GitHub Actions: Complete Learning Guide

Table of Contents

1. Introduction
2. Core Concepts
3. Workflow Anatomy
4. Secrets Management
5. Common Use Cases
6. Step-by-Step Setup
7. Debugging & Troubleshooting
8. Best Practices
9. Real-World Examples

Introduction

What is GitHub Actions?

GitHub Actions is an **automation platform** built into GitHub that allows you to automatically run code when specific events occur in your repository.

Think of it as:

- A robot that watches your repository
- When something happens (like a push), the robot follows instructions
- The robot can build, test, deploy, or perform any automated task

Key Benefits:

- ☑ Free (up to certain limits)
- ☑ Built into GitHub (no external setup needed)
- ☑ Runs on GitHub's servers
- ☑ Can automate repetitive tasks
- ☑ Supports custom scripts and third-party actions

Real-World Analogy

Without GitHub Actions:

```
Developer: Write code → Push to GitHub → Manually run build command
         → Manually run tests → Manually upload to server
         (Time: 30+ minutes, Error-prone)
```

With GitHub Actions:

```
Developer: Write code → Push to GitHub → Automatic workflow runs
         → Build, test, deploy complete → Website updated
         (Time: 5-10 minutes, Hands-free)
```

Core Concepts

1. **Workflows**

A workflow is a file that defines when and what automation should run.

- **Location:** `.github/workflows/` directory in your repo
- **Format:** YAML (.yml or .yaml files)
- **Purpose:** Contains all automation instructions

Example workflow file name: `deploy.yml`, `test.yml`, `build.yml`

2. **Events (Triggers)**

Events are actions that cause a workflow to run.

Common events:

- `push` - When code is pushed to a branch
- `pull_request` - When a pull request is created/updated
- `schedule` - On a timer (like cron jobs)
- `workflow_dispatch` - Manual trigger from GitHub UI
- `release` - When a release is published
- `issue` - When an issue is opened/closed

```yaml
# Example: Run on push to main branch
on:
  push:
    branches:
      - main
```

3. **Jobs**

A job is a set of tasks that run on the same virtual machine.

- Jobs run **in parallel** by default
- Can have dependencies (run one after another)
- Each job runs on a separate machine

```yaml
jobs:
  build:
    # This is one job
```

```
    ...
  test:
    # This is another job
    ...
```

4. **Steps**

Steps are individual commands or actions within a job.

- Run **sequentially** (one after another)
- Stop if any step fails (unless configured otherwise)
- Can be shell commands or pre-built actions

```
steps:
  - name: Step 1
    run: npm install
  - name: Step 2
    run: npm test
```

5. **Actions**

Pre-built reusable units of code that perform specific tasks.

- Created by GitHub or community
- Saves time vs. writing custom scripts
- Examples: `actions/checkout`, `actions/setup-node`

Format: `owner/repo@version` or `owner/repo/path@version`

```
uses: actions/setup-node@v3
```

6. **Runners**

Virtual machines where workflows run.

Types:

- **GitHub-hosted:** Free servers managed by GitHub
  - `ubuntu-latest` (Linux)
  - `windows-latest` (Windows)
  - `macos-latest` (macOS)
- **Self-hosted:** Your own machines

```
runs-on: ubuntu-latest
```

Workflow Anatomy

Here's a complete workflow broken down line-by-line:

```yaml
# ============================================
# WORKFLOW METADATA
# ============================================

name: Build and Test Application
# Display name for this workflow in GitHub UI

env:
  NODE_VERSION: 18
  # Variables available to all jobs and steps

# ============================================
# TRIGGERS
# ============================================

on:
  push:
    branches:
      - main
      - develop
  pull_request:
    branches:
      - main
  schedule:
    # Run daily at 2 AM UTC
    - cron: '0 2 * * *'
  workflow_dispatch:
  # Allows manual trigger from GitHub UI

# ============================================
# JOBS
# ============================================

jobs:
  # JOB 1: Build
  build:
    name: Build Application
    runs-on: ubuntu-latest
    # Runs on GitHub-hosted Ubuntu machine

    # Job-level environment variables
    env:
      BUILD_ENV: production

    outputs:
      # Values that other jobs can use
      build-number: ${{ steps.build.outputs.number }}
```

```yaml
    steps:
      # STEP 1: Get code from repository
      - name: Checkout code
        uses: actions/checkout@v3
        # This action downloads the repo

      # STEP 2: Setup Node.js
      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: ${{ env.NODE_VERSION }}
          cache: npm
          # 'cache: npm' speeds up future runs

      # STEP 3: Install dependencies
      - name: Install dependencies
        run: npm install
        # 'run' executes shell commands

      # STEP 4: Build application
      - name: Build application
        id: build
        # 'id' allows other steps to reference this one
        run: |
          npm run build
          echo "number=$(date +%s)" >> $GITHUB_OUTPUT
          # $GITHUB_OUTPUT stores values for other steps

      # STEP 5: Upload build artifact
      - name: Upload build artifact
        uses: actions/upload-artifact@v3
        with:
          name: build-output
          path: dist/
          # Saves 'dist' folder for other jobs to download

  # JOB 2: Test (runs in parallel with build)
  test:
    name: Run Tests
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: 18

      - name: Install dependencies
        run: npm install

      - name: Run tests
```

```
        run: npm test

      - name: Upload coverage
        uses: actions/upload-artifact@v3
        with:
          name: coverage-report
          path: coverage/

  # JOB 3: Deploy (runs AFTER build and test)
  deploy:
    name: Deploy Application
    runs-on: ubuntu-latest
    needs: [build, test]
    # 'needs' means this job waits for build & test to complete
    if: github.ref == 'refs/heads/main' && success()
    # 'if' adds conditions: only deploy if on main AND previous jobs passed

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Download build artifact
        uses: actions/download-artifact@v3
        with:
          name: build-output

      - name: Deploy to server
        run: |
          mkdir -p ~/.ssh
          echo "${{ secrets.SSH_PRIVATE_KEY }}" > ~/.ssh/id_rsa
          chmod 600 ~/.ssh/id_rsa
          scp -r dist/* user@server:/var/www/html/
```

Breaking Down Key Lines:

Line	Meaning
`name:`	Display name for workflow/job/step
`on:`	What triggers the workflow
`runs-on:`	What machine to run on
`uses:`	Pre-built action to use
`run:`	Shell command to execute
`with:`	Parameters for an action
`env:`	Environment variables
`needs:`	Job dependencies
`if:`	Conditional execution

Line	Meaning
`id:`	Identifier for referencing in other steps
`outputs:`	Values available to other jobs

Secrets Management

Why Secrets Matter

Problem:

```
deploy:
  run: scp file user:password@server
  # Password is visible in code! ❌ SECURITY RISK
```

Solution: Use Secrets

```
deploy:
  run: scp file ${{ secrets.SSH_KEY }}@server
  # Password never appears in logs ☑ SECURE
```

How Secrets Work

1. **Encrypted Storage:** GitHub stores secrets encrypted
2. **No Logging:** Secrets never appear in build logs (replaced with `***`)
3. **Runtime Access:** Available as environment variables during workflow
4. **No Inheritance:** Not passed to sub-processes or exported

Creating Secrets

Step-by-Step:

1. **Go to Repository Settings**
   - GitHub → Your Repo → Settings (top right)
2. **Navigate to Secrets**
   - Left sidebar: "Secrets and variables" → "Actions"
3. **Create Secret**
   - Click "New repository secret"
   - Name: `MY_SECRET_NAME` (must be uppercase with underscores)
   - Value: Your actual secret value
   - Click "Add secret"

4. **Use in Workflow**

```
steps:
  - run: echo ${{ secrets.MY_SECRET_NAME }}
```

Secret Naming Best Practices

```
# Good names:
FTP_PASSWORD
DATABASE_URL
API_KEY
SSH_PRIVATE_KEY
DEPLOYMENT_TOKEN

# Avoid:
password
secret
key123
```

Types of Secrets

1. Credentials

- API keys
- Passwords
- Tokens
- SSH keys

2. URLs

- Server addresses
- Database URLs

3. Configuration

- API endpoints
- Region settings

Security Best Practices

- 🔒 Never copy/paste secrets into code
- 🔒 Use secrets for all sensitive data
- 🔒 Rotate secrets periodically
- 🔒 Use service accounts (not personal credentials)
- 🔒 Limit permissions to minimum needed
- 🔒 Audit secret usage in workflows

Common Use Cases

1. **Continuous Integration (Testing)**

Run tests automatically on every push.

```yaml
name: Tests

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
      - run: npm install
      - run: npm test
```

Benefits:

- Catch bugs before merging
- Ensure code quality
- Prevent broken code in main branch

2. **Continuous Deployment**

Automatically deploy when code is pushed.

```yaml
name: Deploy

on:
  push:
    branches: [main]

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - run: npm install && npm run build
      - run: |
          # Deploy via FTP, SSH, cloud API, etc.
          rsync -r dist/ user@server:/var/www/
```

Benefits:

- Instant updates
- Less manual work

- Fewer deployment errors

3. **Build on Schedule**

Run tasks periodically (daily, hourly, weekly).

```yaml
on:
  schedule:
    # Run every day at 9 AM UTC
    - cron: '0 9 * * *'

jobs:
  scheduled-task:
    runs-on: ubuntu-latest
    steps:
      - run: echo "Daily task running"
```

Use cases:

- Database backups
- Generate reports
- Cleanup old files
- Health checks

4. **Cross-Platform Testing**

Test on multiple operating systems.

```yaml
jobs:
  test:
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest, macos-latest]
        node-version: [16, 18, 20]
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          node-version: ${{ matrix.node-version }}
      - run: npm test
```

Benefits:

- Ensure compatibility
- Catch OS-specific bugs

5. **Publish to Package Registry**

Automatically publish packages when version changes.

```yaml
on:
  push:
    tags:
      - 'v*'

jobs:
  publish:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
      - run: npm ci
      - run: npm run build
      - run: npm publish
        env:
          NODE_AUTH_TOKEN: ${{ secrets.NPM_TOKEN }}
```

Step-by-Step Setup

Example: Deploy Frontend to FTP

Step 1: Create Workflow File

In your repository root, create: `.github/workflows/deploy.yml`

```yaml
name: Build and Deploy Frontend

on:
  push:
    branches:
      - main
  workflow_dispatch:

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '18'
          cache: 'npm'
```

```
      - name: Install dependencies
        run: npm install

      - name: Build
        run: npm run build
        env:
          API_URL: ${{ secrets.API_URL }}

      - name: Deploy via FTP
        uses: SamKirkland/FTP-Deploy-Action@v4.3.5
        with:
          server: ${{ secrets.FTP_HOST }}
          username: ${{ secrets.FTP_USERNAME }}
          password: ${{ secrets.FTP_PASSWORD }}
          local-dir: ./dist/
          server-dir: /public_html/
          dangerous-delete-files: true
```

Step 2: Gather FTP Information

From your hosting provider:

- FTP Host (IP or hostname)
- FTP Username
- FTP Password
- FTP Port (usually 21)
- Remote directory path

Step 3: Add GitHub Secrets

1. Go to: GitHub → Your Repo → Settings → Secrets and variables → Actions
2. Click "New repository secret" and add:

Secret Name	Value
`FTP_HOST`	Your FTP hostname/IP
`FTP_USERNAME`	Your FTP username
`FTP_PASSWORD`	Your FTP password
`API_URL`	Your API endpoint URL

Step 4: Commit and Push

```
git add .github/workflows/deploy.yml
git commit -m "add: GitHub Actions deployment workflow"
git push
```

Step 5: Monitor Workflow

1. Go to GitHub repo → "Actions" tab
2. See your workflow running (yellow) → completed (green/red)
3. Click on workflow to see detailed logs

Debugging & Troubleshooting

Understanding Status Indicators

Symbol	Meaning
🟡 Yellow circle	Workflow running
✅ Green checkmark	Workflow succeeded
❌ Red X	Workflow failed
⚪ Gray circle	Skipped

Common Issues and Solutions

Issue 1: Workflow Not Triggering

Problem: You pushed code but workflow didn't run

Solutions:

- Check the `on:` trigger matches your action
- Workflows only trigger on pushed commits, not local changes
- Workflow file must be in `.github/workflows/` directory
- Workflow file must have `.yml` or `.yaml` extension

Verify:

```
on:
  push:
    branches:
      - main  # Make sure this is your branch name
```

Issue 2: Secrets Not Working

Problem: Getting "secret is not defined" error

Solutions:

- Check secret name spelling (case-sensitive)
- Use `${{ secrets.SECRET_NAME }}` syntax
- Restart workflow after adding secrets

- Secrets not available in pull requests (for security)

Verify:

```
- run: echo "Checking secret..."
- run: echo ${{ secrets.MY_SECRET }}
# Output shows: *** (secrets are masked)
```

Issue 3: Deployment Permission Denied

Problem: FTP upload fails with "Permission denied"

Solutions:

- Verify FTP credentials are correct
- Check if remote directory path is correct
- Ensure FTP user has write permissions
- Test credentials manually with FTP client

Debug:

```
- name: Test FTP connection
  run: |
    echo "Testing FTP connectivity..."
    echo "Host: ${{ secrets.FTP_HOST }}"
    echo "Username: ${{ secrets.FTP_USERNAME }}"
```

Issue 4: Build Command Fails

Problem: `npm run build` fails in workflow

Solutions:

- Works locally but fails in CI? Something is different
- Check Node.js version
- Check if dependencies are installed
- Check for hardcoded paths (use relative paths)

Debug:

```
- name: Debug environment
  run: |
    node --version
    npm --version
    npm list
    # Shows what's installed
```

Viewing Logs

How to see what went wrong:

1. Click on the failed workflow in Actions tab
2. Click on the failed job
3. Click on the failed step
4. Read the error message
5. Adjust workflow based on error

Common error messages:

```
Error: File not found
→ Solution: Check file path, verify build step completed

Error: Invalid credentials
→ Solution: Verify secrets are correct

Error: Permission denied
→ Solution: Check file permissions and user access

Error: Command not found
→ Solution: Install required tool before using it
```

Enabling Debug Logging

For extra verbose output, add secrets to your repo:

```
ACTIONS_STEP_DEBUG: true
ACTIONS_RUNNER_DEBUG: true
```

Then view more detailed logs in Actions tab.

Best Practices

1. **Use Specific Versions**

```yaml
# ❌ Avoid: Uses latest (could break anytime)
uses: actions/checkout@main

# ☑ Good: Uses specific version
uses: actions/checkout@v3.0.0

# ☑ Also good: Uses latest patch in series
uses: actions/checkout@v3
```

2. **Cache Dependencies**

Speeds up workflow runs significantly.

```
- uses: actions/setup-node@v3
  with:
    node-version: '18'
    cache: 'npm'
    # Cache npm packages between runs
```

3. **Use Meaningful Names**

Makes logs easier to understand.

```
# ❌ Avoid
- run: npm run build

# ☑ Good
- name: Build production assets
  run: npm run build
```

4. **Separate Concerns**

Different jobs for different purposes.

```
jobs:
  lint:
    # Check code style
  test:
    # Run tests
  build:
    # Build application
  deploy:
    # Deploy to server
```

5. **Add Conditions Carefully**

```
# Only deploy on main branch AND if tests pass
if: github.ref == 'refs/heads/main' && success()

# Available contexts:
# github.ref - Current branch
# github.actor - Who triggered it
# success() - If previous job succeeded
# failure() - If previous job failed
```

6. **Use Environment Variables**

```
# ❌ Avoid: Hardcoding values
run: npm run build --target production

# ☑ Good: Use variables
env:
  BUILD_TARGET: production
run: npm run build --target ${{ env.BUILD_TARGET }}
```

7. **Limit Permissions**

Always use least-privilege principle.

```
jobs:
  my-job:
    permissions:
      contents: read      # Only read repo content
      pull-requests: read # Only read PRs
      # Don't give more permissions than needed
```

Real-World Examples

Example 1: React App Auto-Deploy to Hosting

```
name: Deploy React App

on:
  push:
    branches: [main]

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          node-version: 18
          cache: npm

      - run: npm install
      - run: npm run build
        env:
          VITE_API_URL: ${{ secrets.API_URL }}
```

```
      - name: Deploy to Netlify
        uses: netlify/actions/cli@master
        with:
          args: deploy --prod
        env:
          NETLIFY_SITE_ID: ${{ secrets.NETLIFY_SITE_ID }}
          NETLIFY_AUTH_TOKEN: ${{ secrets.NETLIFY_AUTH_TOKEN }}
```

Example 2: Python App Tests

```
name: Python Tests

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: ['3.8', '3.9', '3.10', '3.11']

    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-python@v4
        with:
          python-version: ${{ matrix.python-version }}
          cache: 'pip'

      - run: pip install -r requirements.txt
      - run: pytest
      - run: black --check .
```

Example 3: Database Backup on Schedule

```
name: Backup Database

on:
  schedule:
    - cron: '0 2 * * *'  # Every day at 2 AM UTC

jobs:
  backup:
    runs-on: ubuntu-latest
    steps:
      - name: Backup database
        run: |
          mysqldump -u${{ secrets.DB_USER }} \
            -p${{ secrets.DB_PASSWORD }} \
```

```
            ${{ secrets.DB_NAME }} > backup.sql

      - name: Upload to storage
        uses: actions/upload-artifact@v3
        with:
          name: database-backup-${{ github.run_id }}
          path: backup.sql
          retention-days: 7
```

Example 4: Publish Package on Release

```
name: Publish Package

on:
  release:
    types: [created]

jobs:
  publish:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3

      - run: npm install
      - run: npm test

      - name: Publish to npm
        run: npm publish
        env:
          NODE_AUTH_TOKEN: ${{ secrets.NPM_TOKEN }}
```

Advanced Concepts

Matrix Strategy

Run the same job with different configurations.

```
strategy:
  matrix:
    os: [ubuntu-latest, windows-latest]
    node-version: [16, 18, 20]

# Creates 6 jobs: 3 Node versions × 2 OSes
```

Artifacts

Save and share files between jobs.

```yaml
# Job 1: Create artifact
- uses: actions/upload-artifact@v3
  with:
    name: my-artifact
    path: dist/

# Job 2: Download artifact
- uses: actions/download-artifact@v3
  with:
    name: my-artifact
```

Environment Variables

Different scope levels:

```yaml
# Global (all jobs/steps)
env:
  GLOBAL_VAR: value

jobs:
  job1:
    # Job-level (this job only)
    env:
      JOB_VAR: value
    steps:
      # Step-level (this step only)
      - run: echo $JOB_VAR
        env:
          STEP_VAR: value
```

Conditional Execution

```yaml
# Run only if branch is main
if: github.ref == 'refs/heads/main'

# Run only if previous job succeeded
if: success()

# Run only if files changed
if: contains(github.event.head_commit.modified, 'src/')

# Run only for pull requests
if: github.event_name == 'pull_request'
```

Summary

Concept	Purpose
Workflow	File defining automation rules
Event	Trigger for workflow (push, PR, schedule)
Job	Set of steps running on same machine
Step	Individual command or action
Action	Reusable automation unit
Runner	Virtual machine running the workflow
Secret	Encrypted credential available to workflow
Artifact	Files saved and shared between jobs

Resources

- **Official Docs:** https://docs.github.com/en/actions
- **Marketplace:** https://github.com/marketplace?type=actions
- **YAML Syntax:** https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions
- **Context & Variables:** https://docs.github.com/en/actions/learn-github-actions/contexts
- **Cron Syntax:** https://crontab.guru/

Quick Reference

Workflow File Template

```yaml
name: My Workflow

on:
  push:
    branches: [main]
  pull_request:
  workflow_dispatch:

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          node-version: 18
      - run: npm install
```

```
      - run: npm test
      - run: npm run build
```

Common Secrets Pattern

```
# 1. Create secrets in GitHub Settings
# 2. Reference in workflow
- run: ./deploy.sh
  env:
    USERNAME: ${{ secrets.USERNAME }}
    PASSWORD: ${{ secrets.PASSWORD }}
    API_KEY: ${{ secrets.API_KEY }}
```

Debug Tips

```
- name: Show environment info
  run: |
    echo "Node: $(node --version)"
    echo "NPM: $(npm --version)"
    echo "Branch: ${{ github.ref }}"
    echo "Actor: ${{ github.actor }}"

- name: List files
  run: ls -la

- name: Show current directory
  run: pwd
```

Last Updated: February 2026
Version: 1.0
Status: Complete Learning Guide