
▼ ON THIS PAGE

Base directory

Commit status notifications

Recommendations for specific setups

Build a single site from a subdirectory or monorepo

Build multiple sites from a monorepo

Declare all dependencies within a subdirectory

Declare dependencies at the root level and within a subdirectory

Build from a subdirectory or monorepo

Netlify allows you flexibility in how you organize and build a site or application. Although some sites are built directly from the root of a repository, others have a slightly more complex setup. You can build a site from a subdirectory of a repository, or you can build from a monorepo — a repository that contains multiple sites or apps, each in its own subdirectory.

To help you manage these more complex build configurations, Netlify offers options such as setting a base directory, controlling the volume of commit status notifications, and configuring a custom [ignore command](#).

Base directory

The base directory setting prompts our buildbots to change to the specified directory to detect dependencies and perform caching during the build process. It's useful for building from a monorepo or subdirectory.

You can set the base directory in the following ways, though the [recommended method](#) may differ depending on your project setup:

in the UI when you create a **New site from Git**. For an existing site, you can update the setting at **Site settings > Build & deploy > Continuous Deployment > Build settings**.

in a [Netlify configuration file](#) under `[build]` settings. If you're using a `netlify.toml` file to set a base directory for a monorepo, the file must be at the root of the repository. You can include an additional `netlify.toml` in the base directory for all other site settings aside from `base`.

using [Netlify CLI](#) when setting up continuous deployment for a site. Change to the subdirectory that you'd like to set as the base, then run the `netlify init` command. The current working directory is then specified as the base.

If not explicitly set, the base directory defaults to the root of the repository. A base directory specified in a root-level `netlify.toml` overrides the UI setting.



Base directory versus `cd`

Configuring a base directory instead of prepending something like `cd SUBDIRECTORY_PATH &&` to your build command ensures a

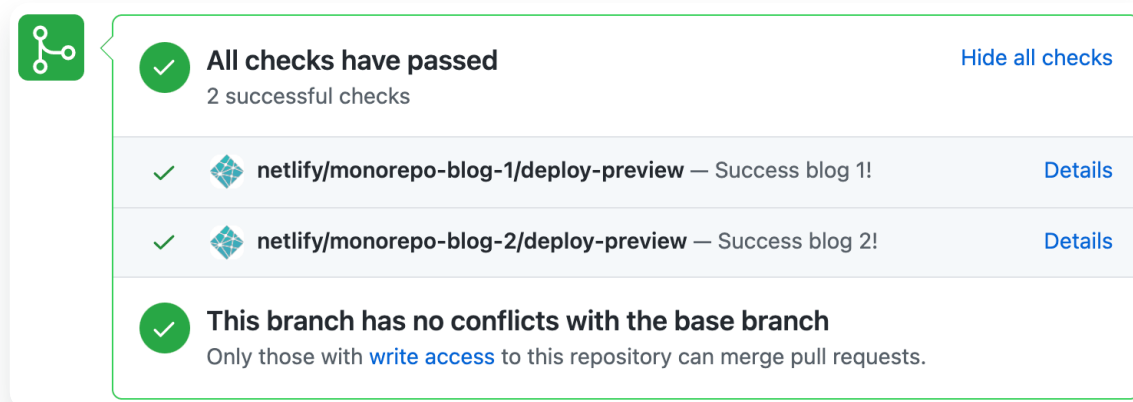
build appropriately, and they won't automatically ignore everything outside that subdirectory when triggering a build. This can result in slower builds and triggering unwanted builds from changes elsewhere in your project repository.



Commit status notifications





You can manage the amount of commit status and commit check notifications for projects where one repository builds multiple applications. These types of [deploy notifications](#) are available on commit lists and pull/merge requests for Netlify sites connected to GitHub or GitLab.


For your Netlify team, go to **Team settings > Sites > Notifications > Commit status notifications** and select one of the following options.

Multiple notifications per repo: enables commit statuses and commit checks for multiple sites linked to a repository.



  **All checks have passed** [Hide all checks](#)
2 successful checks

	 netlify/monorepo-blog-1/deploy-preview — Success blog 1! Details
	 netlify/monorepo-blog-2/deploy-preview — Success blog 2! Details

 **This branch has no conflicts with the base branch**
Only those with [write access](#) to this repository can merge pull requests.

One notification per repo: limits commit checks and commit statuses in a pull/merge request or commit list to one linked site only, regardless of the number of sites linked to a repository.

and tools. For each setup below, note that you can customize the default [ignore builds](#) behavior to determine whether or not your site builds as changes are made.

Build a single site from a subdirectory or monorepo

You may need to build a self-contained site from within a larger repository. For example, consider a Python library that has documentation in a fully self-contained subdirectory that includes any necessary configuration files. In this example, only the Python library's documentation site will be hosted on Netlify.

Similar to the setup described above, you may have a JavaScript monorepo that contains a subdirectory with an app to deploy with Netlify and another subdirectory with a component library that's maintained in the same repo.

To accomplish either configuration, set the subdirectory containing the site you want to deploy as your [base directory](#). You can do this in a `netlify.toml` file in the root of your repository, using the UI, or using Netlify CLI. Note that you can add a `netlify.toml` within the base directory as well to configure other settings aside from `base`.

```
repository-root/  
├─ netlify.toml  
├─ package.json  
├─ app/  
│   ├─ netlify.toml  
│   ├─ package.json  
│   └─ source-file
```



With the default [ignore builds](#) behavior, changes in the example `app` base directory trigger a full build. Meanwhile, changes to the repository outside of the `app` base directory cause the build to cancel early, as soon as possible after our buildbot clones the repo, checks the base directory settings, and runs a diff between relevant commits.

Build multiple sites from a monorepo

You may want to build and deploy more than one site from a single monorepo. For example, consider a JavaScript monorepo that contains two component libraries and three sites to be hosted on Netlify.

In this scenario, you must link each site to Netlify separately and use the Netlify UI or CLI to set the [base directory](#) for each site to point to its subdirectory within the monorepo. You can add a separate `netlify.toml` file under the base directory of each site you're deploying to control custom configurations. With the default [ignore builds](#) behavior, changes in a designated base directory trigger a build for that site only.

```
repository-root/  
├─ package.json  
├─ site-1/  
│   ├─ netlify.toml  
│   ├─ package.json  
│   └─ source-1  
├─ site-2/  
│   ├─ netlify.toml  
│   ├─ package.json  
│   └─ source-2
```



```
|   └─ source-3
├─ component-library-1/
|   └─ package.json
|   └─ code-1
└─ component-library-2/
    └─ package.json
    └─ code-2
```

Declare all dependencies within a subdirectory

In the examples above for building a [single site](#) or [multiple sites](#), all necessary dependencies are declared in configuration files within each subdirectory. The root-level `package.json` is used to define a Yarn or npm workspace.

Although it's possible to declare dependencies at the root, we recommend a setup where you define all dependencies at a more specific, subdirectory level. For example, if you have a monorepo with a site that uses React and a component library that also uses React, you should list `react` as a dependency in `package.json` files within each subdirectory.

If you have this self-contained setup, you can set each subdirectory as a base directory to build a separate Netlify site. This leaves dependency management up to the tools you're using, such as Yarn, npm, or Lerna. With a Yarn workspace, for example, Yarn can hoist dependencies dynamically, moving them to the root of the repository whenever it makes sense to do so.

Declare dependencies at the root level and within a subdirectory

Yarn workspaces

If Netlify detects that a linked site is either a Yarn workspace or has a root package that contains multiple Yarn workspaces, our build system caches all `node_modules` directories within the repository. This includes the root of the project, the actual project, and any potential sibling workspaces. Because Yarn workspaces are *context-aware*, they work well with a base directory setting and dependencies declared at any level of a project. Yarn is able to detect that a base directory is part of a workspace and therefore installs dependencies from all locations.

As stated in our [manage Yarn](#) docs, by default Netlify uses the presence of a `yarn.lock` file in the base directory to detect the use of Yarn. For monorepo setups with subdirectories and workspaces, a `yarn.lock` file may not be present in the base directory. To enable Yarn usage in these situations, set the `NETLIFY_USE_YARN` [environment variable](#) to `true`.

npm workspaces

Unlike Yarn workspaces, npm workspaces aren't yet context-aware, which means that npm doesn't detect that a subdirectory is part of a larger workspace. When a base directory is set, our build system changes into the specified subdirectory and runs `npm install`, but it will use the `package.json` in only that subdirectory as a basis for the entire installation and build process.

This can be a problem if a package in your subdirectory has dependencies declared in the root-level `package.json` or if it depends on sibling packages within the monorepo. The `npm install` process will be unable to resolve those dependencies. A potential workaround is to prepend



netlify

docs

Menu



Did you find this doc useful?

Your feedback helps us improve our docs.



[Netlify](#) [Careers](#) [Blog](#) [Terms](#) [Privacy](#)

Netlify, Inc. © 2021