circleci

HOME DOCS ORBS IMAGES

Introduction to environment variables

1+ year ago 5 min read

Cloud Server v4.x Server v3.x

Helpful Resources

- Keep environment variables private
- Troubleshoot environment variables settings
- Insert files as environment variables

On This Page

- Introduction
- Built-in environment variables
- Private keys and secrets
- Secrets masking
- Environment variable usage options
  - Order of precedence
  - Example configuration of environment variables
  - Parameters and bash environment
  - Environment variable substitution
    - Usage
  - Alpine Linux
- Notes on security
- Contexts
- See also

Ask AI

Introduction

Use environment variables to set up various configuration options, and keep your set-up secure with secrets, private keys, and contexts. Environment variables in CircleCI are governed by an order of precedence, allowing control at each level in your configuration. See the Set an environment variable page for guidance on the different ways to set an environment variable.

If you have existing environment variables (or contexts) and you would like to rename your organization or repository, please follow the Rename organizations and repositories guide to make sure you do not lose access to environment variables or contexts in the process.

Built-in environment variables

All projects have access to CircleCI's built-in environment variables. These environment variables are scoped at the job level, so they can be used with the `context` key in a job, but they do not exist at a pipeline level.

For a full list of built-in environment variables, see the Project values and variables page.

Private keys and secrets

To add private keys or secrets as environment variables for use throughout your project, navigate to **Project Settings > Environment Variables** in the CircleCI web app ↗ . You can find step-by-step instructions of this process on the Environment variables page. The variable values are neither readable nor editable in the app after they are set. To change the value of an environment variable, delete the current variable, and add it again with the new value.

Private environment variables enable you to store secrets safely, even when your project is public. Refer to the Building open source projects page for associated security and settings information.

Secrets masking

Environment variables and contexts may hold project secrets or keys that perform crucial functions for your applications. Secrets masking provides added security within CircleCI by obscuring environment variables in the job output when `echo` or `print` is used.

Secrets masking is applied to environment variables set within **Project Settings** or **Contexts** in the web app.

The value of the environment variable or context will *not* be masked in the job output if:

- the value of the environment variable is less than 4 characters
- the value of the environment variable is equal to one of `true`, `True`, `false`, or `Fals`

> ⚠ Secrets masking will only prevent values from appearing in your job output. Invoking a bash shell with the `-x` or `-o xtrace` options may inadvertently log unmasked secrets (please refer to Using shell scripts). If your secrets appear elsewhere, such as test results or artifacts, they will not be masked. Additionally, values are still accessible to users debugging builds with SSH.

> ⚠ The secrets masking feature exists as a preventative measure to catch unintentional display of secrets at the output. Best practice is to avoid printing secrets to the output. There are many ways that secrets masking could be bypassed, either accidentally or maliciously. For example, any process that reformats the output of a command or script could remove secrets masking.

Environment variable usage options

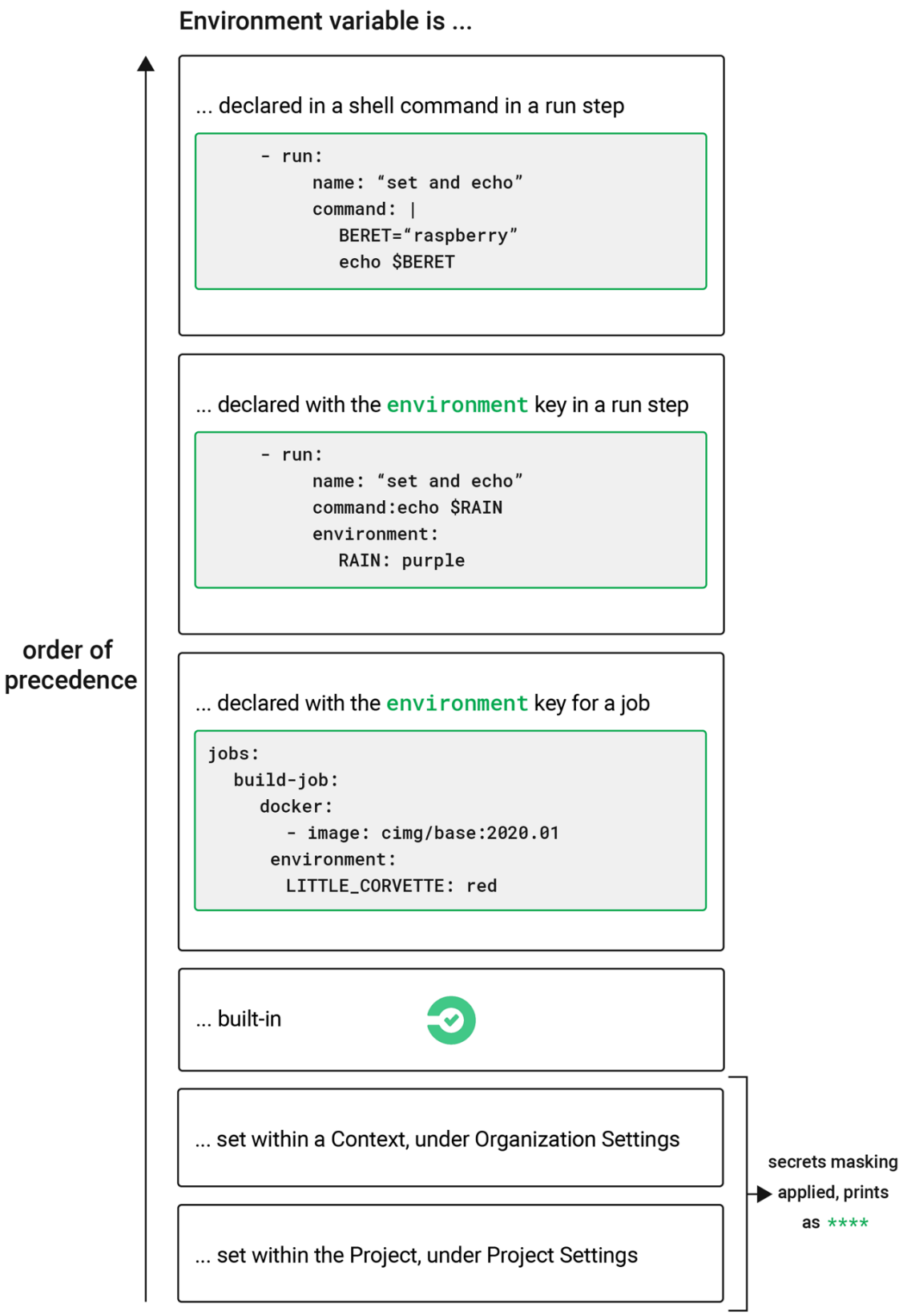
CircleCI uses Bash, which follows the POSIX naming convention for environment variables. Valid characters include letters (uppercase and lowercase), digits, and the underscore. The first character of each environment variable must be a letter.

Order of precedence

Environment variables are used according to a specific precedence order, as follows:

1. Environment variables declared inside a shell command in a `run` step, for example `FOO=bar make install`.
2. Environment variables declared with the `environment` key for a `run` step.
3. Environment variables set with the `environment` key for a job.
4. Special CircleCI environment variables defined in the CircleCI Built-in environment variables document.
5. Context environment variables (assuming the user has access to the context). See the Contexts documentation for more information.
6. Project-level environment variables set on the **Project Settings** page in the web app.

Environment variables declared inside a shell command `run step`, for example `FOO=bar make install`, will override environment variables declared with the `environment` and `contexts` keys. Environment variables added on the **Contexts** page in the web app will take precedence over variables added on the **Project Settings** page.



Example configuration of environment variables

Consider the example `.circleci/config.yml` below:

```
1  version: 2.1
2
3  jobs: # basic units of work in a run
```

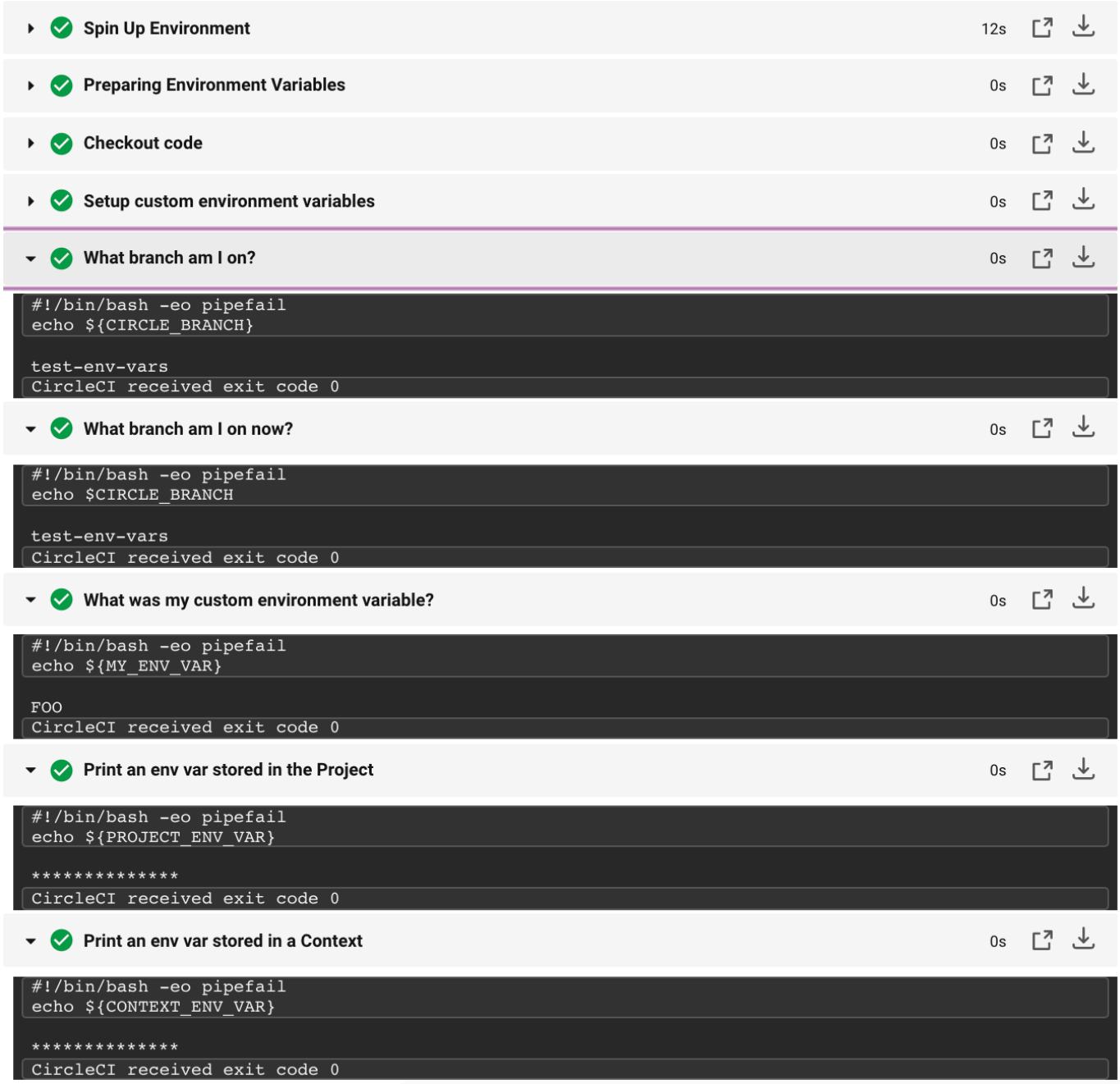
```
  build:
    docker: # use the Docker executor
      # CircleCI Node images available at: https://circleci.com/developer/images/image/cimg/node
      - image: cimg/node:18.11.0
    steps: # steps that comprise the `build` job
      - checkout # check out source code to working directory
      # Run a step to setup an environment variable
      # Redirect MY_ENV_VAR into $BASH_ENV
      - run:
          name: "Setup custom environment variables"
          command: echo 'export MY_ENV_VAR="FOO"' >> "$BASH_ENV"
      - run: # print the name of the branch we're on
          name: "What branch am I on?"
          command: echo ${CIRCLE_BRANCH}
      # Run another step, the same as above; note that you can
      # invoke environment variable without curly braces.
      - run:
          name: "What branch am I on now?"
          command: echo $CIRCLE_BRANCH
      - run:
          name: "What was my custom environment variable?"
          command: echo ${MY_ENV_VAR}
      - run:
          name: "Print an env var stored in the Project"
          command: echo ${PROJECT_ENV_VAR}
      - run:
          name: "Print an env var stored in a Context"
          command: echo ${CONTEXT_ENV_VAR}

workflows: # a single workflow with a single job called build
  build:
    jobs:
      - build:
          context: Testing-Env-Vars
```

The above `.circleci/config.yml` demonstrates the following:

- Setting custom environment variables
- Reading a built-in environment variable that CircleCI provides (`CIRCLE_BRANCH`)
- How variables are used (or interpolated) in your `.circleci/config.yml`
- Secrets masking, applied to environment variable set in the project or within a conte

When the above configuration runs, the output looks like the below image. Notice the environment variables stored in the project is masked, and displays as `****`:



Notice there are two similar steps in the above image and configuration - “What branch am I on?” These steps illustrate two different methods to read environment variables.

In the example configuration above, two syntaxes are used: `${VAR}` and `$VAR`. Both syntaxes are supported. You can read more about shell parameter expansion in the Bash documentation ↗ .

Parameters and bash environment

In general, CircleCI does not support interpolating environment variables in the configuration. Values used are treated as literals. This can cause issues when defining `working_directory`, modifying `PATH`, and sharing variables across multiple `run` steps.

In the example below, `$ORGNAME` and `$REPONAME` will not be interpolated.

```
1  working_directory: /go/src/github.com/$ORGNAME/$REPONAME
```

> An exception to this rule is using project environment variables to pull private images .

You can reuse pieces of configuration across your `.circleci/config.yml` file. By using the `parameters` declaration, you can pass values into reusable `commands`, `jobs`, and `executors`:

```
version: 2.1 # version 2.1 is required for reusing configuration

jobs:
  build:
    parameters:
      org_name:
        type: string
        default: my_org
      repo_name:
        type: string
        default: my_repo
    docker:
      - image: cimg/go:1.17.3
    steps:
      - run: echo "project directory is go/src/github.com/<<
parameters.org_name >>/<< parameters.repo_name >>"

workflows:
  my_workflow:
    jobs:
      - build:
          org_name: my_organization
          repo_name: project1

      - build:
          org_name: my_organization
          repo_name: project2
```

For more information, read the documentation on using the parameters declaration.

Another possible method to interpolate values into your configuration is to use a `run` step to export environment variables to `BASH_ENV`, as shown below.

> The `$BASH_ENV` workaround only works with `bash`, and has not been confirmed to work with other shells.

```
steps:
  - run:
      name: Setup Environment Variables
      command: |
        echo 'export PATH="$GOPATH"/bin:"$PATH"' >> "$BASH_ENV"
        echo 'export GIT_SHA1="$CIRCLE_SHA1"' >> "$BASH_ENV"
```

In every step, CircleCI uses `bash` to source `BASH_ENV`. This means that `BASH_ENV` is automatically loaded and run, allowing you to use interpolation and share environment variables across `run` steps.

Environment variable substitution

The CircleCI CLI offers a wrapper around the `envsubst` tool, available both locally as well as in all jobs running on CircleCI. `envsubst` is a command-line utility used to replace environment variables in text strings.

CLI command:

```
circleci env subst
```

Usage

The `circleci env subst` command can accept text input from `stdin` or as an argument.

Within your repository create a file such as `template.json`, with value replaced by environment variable strings

```
{
  "foo": "$FOO",
  "provider": "${PROVIDER}"
}
```

`envsubst` can convert all types of environment variable strings, including those encased in curly braces (`{}`).

The config example below shows the corresponding environment variables as if they were defined directly within a step in the config. However, we strongly recommend creating the environment variables in the CircleCI app, either in **Project Settings** or as a **context**.

```
version: 2.1
jobs:
  process-template:
    docker:
      - image: cimg/base:current
    steps:
```

```
      - checkout
      - run:
          name: Process template file
          environment:
            # Environment variables would typically be served via a
context
            FOO: bar
            PROVIDER: circleci
          command: |
            circleci env subst < template.json > deploy.json
            cat deploy.json
workflows:
  env-subst-workflow:
    jobs:
      - process-template
```

In this example, the `<` symbol is used to redirect the contents of the `template.json` file as *input* to the `env subst` command, while the `>` symbol is used to redirect the output of the env subst command to the `deploy.json`.

You could alternatively pass input to the `circleci env subst` command as an argument: `circleci env subst "hello \$WORLD"`

Output:

```
{
  "foo": "bar",
  "provider": "circleci"
}
```

For instructions on installing the CircleCI CLI locally, read the **Installing the CircleCI local CLI** guide.

Alpine Linux

An image that has been based on **Alpine Linux** (like **Docker**), uses the `ash` shell.

To use environment variables with `bash`, add the `shell` and `environment` keys to your job.

```
version: 2.1

jobs:
  build:
    shell: /bin/sh -leo pipefail
    environment:
      BASH_ENV: /etc/profile
```

Notes on security

Do not add secrets or keys inside the `.circleci/config.yml` file. The full text of `.circleci/config.yml` is visible to developers with access to your project on CircleCI. Store secrets or keys in **project** or **context** settings in the CircleCI web app. For more information, see the **Encryption** section of the security page.

Running scripts within configuration may expose secret environment variables. See the **Using shell scripts** page for best practices for secure scripts.

Contexts

You can further restrict access to environment variables using **contexts**. Contexts are set from the **Organization Settings** in the CircleCI web app.

See also

- Security recommendations
- Set an environment variable
- Inject variables using the CircleCI API

Suggest an edit to this page

Make a contribution

Learn how to contribute

Still need help?

Ask the CircleCI community

Join the research community

Visit our Support site

circleci Docs

Terms of Use Privacy Policy Cookie Policy Security