GitHub Actions provides us with the advantage of reusing actions to automate processes rather than having to write each and every step from scratch.

You can use a reusable action in another workflow in any of the following scenarios:

- As private reusable actions where both the workflows exist in the same repository.
- As public reusable actions where the actions are available in GitHub Actions Marketplace.

How to reference a public action?

There are ready-made actions available in the GitHub Actions Marketplace. They are contributed by the community to make building workflows easier. To access these actions from the marketplace, use the following keywords:

- `uses`
- `: This value takes the format of`
- `action-name/version`
- `or`
- `github-repo-owner/repo-name`
- `.`
- `with`
- `: The value is an input if the action requires one.`

There are several ways to reference a public action from the marketplace.

- Referencing a branch
- Referencing a version
- Referencing a commit

This is the basic format for referencing a public action:

```
steps:
  - name: step-name
    uses: #reference

  with: #inputs
```

You have to provide a reference with the

`uses` keyword, and you can provide relevant inputs under the `with` keyword.

- Referencing a branch:
- `uses: publisher-username/repo-name@branch-name`
- Referencing a version: You can specify a version for the action by checking the available versions in the repository.
- `uses: publisher-username/repo-name@v2`
- Referencing a commit ID: You can specify a specific commit ID for the action by checking the commit history in the repository.
- `uses: publisher-username/repo-name@commit-ID`

Now, let's explore popular actions provided by the GitHub team.

Checkout

Checkout enables our workflow to access the GitHub repository by checking out our repository to the runner.

Checkout has a `fetch-depth` option that allows you to clone not only the tip of the branch but its full history - handy to do some deeper Git tree analysis.

```
- uses: actions/checkout@v3
  with:
    fetch-depth: 0
```

fetch-depth indicated the number of commits to fetch. 0 means to fetch all commits history, branches, and tags. By default, `fetch-depth`

 is set to 1, which fetches branch tips and no prior history. If fetch-depth is set to 10, it means to fetch the last 10 commits.

The

ref

 keyword can mean one of the following:

- The branch name
- Tag
- SHA for the commit

When you make a commit to save your work, Git creates a unique ID SHA (Secure Hash Algorithm) that allows you to keep a record of the specific changes committed, along with information about who made them and when.

```
- uses: actions/checkout@v3
  with:

  ref: The branch name, tag or SHA to checkout.
```

Artifacts

Artifacts are used for sharing data between jobs. Moreover, it also saves data once the workflow is finished running. Artifacts are generally used to save logs and reports of your workflow. In GitHub Actions, you can upload artifacts and also download them.

To upload an artifact, use the Upload a Build Artifact GitHub Action created by GitHub.

```
- name: Upload deployable package
  uses: actions/upload-artifact@v2
  with:
    name: production-files

  path: path/to/artifact
```

Here, you only need to provide 2 parameters:

- name
- : The name you want to give to identify the artifact.
- path

- : The path to the file or to the directory.

You can see the generated artifact in the GitHub Actions UI once your build is complete.

It is not recommended to upload a high number of artifacts in a short period of time because the requests will get blocked due to high amount of traffic. To circumvent this, you can archive your artifact first, and then proceed to upload it.

Similarly, you can use the *Download artifact* action from the marketplace, to download an artifact.

```
- name: Download artifact
  uses: actions/download-artifact@v2
  with:
    name: production-files
    path: path/to/artifact
```

- name
- : specifies what artifact will be downloaded by the action.
- path
- : specifies the destination where the artifact should be downloaded.

Cache

GitHub Workflow runs often reuse the same downloaded dependencies from one run to another. For example, package and dependency management tools such as npm and pip keep a local cache of downloaded dependencies.

Jobs on GitHub-hosted runners start in a clean virtual environment and must download dependencies each time. This means increased network utilization, longer runtime, and increased cost. GitHub can cache dependencies you frequently use in workflows to reduce time it takes to recreate these files.

To cache dependencies for a job, you'll need to use GitHub's cache action. This action retrieves a cache identified by a unique key. It allows caching dependencies and builds outputs to improve workflow execution time. You can use it to upload cache and then download it in different jobs. Caches allow subsequent pipelines and jobs in the same pipeline to use it and are supposed to speed up the same job across pipelines.

The actions themselves have 3 parameters:

- `path`
- (required): The file path on the runner to cache or restore. The path can be an absolute path or relative to the working directory.
- `key`
- (required): The key created when saving a cache and the key used to search for a cache. It tells GitHub Actions how to uniquely identify each version of your cache.
- `restore-keys`
- (optional): An ordered list of alternative keys to find the cache if no cache hit occurred for
- `key`
- .

Whenever your package file changes, the cache action will create a new cache with a cache key. When a key doesn't match an existing cache, it's called a cache miss, and a new cache is created if the job completes successfully.

Cache helps minimize network utilization, so it is very useful for building a CI/CD pipeline.

Conclusion

We have gone over some workflow tools that ease the workflow building process. There are also many more useful actions in the marketplace. Some of the topics covered are summarized below.

- Checkout: enables your workflow to access the GitHub repository by checking out your repository to the runner.
- Artifacts: are used to share data from one job to another job and also store data once a workflow is complete.
- Cache: gives us the advantage of not downloading frequently used dependencies. It allows caching dependencies and builds outputs to improve workflow execution time.

When is it suitable to use Cache?

To reuse files that don't change often between jobs or workflow runs

Fetch-depth 2

- uses: actions/checkout@v3
with:
fetch-depth: ?
What should be the value of fetch-depth in order to fetch the last commit? 1

When is it suitable to use Artifacts?

To create logs, tests results, and reports

You cache the

`requirements.txt`

package file. Then you add a new package to your project. In this case, if you run your workflow again, will it be a cache hit or miss?

Hint by

differentiate between a file in the package and the package file
Miss

Referencing public action
Report a typo
What are the ways in which you can reference a public action?
Referencing a version

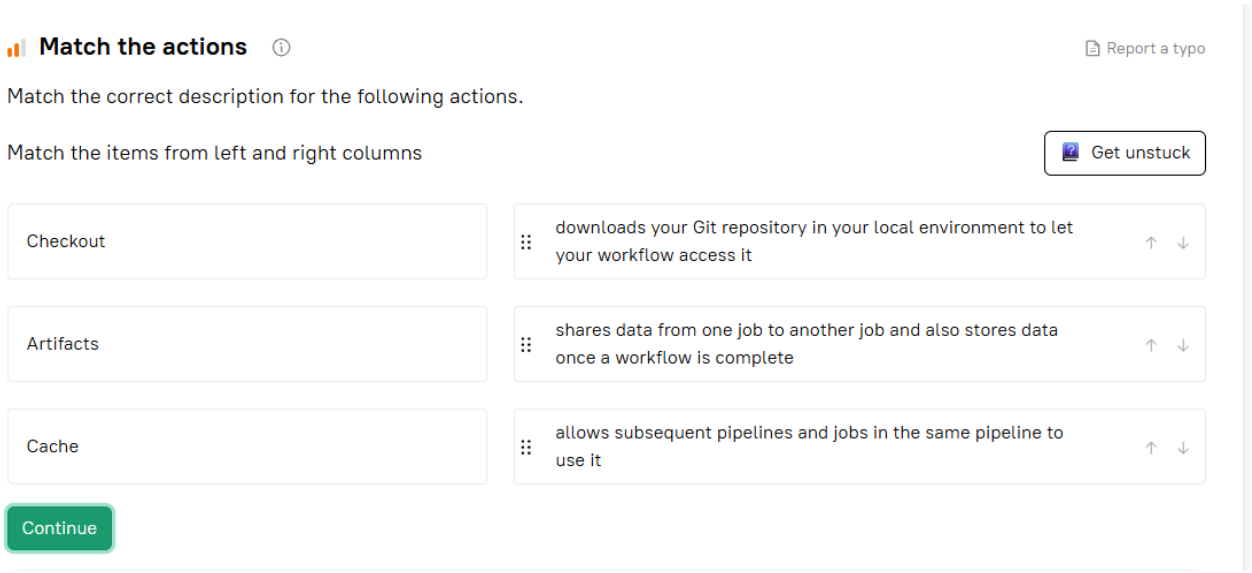
Referencing a commit

Referencing a branch

Fetch-depth 1

- uses: actions/checkout@v3
with:
fetch-depth: ?

What should be the value of `fetch-depth` in order to fetch all history for all tags and branches or the entire commit history? 0



Commit Checkout

Fill out the `uses` step to run the `checkout` action with the repository

`abystoma/external-workflow` and commit reference

`3eb718710f20d7df6874acdc41e189a2ae4182f5` .

Fill out the `uses` step to run the `checkout` action with the repository

`abystoma/external-workflow` and branch named `branch`.

```
name: branch_checkout
on: [push]
jobs:
  external-workflow:
    runs-on: ubuntu-latest
    steps:
      - ??
      - name: Greetings

        run: echo "$(<greetings.txt)"
```

Fill out the `uses` step, then run the workflow in your repo. After successfully running the workflow, go to the `Actions`

tab and click on the name of the workflow. Type the output of the stage "Greetings" below.