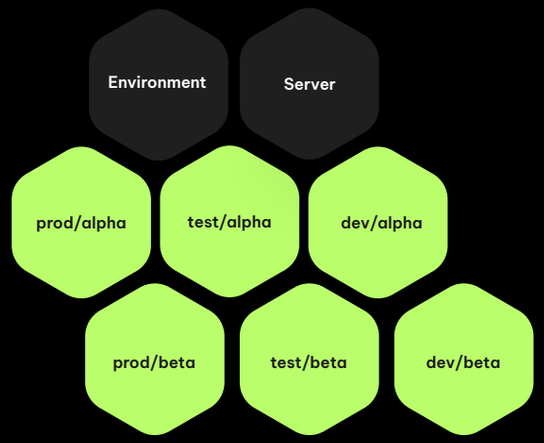
Make sure to draw the parallels between the workflow script on the left to the map on the right

FOLLOW THE WHITE RABBIT

Entering the Matrix

A matrix strategy allows you to specify variables and initiate a number of jobs based on the combination of those variables.





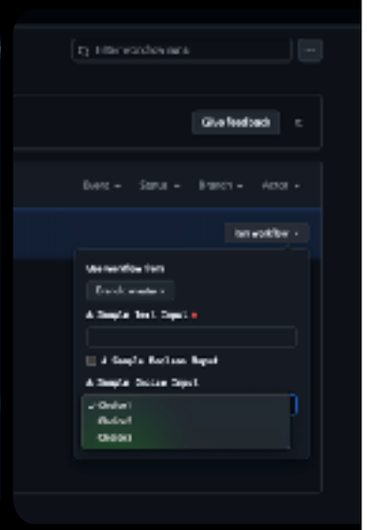
You can use concurrency groups to force these to run in order.
Talk about the SSIS job you wrote for client that needed to compile each dtproj file separately because of the CICD SSIS tool. Listed out the dtproj files in the repository and then generated a matrix to build the ispac files and write them as artifacts

INPUT, INPUT

Using Workflow Inputs

You can pass a number of inputs to your workflow to help customize it. You can use those inputs to set values in your workflow steps or use to it to base conditional logic from.

The types of inputs supported depends on the trigger event for your workflow.



Also mention input typing. When you dont specify a type, it is always a string input. So when you check for “booleans” its actually a string that says true or false

IT'S A SURPRISE TOOL THAT WILL HELP US LATER

Setting Outputs

Foreshadowing But In a Workflow

Values can be set to the $GITHUB_OUTPUT environment variable using the append (>>) operator in the bash shell. Those values can then be accessed in downstream steps and jobs by using the correct keywords. You can either reference the YAML expression directly in your shell script OR pass the value to an environment variable or other type of input for your actions.

Mention how easy it is to forget to define your outputs

FILMING THE ACTION

Getting Down to Business

Workflow Steps and Actions

01 Actions are pre-written building blocks. Github provides a number of built in actions under https://github.com/actions. You can also find actions in the Github Marketplace or define your own as a Composite Action

02 Actions run sequentially in your steps section under a job. Actions have access to the context of the current runner so you can do things like build your application and then package it up for deployment.

Don't forget to Log!

Make sure to write your actions with plenty of logging to help you debug any issues you have. You can always re-run a failed build with extra debug logging or you can enable debug logging to always being on by adding the variable ACTIONS_RUNNER_DEBUG to your repository and setting it to true.

Actions are like the scenes of your movie. an act comprises of several scenes

FILMING THE ACTION

Your Establishing Actions

actions/checkout

Checks out the code for a given repository. If not specified, it check out the code for the repository the action is being ran on. If you do specify a different repository, a token needs to be passed that has access.

actions/setup-x

Built in actions that will setup the runner with the build tools you need. Examples are setup-node, setup-dotnet, setup-java, setup-ruby, etc.

actions/github-script

Essentially a NodeJS scripting environment that is super powerful and has access to a number of GitHub specific APIs.

Talk about establishing shots and actions

FILMING THE ACTION

Your Establishing Actions

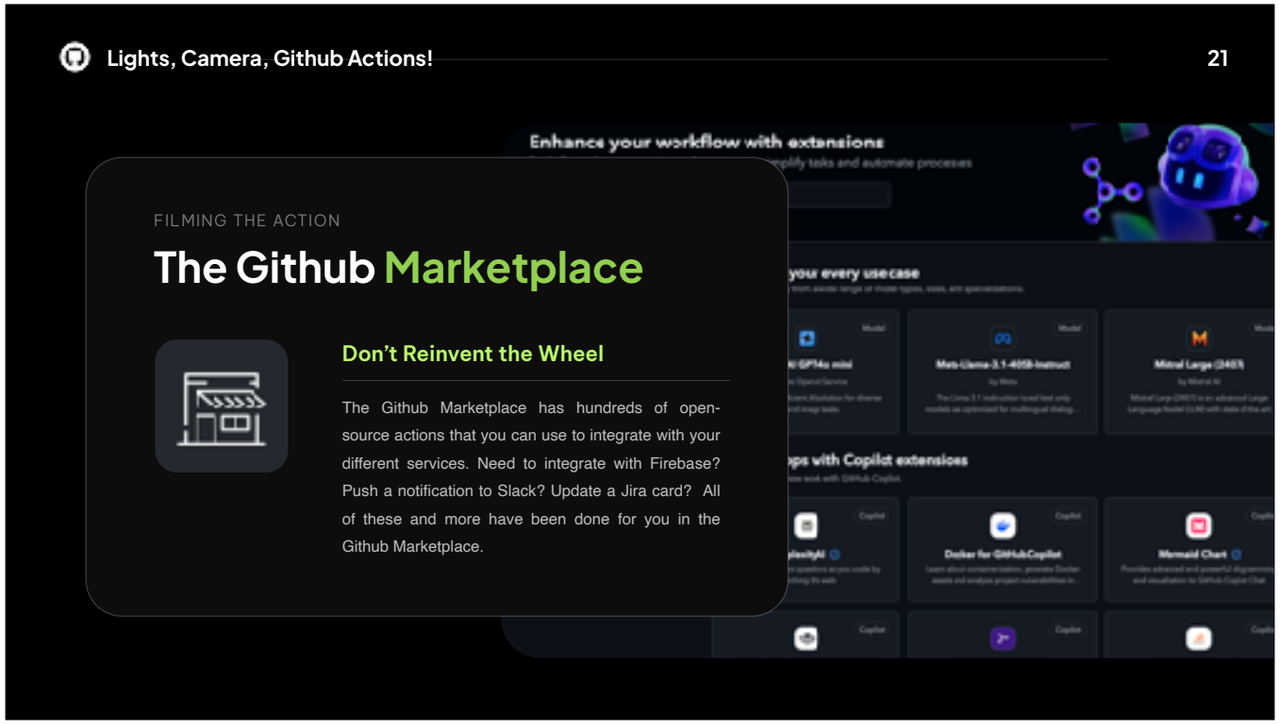
actions/upload-artifact

Uploads one or more files to the workflow's artifact cache which allows you to share those files with other jobs.

actions/download-artifact

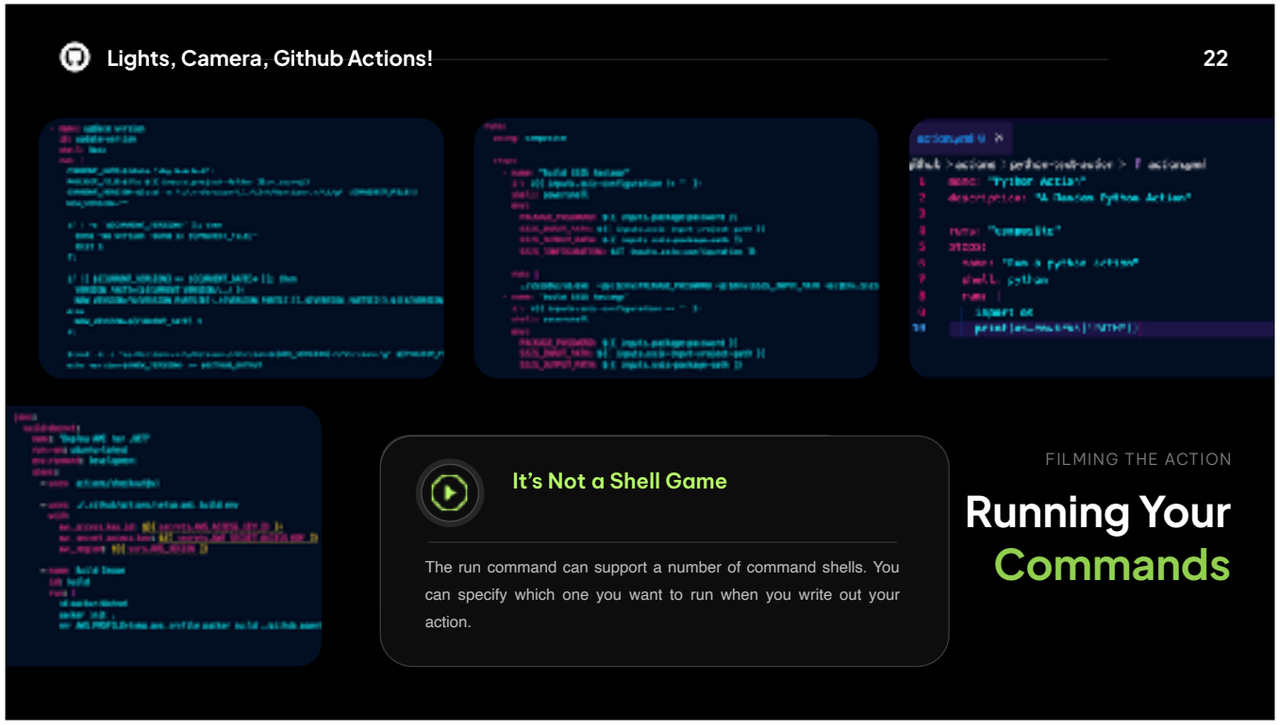
Downloads an artifact package from the workflow's artifact cache so you can use it in the current job.

Make sure to say that upload and download must be at the same version to work with each other



Discuss the Marketplace

Make sure to mention something about how your org admin has control over whether or not you can use these. (If the org admin won’t let you use them, GitHub marketplace actions are open source and you can always look at their source code to pick and choose what you need)



Need to say something about this is where your commands go or something

Need to make sure that the commands that actually run your builds and tests need to be spelled out as shell commands. Github does not give you actions that explicitly do this

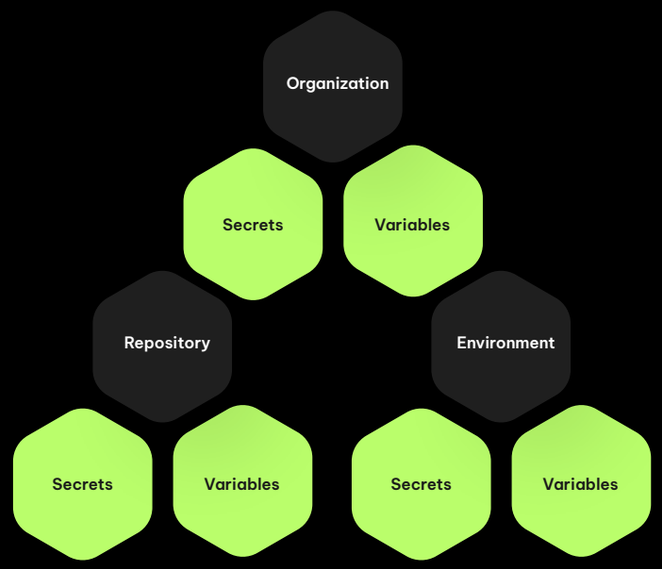
KEEPING IT UNDER WRAPS

Managing Secrets and Variables

A variable is stored in plain text and can be read in the UI. They are also displayed in the action logs.

A secret can only be stored in Github and not visibly retrieved. You do need to be careful with using them because there are ways to get them to show up in your action logs.

Secrets and variables can be stored at many levels. Organization is the highest level, followed by repository, then environment.



Make sure you say lowest level takes precedence

Also talk about how to pass secrets. you can define them like inputs on your workflows but if you pass a secret in as an input, it loses it secretness

SPECIAL EFFECTS

Github Scripts

DevOps But Make it JavaScript

Github Scripts allow for you to write Node.js enabled build scripts and allow you easy access to the Github API. In my opinion, this is the Killer Feature of Github.

github

A pre-authenticated octokit/rest.js client with pagination plugins. You can use this to execute any Github API call.

context

An object containing the context of the workflow run.

core

A reference to the @actions/core package. Core is used to set results, logging, registering secrets, and exporting variables across actions.

SPECIAL EFFECTS

Github Scripts

DevOps But Make it JavaScript

Github Scripts allow for you to write Node.js enabled build scripts and allow you easy access to the Github API. In my opinion, this is the Killer Feature of Github.

glob

A reference to the @actions/glob package. This package is used to search for files matching glob patterns.

io

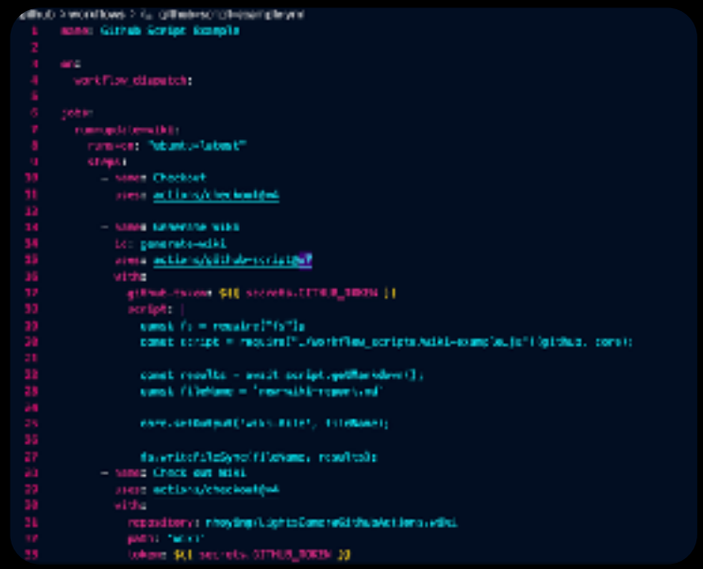
A reference to the @actions/io package. This package contains core functionality for cli filesystem commands.

exec

A reference to the @actions/exec package. Can be used to execute tools in a cross platform way. Helpful if your workflow needs to be used on different operating systems

Mention how these ones you know and have used less. io is used more for stuff like rm -f and other cli commands than like writing and reading files

Lights, Camera, Github Actions!



SPECIAL EFFECTS

Github Scripts

- Make sure you pass your GitHub token to the action.
- All of the javascript objects we talked about before are just “there”.
- Write your Javascript in a way that is testable.

Talk through what this workflow is doing. Elaborate on what “is testable” means

Lights, Camera, Github Actions!

workflow_scripts > JS wiki-example.js > <unknown> > exports

```javascript
module.exports = {
    github,
    core
} => {
    function getMarkdown() {
        const markdownLines = [];
        markdownLines.push("# Markdown Example");
        markdownLines.push(`### Run Date: ${new Date().toUTCString()`

        // table creation

        markdownLines.push("|Repository|Some Value|");
        markdownLines.push("|---|---|");

        markdownLines.push ("|Repo 1| Some Value|");
        markdownLines.push ("|Repo 2| Some Value|");

        return markdownLines.join('\n');;
    }
}
```

SPECIAL EFFECTS

Github Scripts

Explain whats going on in this javascript. Feel free to go back and forth between the two slides.

DON'T REPEAT YOURSELF

Composite Actions

A Composite Action is a collection of workflow job steps that can be reused in any workflow that references it.

01 Create the File

Composite Actions are stored in .github/actions/{action_name}/action.yml.

02 Fill out Action

A Composite Action can be written as a collection of any number of steps. All composite actions need a runs: using: composite flag.

.github > actions > output-test-action > ! action.yml

```yaml
name: Sample Output Composite Action
description: Sample Output Composite Action

outputs:
  sample_output:
    description: The Sample output
    value: ${{ steps.set_output.outputs.output_value }}

runs:
  using: composite
    - name: Set Output
      id: set_output
      shell: bash
      run: |
        echo "output_value=some_value" >> $GITHUB_OUTPUT
```

Talk about how Disney reused animation.

DON'T REPEAT YOURSELF

Composite Actions

A Composite Action is a collection of workflow job steps that can be reused in any workflow that references it.

03 Reference the Action

Actions are referenced using the uses: keyword. You must reference the action using the Organization/Repository/.github/actions/action-name@ref pattern.

If the action lives inside the same repository, you can just reference the file relative to the branch using ./.github/actions/action-name

.github > actions > output-test-action > ! action.yml

```yaml
name: Sample Output Composite Action
description: Sample Output Composite Action

outputs:
  sample_output:
    description: The Sample output
    value: ${{ steps.set_output.outputs.output_value }}

runs:
  using: composite
    - name: Set Output
      id: set_output
      shell: bash
      run: |
        echo "output_value=some_value" >> $GITHUB_OUTPUT
```

```yaml
name: Composite Action Example

on:
  workflow_dispatch

jobs:
  job_1:
    runs-on: ubuntu-latest
    steps:
      - name: Run Composite Action
        id: run_composite_action
        uses: ./.github/actions/output-test-action
      - run |
          echo ${{ steps.run_composite_action.outputs.sample_output }}
```

```
github > workflows > source-reusable-workflow.yml
1   name: Source Reusable Workflow
2
3   ###
4     workflow_call:
5       inputs:
6         environment_name:
7           description: The Passed in Environment
8           type: string
9       outputs:
10        job_output:
11          description: The Output of the Reusable Workflow
12          value: ${{ jobs.job_1.outputs.rw_env }}
13
14  jobs:
15    job_1:
16      runs-on: ubuntu-latest
17      outputs:
18        rw_env: ${{ steps.set_output.outputs.rw_env }}
19      steps:
20        - name: "Set Output"
21          id: set_output
22          run: |
23            echo "rw_env=rw-${{inputs.environment_name}}" >> $GITHUB_OUTPUT
24
```

DON'T REPEAT YOURSELF

Reusable Workflows

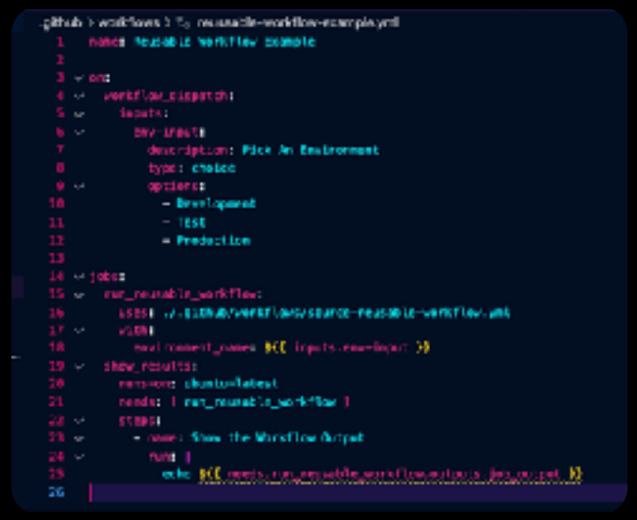
A Reusable Workflow is a collection of workflow job steps that can be reused in any workflow that references it.

01 Create the File

Reusable Workflows are stored in the .github/workflows folder.

02 Fill out Workflow

A reusable workflow can be any number of jobs and steps. All reusable workflows needs to have a workflow_call section defined in the on block. You can setup your inputs and outputs here.



DON'T REPEAT YOURSELF

Reusable Workflows

A Reusable Workflow is a collection of workflow job steps that can be reused in any workflow that references it.

03

Reference the Workflow

Reusable Workflows are referenced using the uses: keyword. You must reference the workflow using the Organization/Repository/.github/workflows/workflow_name.yml@branch pattern.

PUTTING IT ALL TOGETHER

For Your Consideration

Be careful with checkouts

action/checkout will overwrite the contents of the working directory by default if you are checking out another repository. Use clean: 'false' to bypass this functionality

Mind your scope

Composite actions and reusable workflows operate in the scope of the calling workflow. Make sure any secrets and variables are specified in the calling repository or at the organization level.

Know the right use case

In many scenarios, you can use composite actions or reusable workflows interchangeably. However there are some features that are exclusively available to each other.

Composite Actions allow nesting (being able to call another composite action from a composite action)
Secrets cant be passed to Composite actions
Conditionals are only allowed inside reuseable workflows
Composite Actions will log as a single step, where all steps in a reuseable workflow will display as a separate step

PUTTING IT ALL TOGETHER

For Your Consideration

Composite Actions Allow Nesting

A composite action will allow you to call another composite action from within itself. A reusable workflow only goes one level deep.

Reuseable Workflows Can Use Conditionals

You can't specify if blocks on your steps in a composite action.

Composite Actions Can't Access Secrets

Only reuseable workflows allow you to directly access secrets. To use a secret or a variable in a composite action, you must pass it in as an input.

Composite Actions allow nesting (being able to call another composite action from a composite action)
Secrets cant be passed to Composite actions
Conditionals are only allowed inside reuseable workflows
Composite Actions will log as a single step, where all steps in a reuseable workflow will display as a separate step

A StirTrek 2025 Presentation

Lights, Camera, Github Actions!

Thank You

Please Give Me Feedback

Scan the QR code here to fill out a Google Form to give me feedback on this presentation. It's like you're a focus group of one!

You're Still Here?

It's Over! Go Home!

x