

Tudat Developer

None

None

None

Table of contents

1. Welcome	3
1.1 Getting Started	3
1.2 Guides	3
1.3 Reference	3
2. Primer	4
2.1 Documentation	4
3. Reference	5
3.1 Environment Variables	5
3.2 Managing Access Tokens	17
3.3 GitFlow Workflow	30
3.4 Diátaxis Framework Modes	31
3.5 Common Commands	33
3.6 Documentation Styling	34

1. Welcome

Welcome to the Tudat Developer documentation! This documentation is intended for developers who want to contribute to the Tudat project. It contains information on how to set up your development environment, how to build Tudat, and how to contribute to the project.

1.1 Getting Started

To get started, you should first read the Primer section of the documentation. This section contains information on how to set up your development environment, how to build Tudat, and how to contribute to the project.

1.2 Guides

The Guides section contains more in-depth information on specific topics. These guides are intended for developers who want to contribute to the Tudat project.

1.3 Reference

The Reference section contains information on the Tudat codebase. This section is intended for developers who want to contribute to the Tudat project.

2. Primer

2.1 Documentation

Effective documentation is a critical component of any software development project, for several reasons. First, as Guido van Rossum, the creator of Python, has noted, "Code is more often read than written." This means that code that is not properly documented can be difficult to understand and maintain, even for the original author.

As Jeff Atwood, the founder of Coding Horror, has noted, "Code tells you how; Comments tell you why." Documentation provides context and explanation for the code, helping other developers to understand the purpose and functionality of the code, as well as its intended usage.

Finally, as Daniele Procida, the author of "Documentation: An Introduction for Technical Writers and Engineers," has noted, "It doesn't matter how good your software is, because if the documentation is not good enough, people will not use it." In order for software to be widely adopted and successful, it must be accompanied by high-quality documentation that is easy to understand and use.

In summary, effective project documentation is critical for ensuring that code is maintainable, understandable, and usable by other developers. By providing context, explanation, and instruction, documentation can help to ensure that software is widely adopted and successful.

2.1.1 Diátaxis Framework

A systematic framework for technical documentation authoring.

The Diátaxis framework systematically addresses the structure of technical documentation by identifying four documentation modes:

1. **Tutorials:** *Learning-oriented*
2. **How-To Guides:** *Problem-oriented*
3. **Reference:** *Information-oriented*
4. **Explanation:** *Understanding-oriented*

Each mode serves a distinct user need and requires a unique approach to creation. The name Diátaxis comes from the Greek "dia" (across) and "taxis" (arrangement).

The framework advocates for a hierarchical approach to organizing and presenting information, with the four modes of documentation deriving its structure from the relationship between them. Additionally, Diátaxis emphasizes the importance of the following components:

- **Content:** The information presented in the documentation, including text, images, diagrams, and code snippets. The content should be accurate, concise, and easy to understand.
- **Navigation:** The way in which users can move through the documentation to find the information they need. The framework recommends providing multiple ways to navigate the documentation, such as a table of contents, search functionality, and hyperlinks.
- **Presentation:** The way in which the content is visually presented to the user. The framework advocates for a clean and consistent visual design, with a focus on readability and usability. By following the Diátaxis framework, technical writers can create documentation that is well-organized, easy to navigate, and visually appealing, helping to improve user experience and increase the effectiveness of the documentation.

diataxis.png

Example

The document that you are currently reading is an example of the "Explanation" mode of the Diátaxis framework, which provides high-level overviews and context to help users understand a product or technology.

Learn more about the Diátaxis framework from the [Diátaxis Framework](#) website.

3. Reference

3.1 Environment Variables

Note

You are free to use any combination of setting and accessing environment variables that you prefer. You must just ensure that the environment variables are set before you run the script that uses them, and that the environment variables are accessible from the script that uses them.

3.1.1 Python

defined using `os.environ`

In Python, you can define environment variables using the `os.environ` dictionary. The following code snippet demonstrates how to define an environment variable named `VAR` in Python:

```
import os
os.environ["VAR"] = "value"
```

Warning

This is **not** a recommended way to define environment variables for sensitive information such as API keys. This method is not secure, as the environment variables are stored in plain text in the Python script.

defined using `python-dotenv`

An alternative, more secure way to define environment variables in Python is to use the `python-dotenv` library. This library allows you to define environment variables in a `.env` file and load them into your Python script.

1. Install the `python-dotenv` library using `pip`:

```
pip install python-dotenv
```

2. Create a `.env` file in the same directory as your Python script and define the environment variables in the following format:

```
VAR=value
```

3. Load the environment variables into your Python script using the `load_dotenv` function:

```
from dotenv import load_dotenv
load_dotenv()
```

retrieved using `os.environ.get`

Following the definition of environment variables, you can access them in your Python script using the `os.environ.get` function.

```
import os

my_variable = os.environ.get("VAR",
                            None) # Returns None if VAR is not set
if my_variable:
    # Use my_variable in your script
    print("VAR is set to:", my_variable)
else:
```

```
# VAR is not set, handle the error appropriately
printf("VAR is not set.")
```

3.1.2 C++

defined using `setenv`

In C++, you can define environment variables using the `setenv` function.

```
#include <stdlib.h>

setenv("VAR", "value", 1); // 1 indicates that the variable is overwritten if it already exists
                           // 0 indicates that the variable is not overwritten if it already exists
```

retrieved using `getenv`

Following the definition of environment variables, you can access them in your C++ script using the `getenv` function.

```
#include <stdlib.h>

char* my_variable = getenv("VAR");

if (my_variable) {
    // Use my_variable in your script
    std::cout << "VAR is set to: " << my_variable << std::endl;
} else {
    // VAR is not set, handle the error appropriately
    std::cout << "VAR is not set." << std::endl;
}
```

3.1.3 Unix (Linux, macOS)

defined using `export`

In Unix-based systems, environment variables can be defined temporarily in the current terminal session using the `export` keyword. This can also be done within a Bash script (`.sh`).

```
export VAR="value"
```

defined in `~/.bashrc`

In Unix-based systems, environment variables can be defined persistently across terminal sessions by adding the `export` statements to your shell profile file.

1. Open your shell profile file using a text editor (e.g. `nano ~/.bashrc`).
2. Add the `export` statements to the file (preferably at the end).
3. Save the file and close the text editor.
4. Restart your terminal session.
5. Verify that the environment variables are set by running the following command:

```
env | grep VAR
```



If you are using `zsh`, you should use `~/zshrc` instead of `~/.bashrc`.

retrieved using \$VAR

Following the definition of environment variables, you can access them in your Bash script using the `$VAR` syntax.

```
if [ -n "$VAR" ]; then
    # Use $VAR in your script
    echo "VAR is set to: $VAR"
else
    # VAR is not set, handle the error appropriately
    echo "VAR is not set."
fi
```

3.1.4 Windows**defined using set**

In Windows, environment variables can be defined temporarily in the current terminal session using the `set` keyword. This can also be done within a Batch script (`.bat`).

```
set VAR=value
```

defined using setx

In Windows, environment variables can be defined persistently across terminal sessions by adding the `setx` statements to your system environment variables.

1. Open the "System Properties" window by pressing `Windows + Pause/Break` on your keyboard.
2. Click "Advanced system settings" on the left.
3. Click "Environment Variables" on the bottom right.
4. Click "New" under "System variables" and add the environment variables.
5. Click "OK" to save the changes.
6. Restart your terminal session.
7. Verify that the environment variables are set by running the following command:

```
set | findstr VAR
```

retrieved using %VAR%

Following the definition of environment variables, you can access them in your Batch script using the `%VAR%` syntax.

```
if defined VAR (
    rem Use %VAR% in your script
    echo VAR is set to: %VAR%
) else (
    rem VAR is not set, handle the error appropriately
    echo VAR is not set.
)
```

3.1.5 Azure Pipelines

This section is intended for maintainers of the feedstock repositories (at current). If you are not a maintainer, you can safely ignore this section.

1. Navigate to dev.azure.com and sign in.

The screenshot shows the Azure DevOps homepage. At the top, there's a navigation bar with links for Azure, Explore, Products, Solutions, Pricing, Partners, Resources, Search, Learn, Support, Contact Sales, a green 'Free account' button, and a red-bordered 'Sign in' button. Below the navigation is a large illustration of a rocket launching from a platform with people working on it, symbolizing continuous integration and deployment. The main heading 'Azure DevOps' is displayed, followed by the subtext 'Plan smarter, collaborate better, and ship faster with a set of modern dev services.' There are two prominent buttons: 'Start free' and 'Start free with GitHub'. Below these are links for 'Overview', 'Features', 'Security', and 'Get started'. The main content area features six service tiles: 'Azure Boards' (agile planning), 'Azure Pipelines' (CI/CD), 'Azure Repos' (Git repositories), 'Azure Test Plans' (test management), 'Azure Artifacts' (package management), and 'Extensions Marketplace' (third-party extensions). A 'Chat with sales' button is located in the bottom right corner of the main content area.

Azure DevOps

Plan smarter, collaborate better, and ship faster with a set of modern dev services.

Start free Start free with GitHub

Overview Features Security Get started

Azure Boards
Deliver value to your users faster using proven agile tools to plan, track, and discuss work across your teams.
[Learn more >](#)

Azure Pipelines
Build, test, and deploy with CI/CD that works with any language, platform, and cloud. Connect to GitHub or any other Git provider and deploy continuously.
[Learn more >](#)

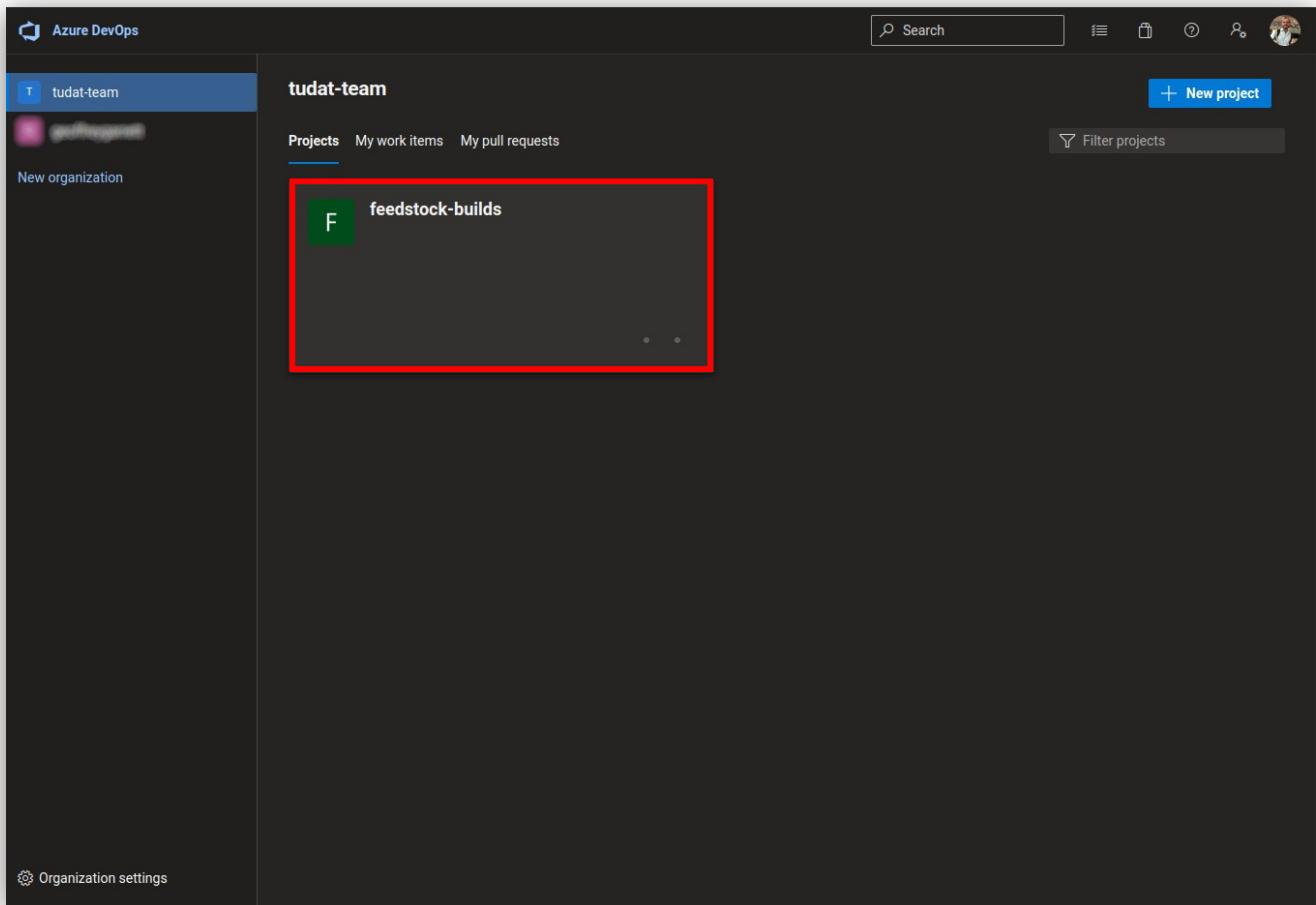
Azure Repos
Get unlimited, cloud-hosted private Git repos and collaborate to build better code with pull requests and advanced file management.
[Learn more >](#)

Azure Test Plans

Azure Artifacts

Extensions Marketplace
Access extensions from Slack to SonarQube
[Chat with sales](#)

1. Click the `feedstock-builds` project under the [tudat-team organization](#).



defined for a pipeline

1. After [navigating to the project](#), click " Pipelines" under the `feedstock-builds` project.

The screenshot shows the Azure DevOps interface for the project 'feedstock-builds'. The left sidebar contains navigation links: Overview, Summary, Dashboards, Wiki, Pipelines (which is highlighted with a red box), and Artifacts. The main content area displays the project's status: 'About this project' (with a placeholder for a project description), 'Project state' (Build succeeded, 0% Deployment success), and 'Members' (a list of team members). The top right corner includes a search bar, a 'Public' button, an 'Invite' button, and a user profile icon.

1. Click on the specific pipeline in which you want to set the environment variable (e.g. tudat-feedstock).

The screenshot shows the Azure DevOps Pipelines interface for the project 'feedstock-builds'. The left sidebar includes options like Overview, Pipelines (selected), Pipelines, Environments, Releases, Library, Task groups, Deployment groups, and Artifacts. The main area displays a list of 'Recently run pipelines' with columns for Pipeline, Last run, and details. A red box highlights the first entry: 'tudat-feedstock' with the ID '#20230215.1', status 'BOT: Changes detected in project, nightly release...', run by 'Individual CI for develop', and last run '13h ago' ago. Other pipelines listed include 'tudat-nightly', 'tudat-nightly-nightly', 'calibration-nightly', 'navigation-nightly', and 'calibration-nightly'.

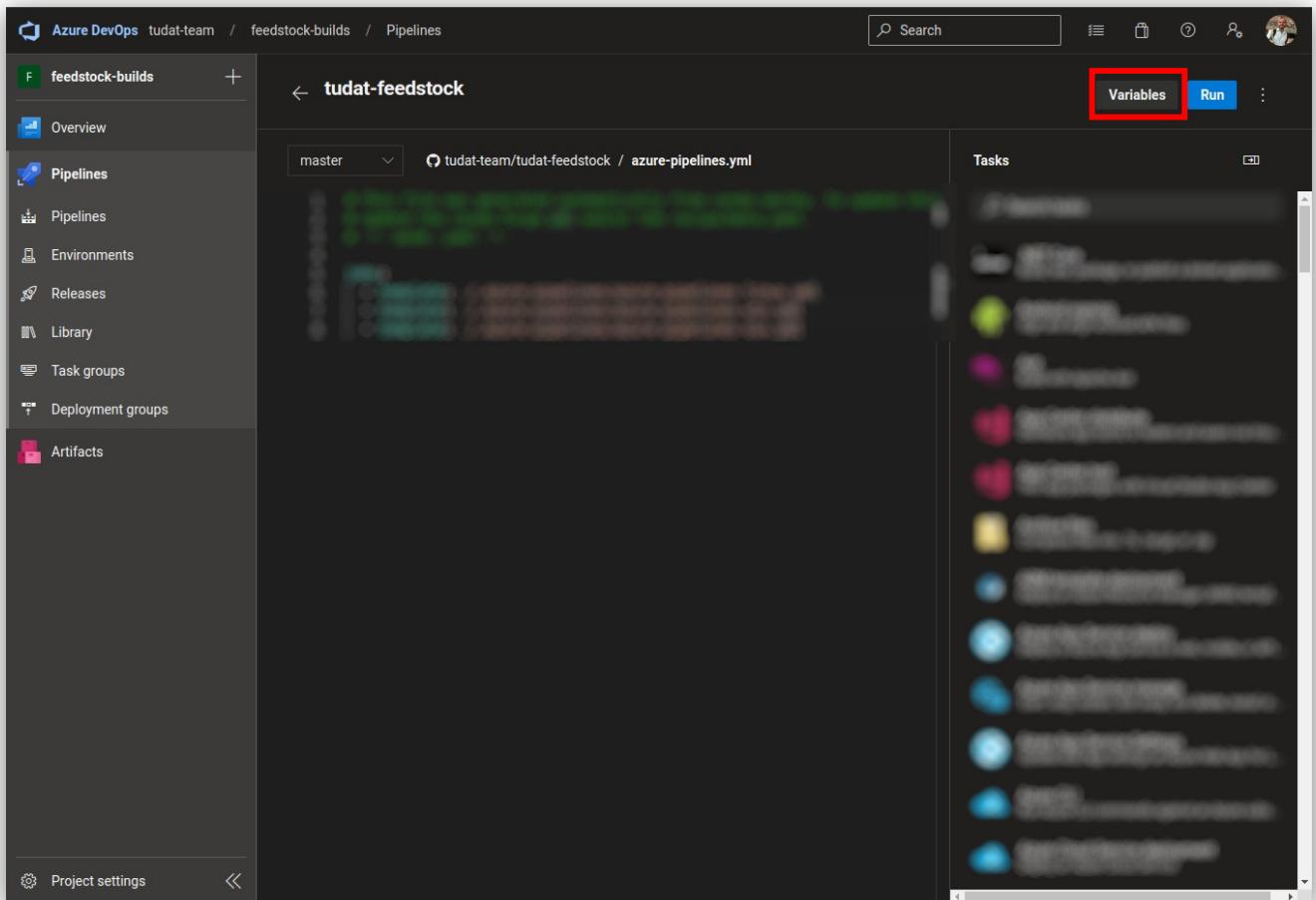
1. Click "Edit" in the top-right of the page.

The screenshot shows the Azure DevOps interface for a project named 'feedstock-builds'. The left sidebar has a dark theme with the following navigation items:

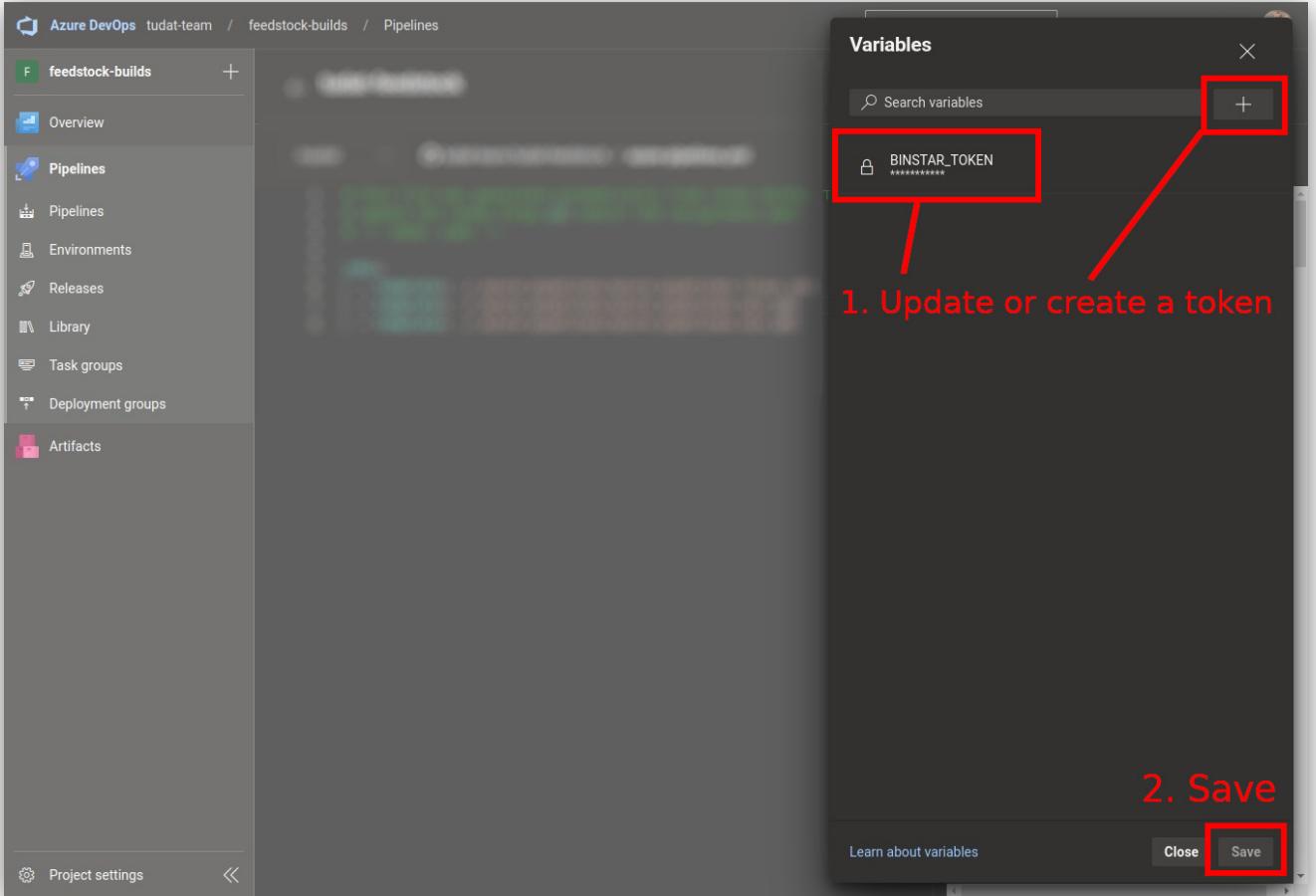
- Overview
- Pipelines (selected)
- Environments
- Releases
- Library
- Task groups
- Deployment groups
- Artifacts

The main area displays a list of pipeline runs for the 'tudat-feedstock' repository. The 'Runs' tab is selected. The first run is highlighted with a pink circle. The 'Edit' button in the top right corner is highlighted with a red box.

1. Click "Variables" in the top-right of the page.



1. Update an existing token (e.g. `BINSTAR_TOKEN`), or create a new one with the `+` icon.



!!! note It is crucial to keep the environment variables private and secure. Ensure that you do not share them or commit them to a public repository, as this could lead to security issues.

defined across all pipelines

Failure

This section is the desired way of defining environment variables in Azure. However, it is currently not possible to define environment variables in an Azure project due to `conda-smithy` overwriting the `pipeline.yaml`. This is a known issue, and we are working on a solution. In the meantime, please use the method described in the previous section. This section therefore exists for future reference.

1. After navigating to the project, click " Library" under the `feedstock-builds` project and select the target " variable group".

The screenshot shows the Azure DevOps interface for the project 'feedstock-builds'. The left sidebar has a red box around the 'Library' item, labeled '1'. The main area shows a table titled 'Variable groups' with one entry: 'fx conda-smithy' (modified on 23/01/2023). A red box highlights this entry, labeled '2'. The top right has a search bar and other navigation icons.

Name	Date modified	Modified by	Description
fx conda-smithy	23/01/2023	[Redacted]	

1. Add a new variable by clicking `+ Add` or edit an existing one by clicking the name of value of the variable.

Variable group name: conda-smithy

Name ↑	Value
BINSTAR_TOKEN	*****
READTHEDOCS_TOKEN	*****

+ Add

Add new or update existing token

1. Make the variable group available by adding the following lines to the `pipeline.yaml` file in the root of the feedstock repository.

```
variables:
  - group: <variable_group_name>
```

!!! note The `<variable_group_name>` should be the name of the variable group you created in step 1.

!!! failure As mentioned, this will be overwritten by `conda-smithy`. It was expected that by adding:

```
```yaml
azure:
 variables:
 - group: <variable_group_name>
```

```

To the `'conda-forge.yml'` file, the `'pipeline.yaml'` would be updated accordingly, as [hinted at here](https://conda-forge.org/docs/maintainer/conda_forge_yml.html#azure). However, this is not the case. We are working on a solution.

3.2 Managing Access Tokens

An important part of the continuous integration (CI) process is the use of access tokens to authenticate the user to the relevant services. This section describes how to generate access tokens for the following services:

- [Anaconda Cloud](#)
- [GitHub](#)

3.2.1 Generating Access Tokens

Access tokens are used to authenticate the user to the relevant service. The following sections describe how to generate access tokens for the following services:

Organization Tokens on Anaconda Cloud**Note**

This section is intended for maintainers of the feedstock repositories (at current). If you are not a maintainer, you can safely ignore this section.

1. Navigate to <https://anaconda.org/> and sign in.

The screenshot shows the Anaconda.org homepage. At the top right, there is a red box highlighting the "Sign In" button. Below the header, there's a "Join Today" form with fields for Username, Email Address, Enter Password, and Re-enter Password. There's also a checkbox for accepting Terms & Conditions and a reCAPTCHA field. A "Register For Free" button is at the bottom of the form. A banner at the bottom of the page encourages users to expedite their data science journey with access to training materials and expert insights on Anaconda Nucleus.

ANACONDA.ORG

Gallery About Anaconda Help Download Anaconda Sign In

ANACONDA.ORG

Where packages, notebooks, projects and environments are shared.

SEARCH PACKAGES

Search Anaconda.org

Join Today

Sign Up Sign In

Username
Pick a username

Email Address
Your email

Enter Password
SecretPassword

Re-enter Password
SecretPassword

I accept the [Terms & Conditions](#)

I'm not a robot 
reCAPTCHA
Privacy - Terms

Register For Free

Expedite your data science journey with easy access to training materials, how-to videos, and expert insights on Anaconda Nucleus, all free for a limited time to Nucleus members.

2. Go to the organization page, for example, if the organization is named "tudat-team", the URL would be <https://anaconda.org/tudat-team>.
3. Click on the organization profile ("tudat-team") at the top-right of the page, followed by "Settings".

The screenshot shows the Anaconda Landscape interface. At the top right, there is a user profile dropdown menu labeled "tudat-team". A red box labeled "1" highlights this menu. A second red box labeled "2" highlights the "Settings" option within the dropdown menu. The main interface displays sections for Packages, Notebooks, Environments, Projects, Favorites, and Activity Feed.

4. Fill in the details for creating a new token: name your token, provide the appropriate scope, and specify an expiration date. Then, click "Create".

The screenshot shows the 'API Tokens' section of the Anaconda.org website. On the left, there's a sidebar with links: 'Public Profile', 'Account', 'Billing', 'Access' (which is highlighted in green), 'Security Log', and 'Storage'. The main area is titled 'API Tokens ?' and contains a form for creating a new access token for the organization 'tudat-team'. The form fields include:

- Token Name:** my-token
- Strength:** Strong (longer token)
- Scopes:** A list of checkboxes. The first one, 'Allow all operations', is checked (indicated by a blue checkmark). The other options are empty checkboxes.
- Expiration date (YYYY/MM/DD):** 2024/02/14

At the bottom right of the form is a large green 'Create' button. A red box labeled '1' surrounds the entire form area, and a red box labeled '2' surrounds the 'Create' button.

5. The newly created organization token will be displayed. Copy the token and keep it safe, as it will only be shown once.

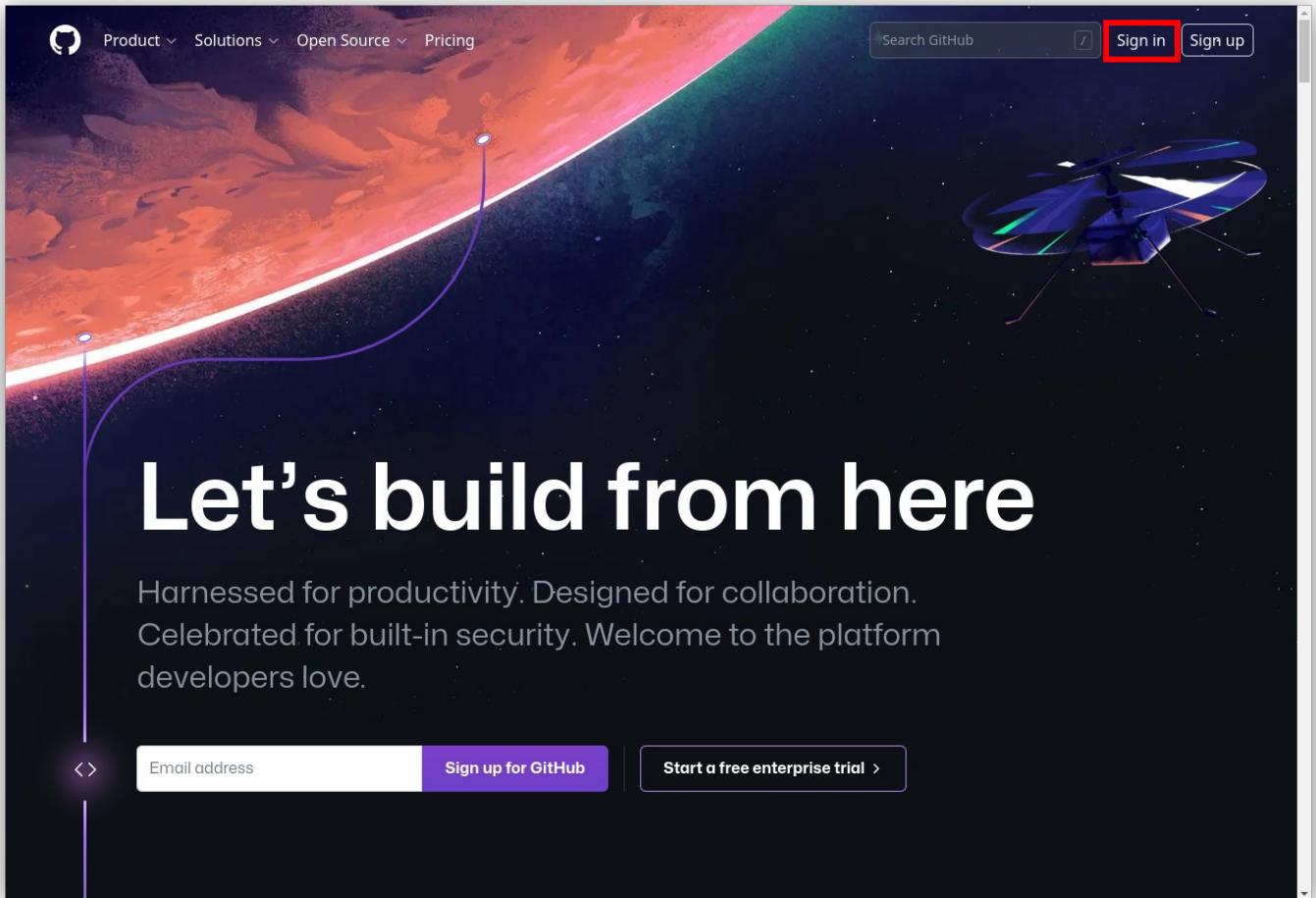
The screenshot shows the Anaconda.org user interface. The top navigation bar includes the Anaconda logo, a search bar, and a user dropdown for 'tudat-team'. A green banner at the top indicates 'Created token tu-3c1'. The left sidebar has links for 'Public Profile', 'Account', 'Billing', 'Access' (which is highlighted in green), 'Security Log', and 'Storage'. The main content area is titled 'API Tokens' and contains a 'Create access token for tudat-team' button. Below it, there's a form for creating a new token: 'Token Name' (empty), 'Strength' (set to 'Strong (longer token)'), 'Scopes' (checkboxes for various API operations like 'Allow all operations', 'Allow all API operations', etc.), and 'Expiration Date' (set to '2024/02/14'). A large green 'Create' button is at the bottom.

Important

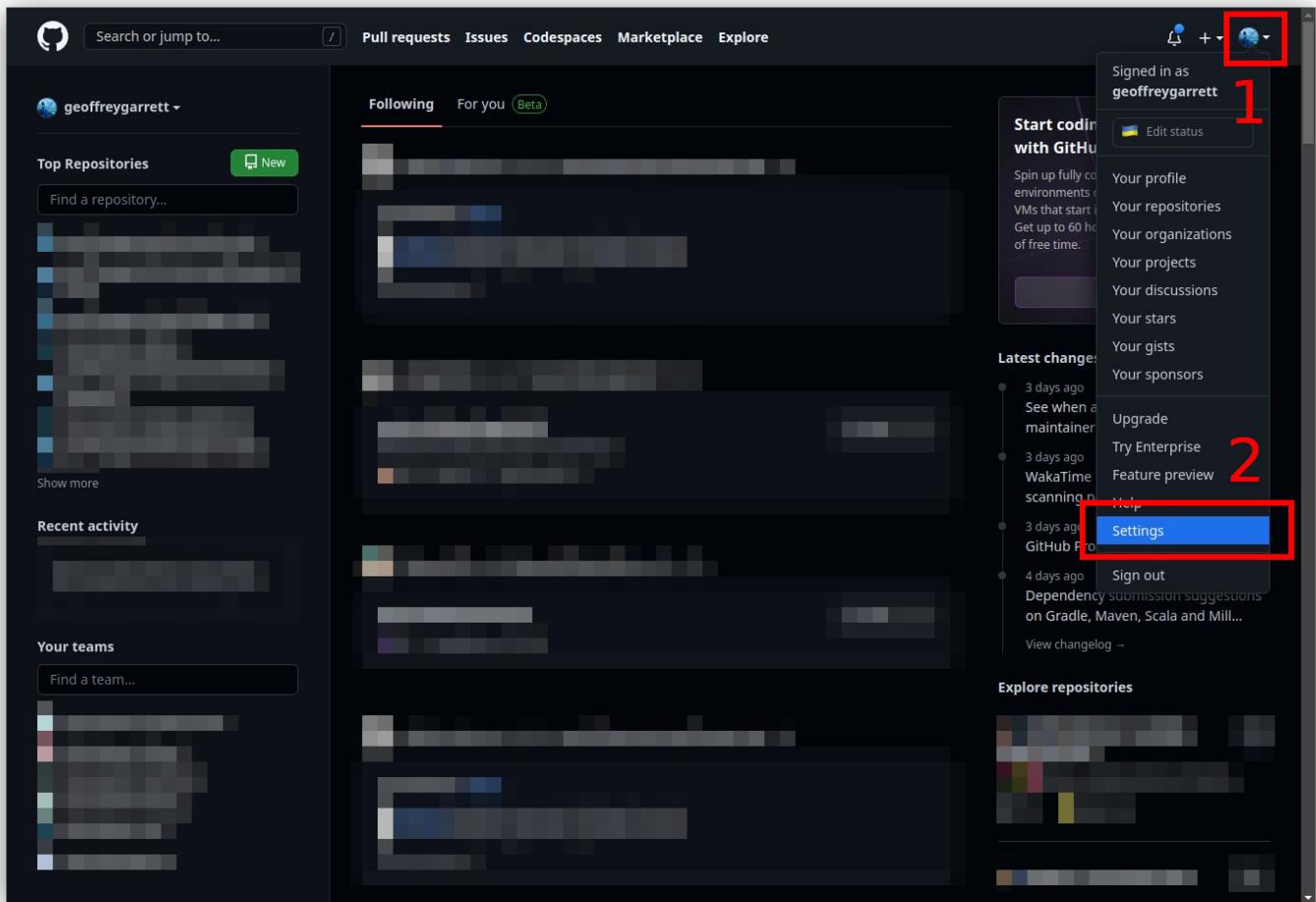
It is crucial to keep the organization token private and secure. Ensure that you do not share it or commit it to a public repository, as this could lead to security issues.

User Tokens on GitHub

1. Navigate to [GitHub](#) and sign in.

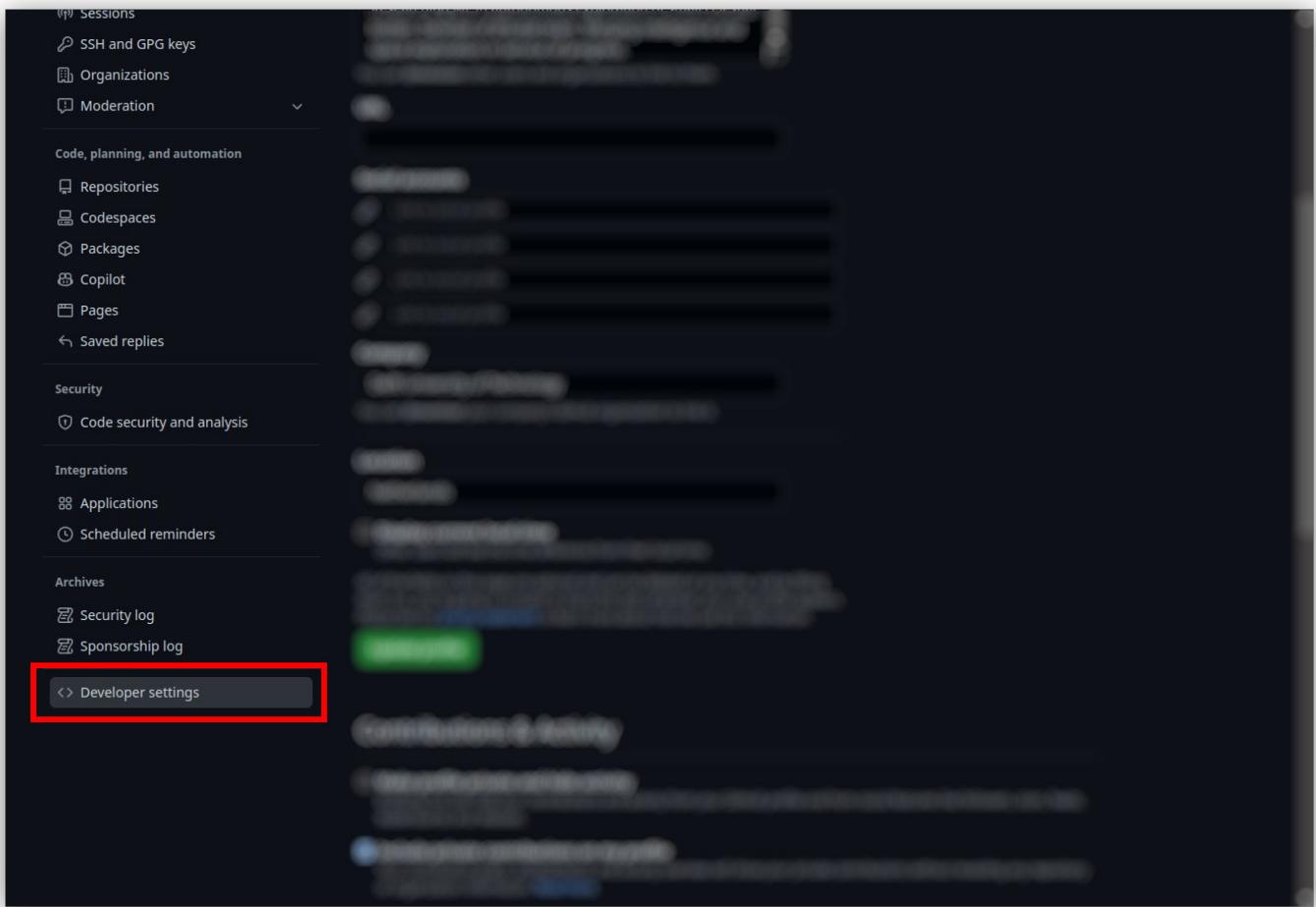


2. Click on your profile picture at the top-right of the page to reveal a drop-down menu, select "Settings".



Dark mode makes you a better developer.

3. Click on "Developer settings".



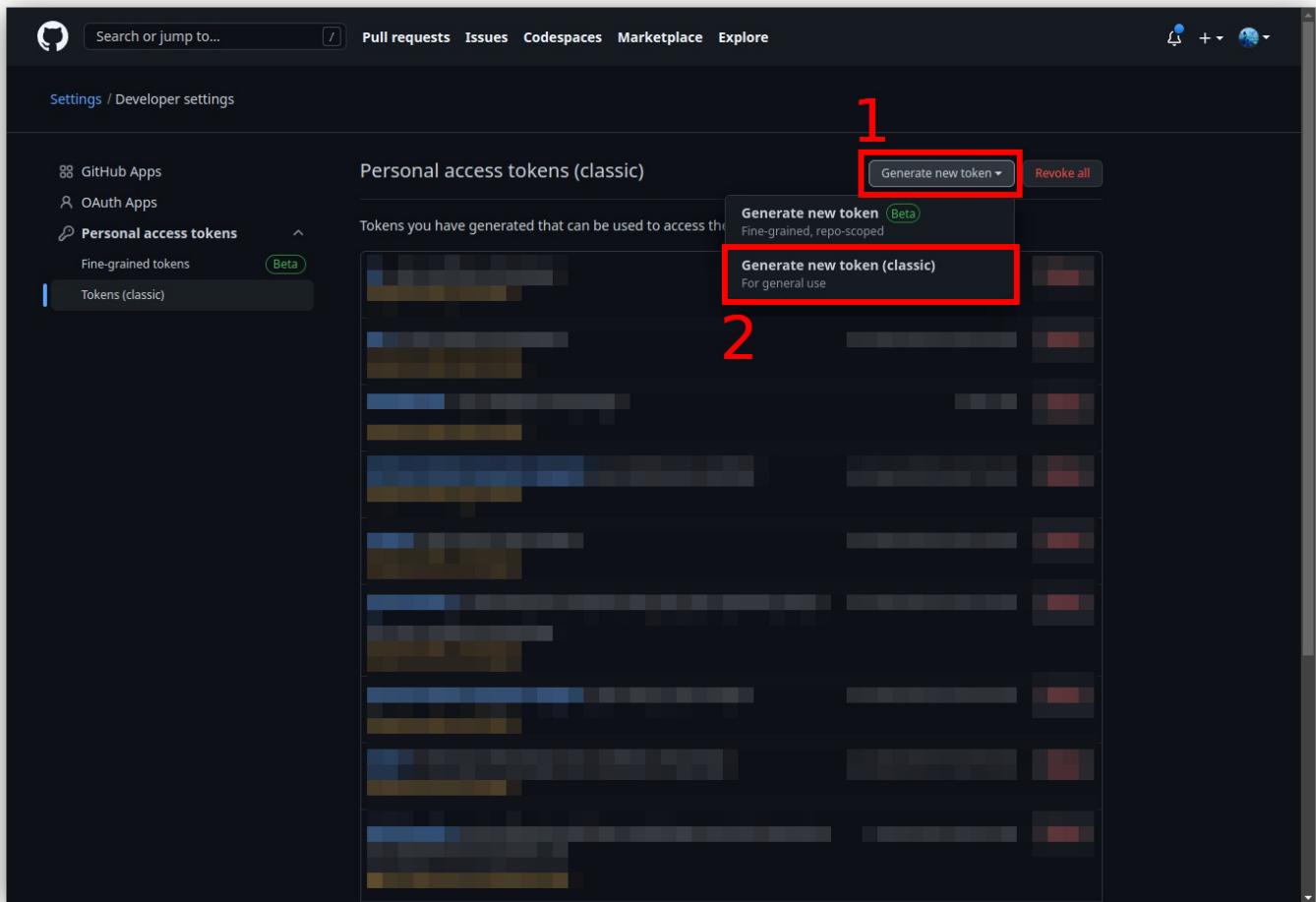
4. Click on "Personal access tokens".

The screenshot shows the GitHub Developer settings page. At the top, there is a navigation bar with links for Pull requests, Issues, Codespaces, Marketplace, and Explore. Below the navigation bar, the path Settings / Developer settings is displayed. On the left, there is a sidebar with two main sections: GitHub Apps and OAuth Apps. The GitHub Apps section is currently selected and expanded, showing a dropdown menu with options: Personal access tokens (which is highlighted with a red box), OAuth tokens, and API tokens. To the right of the sidebar, the main content area is titled "GitHub Apps" and contains a message encouraging users to build GitHub integrations. A "New GitHub App" button is located in the top right corner of this area. At the bottom of the page, there is a footer with copyright information and links to Terms, Privacy, Security, Status, Docs, Contact GitHub, Pricing, API, Training, Blog, and About.

Note

Currently, we use the classic tokens. Any developer is free to use the new fine-grained tokens, which are in Beta at the time of writing the current documentation, so long as it works. That developer must just modify the documentation accordingly.

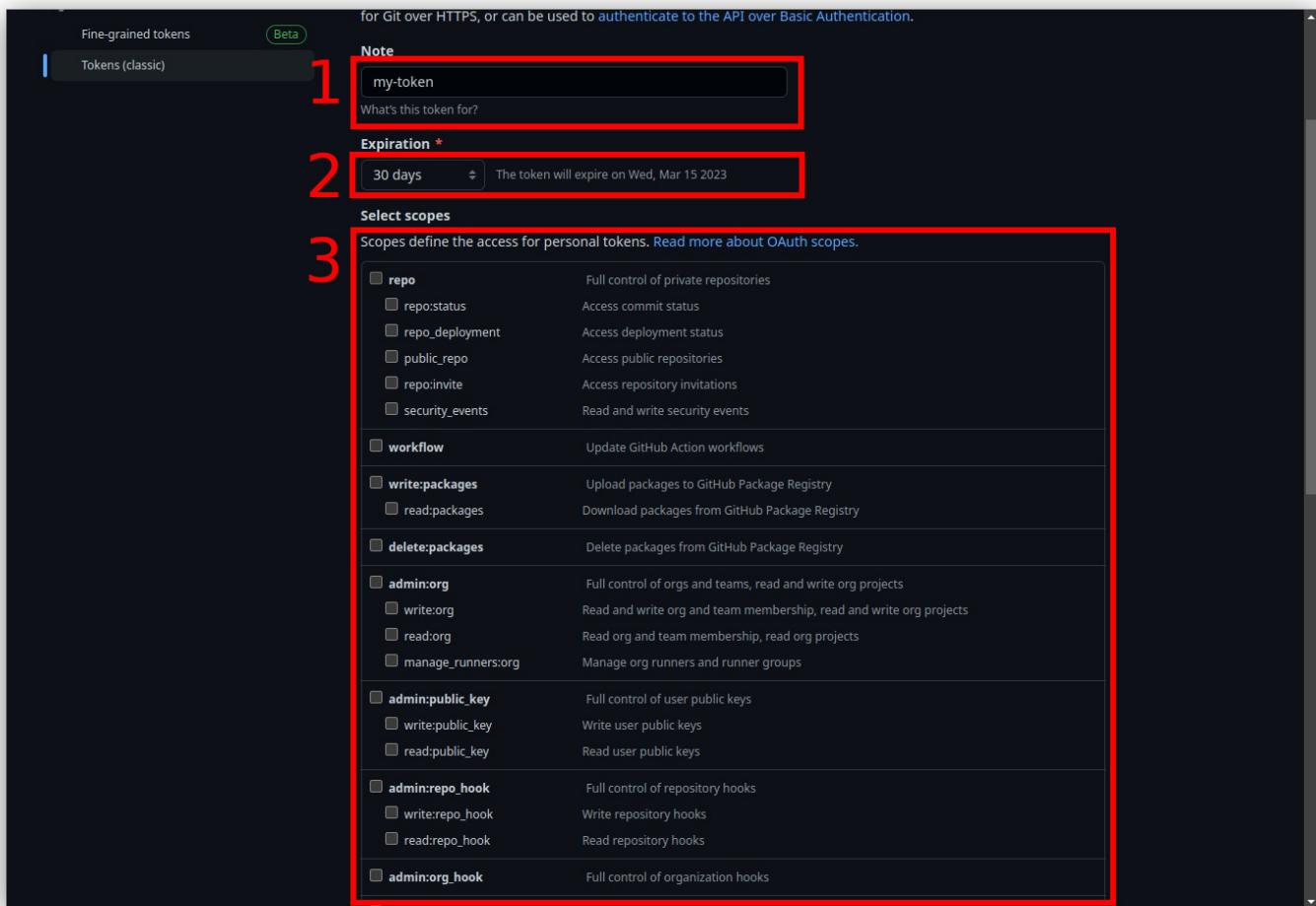
5. Click on "Generate new token (classic)".



Important

The correct selection of the scopes is pertinent to the continued operation of the dependent systems. This may influence one to over-scope the permissions, in order to avoid having to scrutinize the required ones to save time, however over-scoping is a bad practice and should be avoided, so as to avoid any malicious attempts. See the documentation stating the required scopes from the relevant service.

6. Name your token, set an expiration time-delta, select the appropriate scopes, and finally click "Generate token".



7. Click on the copy icon provided, next to the token (or copy it yourself).

The screenshot shows the GitHub developer settings page under 'Personal access tokens (classic)'. A specific token, 'ghp_14', is selected, and a red box highlights the copy icon (a clipboard with a plus sign) located to the right of the token name. Other tokens are listed below, each with a copy icon and a delete button.

Important

It is important to never commit code with the token. If this is done, the token will be deactivated immediately, automatically in most cases by the relevant service. For example, if you commit a Github token to Github, it will be automatically deleted. This is a good safety feature to have, but if done with a token which multiple components in our system are dependent on, the amount of work to rectify this will be tedious and better avoided through taking mental note of this now.

3.2.2 Storing Access Tokens

Access tokens should be stored in a secure manner, so that they are not exposed to the public. This is done by storing them as environment variables, which are not committed to the repository.

Azure Pipelines

See [Environment Variables on Azure Pipelines](#).

3.3 GitFlow Workflow

- Want to add a feature? See [Feature](#)
- Want to release a new stable version? See [Release](#)
- Want to fix a non-critical bug? See [Bugfix](#)
- Want to fix a bug in a release? See [Hotfix](#)

```
gitGraph commit tag: "v1.0.0" id: "Initial commit" type: HIGHLIGHT branch release/v1.1.0 branch develop branch feature/feature1 branch feature/feature2 branch hotfix/v1.0.1 branch bugfix/v1.1.1
```

3.3.1 Develop

```
gitGraph commit commit branch develop checkout develop commit commit checkout main merge develop commit commit
```

3.3.2 Feature

```
%%init: { 'logLevel': 'debug', 'theme': 'base', 'gitGraph': {'rotateCommitLabel': false}} }%% gitGraph commit tag: "v1.0.0" branch release/v1.1.0 branch develop commit branch feature/feature1 commit commit commit checkout develop commit branch feature/feature2 commit id: "feat(api): ..." commit checkout develop merge feature/feature1 merge feature/feature2 commit checkout release/v1.1.0 merge develop checkout main merge release/v1.1.0 commit tag: "v1.1.0"
```

3.3.3 Release

3.3.4 Hotfix

3.3.5 Bugfix

3.4 Diátaxis Framework Modes

The Diátaxis framework is a systematic approach to technical documentation that was developed to address the challenge of structuring documentation in a clear and logical way. The framework consists of four distinct modes of documentation, each of which serves a distinct user need and requires a unique approach to creation.

| Mode | Focus | Purpose |
|---------------------|------------------------|--|
| Tutorials | Learning-oriented | Used to teach new users how to use a product or technology. |
| How-To Guides | Problem-oriented | Provide solutions to specific problems or use cases. |
| Technical Reference | Information-oriented | Provides detailed information on the features, functions, and capabilities of a product or technology. |
| Explanation | Understanding-oriented | Provides conceptual information and context to help users understand a product or technology. |

diataxis.png

By adopting the Diátaxis framework, technical writers can ensure that their documentation is tailored to the user's needs and provides the information they need in a clear and concise manner. The framework has gained widespread adoption in the technical writing community and is considered a best practice for creating effective technical documentation.

3.4.1 Tutorials

Tutorials walk users through a process or task step-by-step. They should be structured in a clear and logical way, with each step building on the previous one. Tutorials should also include visuals and examples to help users understand the process.

Example: A tutorial on how to create a new user account in a software application.

Kemember

- Clearly state the learning objective
- Break down the task into clear and concise steps
- Use images, diagrams, and examples to aid understanding
- Provide feedback and reinforcement to encourage learning

3.4.2 How-To Guides

How-to guides provide solutions to specific problems or use cases. They should be focused and concise, providing clear and actionable steps to solve the problem.

Example: A how-to guide on how to troubleshoot a connectivity issue with a networking device.

Kemember

- Clearly state the problem or use case
- Provide a clear and concise solution
- Use screenshots or other visuals to help users understand the solution
- Test the solution to ensure it works as expected

3.4.3 Technical Reference

Technical reference documentation provides detailed information on the features, functions, and capabilities of a product or technology. It should be well-organized and easy to navigate, with a clear and concise description of each feature or function.

Example: Technical reference documentation for a software library, detailing each of its APIs and functions.

Remember

- Clearly define each feature or function
- Provide detailed information on each feature or function, including syntax, parameters, and return values
- Organize the information in a clear and logical manner
- Provide examples of how to use each feature or function

3.4.4 Explanation

Explanation provides conceptual information and context to help users understand a product or technology. It should be clear and concise, providing context and explanation in a way that is easy to understand.

Example: An explanation of the importance of project documentation and an overview of the Diátaxis framework.

Remember

- Provide high-level overviews of the product or technology
- Provide conceptual information and context to help users understand the product or technology
- Explain why the product or technology is important and how it fits into the broader technology landscape
- Use analogies or other relatable examples to aid understanding

3.5 Common Commands

3.5.1 conda smithy

```
conda smithy rerender --feedstock_directory ./foo-feedstock
```

Concerning the `conda-forge.yml`

A change to the `conda-forge.yml` file will not automatically trigger a re-rendering of the feedstock. To have the changes to `conda-forge.yml` reflected in the feedstock, you need to manually update the file and then re-render the feedstock.

3.5.2 mkdocs

```
mkdocs new [dir-name] # Create a new project.
```

```
mkdocs.yml # The configuration file.  
docs/  
    index.md # The documentation homepage.  
    ...       # Other markdown pages, images and other files.
```

```
mkdocs serve # Start the live-reloading docs server.
```

```
mkdocs build # Build the documentation site.
```

```
mkdocs -h # Print help message and exit.
```

Further reference

- [MkDocs: Project documentation with Markdown](#)

3.5.3 Project layout

3.6 Documentation Styling

Further reference

- [MkDocs Material: Our chosen theme](#)
- [MkDocs Material Reference: Best quick-hand reference for styling](#)