GitHub Actions Quick Start: Using Marketplace & Custom Actions

Detailed, Runnable Examples

Updated: November 3, 2025

Contents

How to compile this LaTeX file with code highlighting:
Install Python + Pygments, then run:

```
latexmk -pdf -shell-escape Actions_Custom_Actions_Quick_Start.tex
```

If your environment forbids `-shell-escape`, replace `minted` with `verbatim` or use Overleaf with shell-escape enabled.

1 What You'll Learn

This guide gives you a working, step-by-step path to:

- Use Marketplace actions and Docker images in a workflow.
- Pass data across jobs with **artifacts**.
- Speed up builds with **cache**.
- Author three kinds of custom actions: **Composite**, **JavaScript**, and **Docker**.
- Apply version pinning and output handling best practices.

Key conventions used here

- Workflows live under `.github/workflows/*.yml`.
- Local actions are referenced via `uses: ./`. Pin third-party actions by tag or SHA.
- For action outputs, write `key=value` lines to `$GITHUB_OUTPUT`.
- Avoid Unicode box-drawing characters in docs; use plain ASCII trees.

2 Discover and Use Actions (Marketplace)

Create `.github/workflows/actions.yml`:

```yaml
# .github/workflows/actions.yml
name: actions-basics
on: [push]

jobs:
  demo:
    runs-on: ubuntu-latest
    permissions:
      contents: read
    steps:
      - name: Show files before checkout
        run: ls -la

      - name: Checkout repo
        uses: actions/checkout@v4

      - name: Show files after checkout
        run: ls -la

      - name: Hello World (JS action example)
        id: hello
        uses: actions/hello-world-javascript-action@main
        with:
          who-to-greet: "Jordan"

      - name: Print greeting time from previous step
        run: echo "greeting time: ${{ steps.hello.outputs.time }}"

      - name: Run a Docker action (hello-world image)
        uses: docker://hello-world:latest
```

Notes

- Prefer tags or SHAs for third-party actions to control supply-chain risk.
- `docker://` lets you run any public image as a step.

3 Share Data Between Jobs (Artifacts)

Create `.github/workflows/artifacts.yml`:

```yaml
# .github/workflows/artifacts.yml
name: artifacts
on: [push]

jobs:
  upload:
    name: Upload an artifact
    runs-on: ubuntu-latest
    steps:
      - name: Generate file
        run: docker info > myArtifact.txt
      - name: Upload artifact
        uses: actions/upload-artifact@v4
        with:
          name: my-artifact
          path: myArtifact.txt

  download:
    name: Download & use artifact
    runs-on: ubuntu-latest
    needs: upload
    steps:
      - name: Download artifact
        uses: actions/download-artifact@v4
        with:
          name: my-artifact
      - name: List
        run: ls -la
      - name: Show contents
        run: cat myArtifact.txt
```

Tips

- Artifacts are for sharing build outputs/logs across jobs and runs. Default retention is often 90 days (can vary).
- Artifacts are not caches. Use §4 for dependency acceleration.

4 Speed Up Runs (Cache)

Create `.github/workflows/cache.yml`:

```yaml
# .github/workflows/cache.yml
name: cache
on: workflow_dispatch

jobs:
  test-cache:
    name: Cache pip dependencies
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: "3.12"

      - name: Cache pip
        id: pip-cache
        uses: actions/cache@v4
        with:
          path: ~/.cache/pip
          key: ${{ runner.os }}-pip-${{ hashFiles('**/requirements.txt') }}
          restore-keys: |
            ${{ runner.os }}-pip-

      - name: Install deps (skips downloads if cache hit)
        run: pip install -r requirements.txt

      - name: Confirm cache hit
        if: steps.pip-cache.outputs.cache-hit == 'true'
        run: echo "Cache entry found"
```

Notes

- Use language setup actions (e.g., `actions/setup-node`) that add first-class caching when available.
- Keep cache keys stable and specific; size limits apply per key.

5 Build a Composite Action

Repo Layout

```
/action-composite/
  action.yml
  .github/workflows/composite.yml
```

Composite action `action.yml`

```yaml
name: "composite-calc"
description: "Adds two numbers (composite action)"
inputs:
  number1:
    description: "First integer"
    required: true
    default: "0"
  number2:
    description: "Second integer"
    required: true
    default: "0"
outputs:
  result:
    description: "Sum of number1 and number2"
    value: ${{ steps.sum.outputs.result }}

runs:
  using: "composite"
  steps:
    - id: sum
      shell: bash
      run: |
        n1="${{ inputs.number1 }}"
        n2="${{ inputs.number2 }}"
        echo "result=$(( n1 + n2 ))" >> "$GITHUB_OUTPUT"
```

Workflow to use it

```yaml
# .github/workflows/composite.yml
name: composite-demo
on: [push]

jobs:
  run-composite:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - id: add
        uses: ./
        with:
          number1: "4"
          number2: "2"
      - run: echo "Result = ${{ steps.add.outputs.result }}"
```

6 Build a JavaScript Action

Repo Layout

```
/action-js/
  action.yml
  index.js
  package.json
  .github/workflows/js.yml
```

`action.yml`

```yaml
name: "js-calc"
description: "Adds two numbers (JavaScript action)"
inputs:
  number1: { description: "First integer", required: true, default: "0" }
  number2: { description: "Second integer", required: true, default: "0" }
outputs:
  result: { description: "Sum of inputs" }
runs:
  using: "node20"
  main: "dist/index.js"
```

`index.js` (source)

```javascript
const core = require('@actions/core');

async function run() {
  try {
    const n1 = parseInt(core.getInput('number1'), 10);
    const n2 = parseInt(core.getInput('number2'), 10);
    const result = (n1 || 0) + (n2 || 0);
    core.setOutput('result', String(result));
  } catch (err) {
    core.setFailed(err.message || String(err));
  }
}

run();
```

`package.json`

```json
{
  "name": "js-calc-action",
  "version": "1.0.0",
  "private": true,
  "main": "dist/index.js",
  "scripts": {
    "build": "npx @vercel/ncc build index.js -o dist"
  },
  "dependencies": {
    "@actions/core": "^1.11.1"
  },
  "devDependencies": {
    "@vercel/ncc": "^0.38.3"
  }
}
```

Workflow: build then use (without committing `dist/`)

```yaml
# .github/workflows/js.yml
name: js-action-demo
on: [push]

jobs:
  run-js-action:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: "20"
      - run: npm ci
      - run: npm run build
      - id: add
        uses: ./
        with:
          number1: "4"
          number2: "2"
      - run: echo "Result = ${{ steps.add.outputs.result }}"
```

Tip Many JS actions commit compiled `dist/` to their repos to avoid building at runtime. The example shows a “build then use” pattern.

7 Build a Docker Container Action

Repo Layout

```
/action-docker/
  action.yml
  Dockerfile
  entrypoint.sh
  .github/workflows/docker.yml
```

`entrypoint.sh`

```sh
#!/usr/bin/env sh
set -euo pipefail

n1="${1:-0}"
n2="${2:-0}"
result=$(( n1 + n2 ))

echo "result=$result" >> "$GITHUB_OUTPUT"

# Also write a file into the workspace to demonstrate sharing
echo "container output file" > "$GITHUB_WORKSPACE/container_output.txt"
```

`Dockerfile` (capital "D" is required)

```dockerfile
FROM alpine:3.20
COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh
ENTRYPOINT ["/entrypoint.sh"]
```

`action.yml`

```yaml
name: "docker-calc"
description: "Adds two numbers (Docker action)"
inputs:
  number1: { description: "First integer", required: true, default: "0" }
  number2: { description: "Second integer", required: true, default: "0" }
outputs:
  result: { description: "Sum of inputs" }
runs:
  using: "docker"
  image: "Dockerfile"
  args:
    - ${{ inputs.number1 }}
    - ${{ inputs.number2 }}
```

Workflow to use it

```yaml
# .github/workflows/docker.yml
name: docker-action-demo
on: [push]

jobs:
  run-docker-action:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - id: add
        uses: ./
        with:
          number1: "4"
          number2: "2"
      - run: echo "Result = ${{ steps.add.outputs.result }}"
      - name: Show file dropped by container
        run: cat "$GITHUB_WORKSPACE/container_output.txt"
```

8 Best Practices & Troubleshooting

Version Pinning

Pin third-party actions to a release tag or commit SHA. Maintain a refresh policy (e.g. scheduled PRs to bump tags).

Outputs and Files

Use `$GITHUB_OUTPUT` for step outputs. Use artifacts to share files across jobs; use cache to speed dep installs.

Common `minted` Pitfalls

- **Missing Pygments**: install with `pip install Pygments`.
- **Frozencache/sty errors**: avoid `frozencache` unless you manage pygstyles; compile with `-shell-escape`.
- **Unicode line-drawing**: replace with ASCII.

Security Considerations

- Use fine-grained `permissions:` in jobs; default to least privilege.
- Avoid untrusted actions; fork and pin, or vendor actions when necessary.
- Sanitize inputs; treat outputs and artifacts as untrusted data.

9 Checklist

1. Create the sample workflows (§1–§3) and run them.
2. Add the composite, JS, and Docker action folders with files from this guide.
3. Trigger the demo workflows and verify outputs.
4. Pin versions, set permissions, and add CODEOWNERS/branch protection as needed.