Persisting environment variables and temporary files

In this article

Setting persistent environment variables

Preventing temporary files from being automatically deleted

Further reading

You can configure custom environment variables so that they are set to the same value every time you open a codespace. You can also ensure that temporary files are not deleted when a codespace stops.

Setting persistent environment variables

You can set persistent custom environment variables in multiple ways, depending on which codespaces, repositories, or users you want the variables to be available to.

For all the methods of setting custom variables listed below, you can access the custom variable in your codespace by using syntax like `echo $VARNAME`.

For a single codespace

You can set the value of the environment variable in the `~/.bashrc` file, or in an equivalent configuration file if you are not using the Bash shell. For example, add the statement `VARNAME=value`.

After you save the change to this file, the value will be set the next time you open the codespace, or you can set it immediately by using a command such as `source ~/.bashrc`. The variable will remain set if you stop and start the codespace. However, changes to files in the home directory will be reset if you rebuild the container, so variables set in the `~/.bashrc` file will not persist over a rebuild. For more information, see "Preventing temporary files from being automatically deleted."

For all codespaces for a repository

There are three ways that you can set persistent custom environment variables for all codespaces that you create for a repository:

- You can edit the `devcontainer.json` configuration file for the repository
- You can use a custom Dockerfile
- You can use secrets

Edit the `devcontainer.json` configuration file for the repository

Edit the `devcontainer.json` configuration file for the repository, and use the `remoteEnv` property to set the environment variable value:

```
{
    "remoteEnv": {
      "VARNAME": "value"
   }
}
```

Only use this method for values that you are happy to commit to your repository as plaintext. For sensitive values such as access tokens, use secrets.

The environment variable will be set within your editor's remote server process, and will be available for sub-processes of that remote server process, such as terminals and debugging sessions. However, the variable will not be available more broadly inside the container. This method is useful if you don't need the environment variable to be set for other background processes that run at startup, and if you are using a premade image and don't have or want a custom Dockerfile.

This setting will take effect when you rebuild your container or create a new codespace after pushing this change to the repository. For more information about applying configuration changes to a codespace, see "Introduction to dev containers."

Use a custom Dockerfile

If you are using a custom Dockerfile you can set the environment variable there by adding `ENV VARNAME=value`.

This method is useful if you already have a Dockerfile and want to set a variable on a container-wide level.

This setting will take effect when you rebuild your container or create a new codespace after pushing this change to the repository. For more information about applying configuration changes to a codespace, see "Introduction to dev containers."

Use secrets

You can use secrets for GitHub Codespaces to set custom variables for codespaces created for the repository. For more information, see "Managing secrets for your codespaces."

You should use this method for environment variable values that you do not want to commit to the repository as plaintext.

This setting will take effect the next time you create a codespace for this repository, or when you restart an existing codespace.

For all codespaces that you create

If you want to set a personalized environment variable for all codespaces that you create you can set this using a file in your `dotfiles` repository. For example, add `VARNAME=value` in the `.bash_profile` file. Environment variables you set in a dotfile are personal to you and are not set for anyone else. For more information about Dotfiles, see "Personalizing GitHub Codespaces for your account."

Preventing temporary files from being automatically deleted

When you create a codespace, your repository is cloned into the `/workspaces` directory in your codespace. This is a persistent directory that is mounted into the container. Any changes you make inside this directory, including editing, adding, or deleting files, are preserved when you stop and start the codespace, and when you rebuild the container in

the codespace.

Outside the `/workspaces` directory, your codespace contains a Linux directory structure that varies depending on the image used to build your codespace. You can add files or make changes to files outside the `/workspaces` directory: for example, you can install new programs, or you can set up your shell configuration in a file such as `~/.bashrc`. As a non-root user, you may not automatically have write access to certain directories, but most images allow root access to these directories with the `sudo` command.

Outside `/workspaces`, with the exception of the `/tmp` directory, the directories in a codespace are tied to the lifecycle of the container. This means any changes you make are preserved when you stop and start your codespace, but are not preserved when you rebuild of the container. For information about creating symlinks to preserve data outside the `/workspaces` directory, see "Rebuilding the container in a codespace."

The `/tmp` directory is an exception because it is mounted into the container, but it is not persistent. Therefore, the contents of the `/tmp` directory are persisted over a rebuild, but are cleared each time the codespace stops. For example, the `/tmp` directory is cleared when a codespace session times out after a period of inactivity. For more information, see "Setting your timeout period for GitHub Codespaces."

If you have temporary files that you want to be available the next time you start the codespace, do not save them in the `/tmp` directory.

Further reading

- "Changing the shell in a codespace"