# Storing workflow data as artifacts

**In this article**



Artifacts allow you to share data between jobs in a workflow and store data once that workflow has completed.

## About workflow artifacts

Artifacts allow you to persist data after a job has completed, and share that data with another job in the same workflow. An artifact is a file or collection of files produced during a workflow run. For example, you can use artifacts to save your build and test output after a workflow run has ended. All actions and workflows called within a run have write access to that run's artifacts.

By default, GitHub stores build logs and artifacts for 90 days, and this retention period can be customized. For more information, see "Usage limits, billing, and administration." The retention period for a pull request restarts each time someone pushes a new commit to the pull request.

These are some of the common artifacts that you can upload:

- Log files and core dumps
- Test results, failures, and screenshots
- Binary or compressed files
- Stress test performance output and code coverage results

Storing artifacts uses storage space on GitHub. GitHub Actions usage is free for standard GitHub-hosted runners in public repositories, and for self-hosted runners. For private repositories, each GitHub account receives a certain amount of free minutes and storage for use with GitHub-hosted runners, depending on the account's plan. Any usage beyond the included amounts is controlled by spending limits. For more information, see "Managing billing for GitHub Actions."

Artifacts are uploaded during a workflow run, and you can view an artifact's name and size in the UI. When an artifact is downloaded using the GitHub UI, all files that were individually uploaded as part of the artifact get zipped together into a single file. This means that billing is calculated based on the size of the uploaded artifact and not the size of the zip file.

GitHub provides two actions that you can use to upload and download build artifacts. For more information, see the upload-artifact and download-artifact actions.

To share data between jobs:

- **Uploading files**: Give the uploaded file a name and upload the data before the job ends.
- **Downloading files**: You can only download artifacts that were uploaded during the same workflow run. When you download a file, you can reference it by name.

The steps of a job share the same environment on the runner machine, but run in their own individual processes. To pass data between steps in a job, you can use inputs and outputs. For more information about inputs and outputs, see "Metadata syntax for GitHub Actions."

## Comparing artifacts and dependency caching

Artifacts and caching are similar because they provide the ability to store files on GitHub, but each feature offers different use cases and cannot be used interchangeably.

- Use caching when you want to reuse files that don't change often between jobs or workflow runs, such as build dependencies from a package management system.
- Use artifacts when you want to save files produced by a job to view after a workflow run has ended, such as built binaries or build logs.

For more information on dependency caching, see "Caching dependencies to speed up workflows."

## Uploading build and test artifacts

You can create a continuous integration (CI) workflow to build and test your code. For more information about using GitHub Actions to perform CI, see "About continuous integration."

The output of building and testing your code often produces files you can use to debug test failures and production code that you can deploy. You can configure a workflow to build and test the code pushed to your repository and report a success or failure status. You can upload the build and test output to use for deployments, debugging failed tests or crashes, and viewing test suite coverage.

You can use the `upload-artifact` action to upload artifacts. When uploading an artifact, you can specify a single file or directory, or multiple files or directories. You can also exclude certain files or directories, and use wildcard patterns. We recommend that you provide a name for an artifact, but if no name is provided then `artifact` will be used as the default name. For more information on syntax, see the actions/upload-artifact action.

### Example

For example, your repository or a web application might contain SASS and TypeScript files that you must convert to CSS and JavaScript. Assuming your build configuration outputs the compiled files in the `dist` directory, you would deploy the files in the `dist` directory to your web application server if all tests completed successfully.

```
|-- hello-world (repository)
|   └── dist
|   └── tests
|   └── src
|       └── sass/app.scss
|       └── app.ts
|   └── output
|       └── test
|
```

This example shows you how to create a workflow for a Node.js project that builds the code in the `src` directory and runs the tests in the `tests` directory. You can assume that running `npm test` produces a code coverage report named `code-coverage.html` stored in the `output/test/` directory.

The workflow uploads the production artifacts in the `dist` directory, but excludes any markdown files. It also uploads the `code-coverage.html` report as another artifact.

YAML

```yaml
name: Node CI

on: [push]

jobs:
  build_and_test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4
      - name: npm install, build, and test
        run: |
          npm install
          npm run build --if-present
          npm test
      - name: Archive production artifacts
        uses: actions/upload-artifact@v3
        with:
          name: dist-without-markdown
          path: |
            dist
            !dist/**/*.md
      - name: Archive code coverage results
        uses: actions/upload-artifact@v3
        with:
          name: code-coverage-report
          path: output/test/code-coverage.html
```

## Configuring a custom artifact retention period

You can define a custom retention period for individual artifacts created by a workflow. When using a workflow to create a new artifact, you can use `retention-days` with the `upload-artifact` action. This example demonstrates how to set a custom retention period of 5 days for the artifact named `my-artifact`:

YAML

```yaml
  - name: 'Upload Artifact'
    uses: actions/upload-artifact@v3
    with:
      name: my-artifact
      path: my_file.txt
      retention-days: 5
```

The `retention-days` value cannot exceed the retention limit set by the repository, organization, or enterprise.

## Downloading or deleting artifacts

During a workflow run, you can use the `download-artifact` action to download artifacts that were previously uploaded in the same workflow run.

After a workflow run has been completed, you can download or delete artifacts on GitHub or using the REST API. For more information, see "Downloading workflow artifacts," "Removing workflow artifacts," and the "Actions."

### Downloading artifacts during a workflow run

The `actions/download-artifact` action can be used to download previously uploaded artifacts during a workflow run.

> **Note:** You can only download artifacts in a workflow that were uploaded during the same workflow run.

Specify an artifact's name to download an individual artifact. If you uploaded an artifact without specifying a name, the default name is `artifact`.

```
- name: Download a single artifact
  uses: actions/download-artifact@v3
  with:
    name: my-artifact
```

You can also download all artifacts in a workflow run by not specifying a name. This can be useful if you are working with lots of artifacts.

```
- name: Download all workflow run artifacts
  uses: actions/download-artifact@v3
```

If you download all workflow run's artifacts, a directory for each artifact is created using its name.

For more information on syntax, see the actions/download-artifact action.

## Passing data between jobs in a workflow

You can use the `upload-artifact` and `download-artifact` actions to share data between jobs in a workflow. This example workflow illustrates how to pass data between jobs in the same workflow. For more information, see the actions/upload-artifact and download-artifact actions.

Jobs that are dependent on a previous job's artifacts must wait for the dependent job to complete successfully. This workflow uses the `needs` keyword to ensure that `job_1`, `job_2`, and `job_3` run sequentially. For example, `job_2` requires `job_1` using the `needs: job_1` syntax.

Job 1 performs these steps:

- Performs a math calculation and saves the result to a text file called `math-homework.txt`.
- Uses the `upload-artifact` action to upload the `math-homework.txt` file with the artifact name `homework`.

Job 2 uses the result in the previous job:

- Downloads the `homework` artifact uploaded in the previous job. By default, the `download-artifact` action downloads artifacts to the workspace directory that the step is executing in. You can use the `path` input parameter to specify a different download directory.
- Reads the value in the `math-homework.txt` file, performs a math calculation, and saves the result to `math-homework.txt` again, overwriting its contents.

- Uploads the `math-homework.txt` file. This upload overwrites the previously uploaded artifact because they share the same name.

Job 3 displays the result uploaded in the previous job:

- Downloads the `homework` artifact.
- Prints the result of the math equation to the log.

The full math operation performed in this workflow example is `(3 + 7) x 9 = 90`.

YAML

```yaml
name: Share data between jobs

on: [push]

jobs:
  job_1:
    name: Add 3 and 7
    runs-on: ubuntu-latest
    steps:
      - shell: bash
        run: |
          expr 3 + 7 > math-homework.txt
      - name: Upload math result for job 1
        uses: actions/upload-artifact@v3
        with:
          name: homework
          path: math-homework.txt

  job_2:
    name: Multiply by 9
    needs: job_1
    runs-on: windows-latest
    steps:
      - name: Download math result for job 1
        uses: actions/download-artifact@v3
        with:
          name: homework
      - shell: bash
        run: |
          value=`cat math-homework.txt`
          expr $value \* 9 > math-homework.txt
      - name: Upload math result for job 2
        uses: actions/upload-artifact@v3
        with:
          name: homework
          path: math-homework.txt

  job_3:
    name: Display results
    needs: job_2
    runs-on: macOS-latest
    steps:
      - name: Download math result for job 2
        uses: actions/download-artifact@v3
        with:
          name: homework
      - name: Print the final result
        shell: bash
        run: |
          value=`cat math-homework.txt`
          echo The result is $value
```

The workflow run will archive any artifacts that it generated. For more information on downloading archived artifacts, see "Downloading workflow artifacts."

## Further reading

- "Managing billing for GitHub Actions".