Deploying with Git

Last updated 09 March 2020

Table of Contents

Heroku manages app deployments with Git (https://git-scm.com/), the popular version control system. You definitely don't need to be a Git expert to deploy code to Heroku, but it's helpful to learn the basics.

Prerequisites: Install Git and the Heroku CLI

- Git installation instructions (https://git-scm.com/book/en/v2/Getting-Started-Installing-Git)
- Heroku CLI installation instructions (https://devcenter.heroku.com/articles/heroku-cli#download-and-install)

Tracking your app in Git

If your app is already tracked in a Git repository, proceed to **Creating a Heroku remote**.

Before you can deploy your app to Heroku, you need to initialize a local Git repository and commit your application code to it.

The following example demonstrates initializing a Git repository for an app that lives in the `myapp` directory:

```
$ cd myapp
$ git init
Initialized empty Git repository in .git/
$ git add .
$ git commit -m "My first commit"
Created initial commit 5df2d09: My first commit
 44 files changed, 8393 insertions(+), 0 deletions(-)
 create mode 100644 README
 create mode 100644 Procfile
 create mode 100644 app/controllers/source_file
...
```


Be sure to initialize the Git repository in your app's root directory. If your app is in a subdirectory of your repository, it won't run when it is pushed to Heroku.

Your app's code is now tracked in a local Git repository. It has not yet been pushed to any remote servers.

Creating a Heroku remote

Git remotes (http://git-scm.com/book/en/Git-Basics-Working-with-Remotes) are versions of your repository that live on other servers. You deploy your app by pushing its code to a special Heroku-hosted remote that's associated with your app.

For a new Heroku app

The `heroku create` CLI command creates a new empty application on Heroku, along with an associated empty Git repository. If you run this command from your app's root directory, the empty Heroku Git repository is automatically set as a remote for your local repository.

```
$ heroku create
Creating app... done, ⬢ thawing-inlet-61413
https://thawing-inlet-61413.herokuapp.com/ | https://git.heroku.com/thawing-inlet-61413.git
```

You can use the `git remote` command to confirm that a remote named `heroku` has been set for your app:

```
$ git remote -v
heroku  https://git.heroku.com/thawing-inlet-61413.git (fetch)
heroku  https://git.heroku.com/thawing-inlet-61413.git (push)
```

For an existing Heroku app

If you have already created your Heroku app, you can easily add a remote to your local repository with the `heroku git:remote` command. All you need is your Heroku app's name:

```
$ heroku git:remote -a thawing-inlet-61413
set git remote heroku to https://git.heroku.com/thawing-inlet-61413.git
```

Renaming remotes

By default, the Heroku CLI names all of the Heroku remotes it creates for your app `heroku`. You can rename your remotes with the `git remote rename` command:

```
$ git remote rename heroku heroku-staging
```

Renaming your Heroku remote can be handy if you have multiple Heroku apps that use the same codebase (for example, the staging and production versions of an app). In this case, each Heroku app has its own remote in your local repository.

The remainder of this article assumes your app has a single Heroku remote that is named `heroku`.

Deploying code

To deploy your app to Heroku, you typically use the `git push` command to push the code from your local repository's `master` branch to your `heroku` remote, like so:

```
$ git push heroku master
Initializing repository, done.
updating 'refs/heads/master'
...
```

Use this same command whenever you want to deploy the latest committed version of your code to Heroku.

Note that Heroku only deploys code that you push to the `master` branch of the `heroku` remote. Pushing code to another branch of the remote has no effect.

Deploying from a branch besides master

If you want to deploy code to Heroku from a non-`master` branch of your local repository (for example, `testbranch`), use the following syntax to ensure it is pushed to the remote's `master` branch:

```
$ git push heroku testbranch:master
```

Applications that rely on Git submodules are supported, in addition to many other **dependency resolution strategies (https://devcenter.heroku.com/articles/git-submodules)**.

git lfs (https://git-lfs.github.com/) is not supported, and using it might cause pushes to fail.

Detaching from the build process

After you initiate a Heroku deploy with `git push`, you can detach from the resulting build process by pressing Ctrl + C. This does not cancel the build or the deploy. The build will continue in the background and will create a new release (https://devcenter.heroku.com/articles/releases) as soon as it completes.

Resolving simultaneous deploys

It's possible to initiate a deploy before a *previous* deploy of the same app completes. For example, two collaborators on an app might push different commits to the `heroku` remote at roughly the same time.

If this occurs, the different versions of your app will be deployed to Heroku **in the order in which their respective builds complete. Note that this can differ from the order in which the pushes occurred.**

For example, consider two builds, A and B. If Build B starts *after* Build A but finishes *before* it, Heroku will deploy Build B *first*. Then, when Build A eventually completes, Heroku will deploy it, replacing Build B.

HTTP Git authentication

By default, Heroku uses HTTP as its Git transport. The Heroku CLI will automatically place credentials in the `.netrc` file on `heroku login`. The Git client uses cURL when interacting with HTTP remotes, and cURL will use the credentials from the `.netrc` file. See the **Authentication section** and the **CLI authentication article (https://devcenter.heroku.com/articles/authentication)** for details.

The Heroku HTTP Git endpoint only accepts API-key based HTTP Basic (http://en.wikipedia.org/wiki/Basic_access_authentication) authentication. A username is not required and any value passed for username is ignored.

You cannot authenticate with the Heroku HTTP Git endpoint using your Heroku username (email) and password. Use an API key as described in this section

If, for any reason, you authenticate to the Git service with incorrect credentials, you'll get this error:

```
remote: ! WARNING:
remote: ! Do not authenticate with username and password using git.
remote: ! Run `heroku login` to update your credentials, then retry the git command.
remote: ! See documentation for details: https://devcenter.heroku.com/articles/git#http-git
```

When you do `heroku login`, the CLI will write an entry for `git.heroku.com` into your `.netrc` file (or its Windows equivalent). Since the Git client uses cURL when interacting with HTTP Git remotes, correct authentication will now happen transparently.

If you're using other Git clients, such as EGit or Tower, configure them to use an empty string for username (or any string you like – it's ignored) and your account API key for password. The API key is available in the CLI (https://devcenter.heroku.com/articles/authentication#retrieving-the-api-token) and in Dashboard (https://dashboard.heroku.com/account).

SSH Git transport

The default Git transport configured by the Heroku CLI is HTTP, but SSH transport is also supported. SSH and HTTP transport can be used interchangeably by the same user and by multiple users collaborating on the same app. To have the Heroku CLI configure SSH transport, you can pass a `--ssh-git` flag to the `heroku create`, `heroku git:remote` and `heroku git:clone` commands.

```
$ heroku create --ssh-git
```

To use SSH Git transport, you have to register your SSH key with Heroku. See the Managing SSH Keys article (https://devcenter.heroku.com/articles/keys) for details.

If you want to always use SSH Git with Heroku on a particular machine, you can add the following global config:

```
$ git config --global url.ssh://git@heroku.com/.insteadOf https://git.heroku.com/
```

HTTP URLs will still be written to `.git` folders but Git will rewrite, on the fly, all Heroku HTTP Git URLs to use SSH.

To remove this rewrite setting, run:

```
$ git config --global --remove-section url.ssh://git@heroku.com/
```

The SSH Git transport isn't supported for SSO users; SSO users must use the HTTP Git transport.

Multiple remotes and environments

The same techniques used to deploy to production can be used to deploy a development branch of your application to a staging application on Heroku, as described in Managing Multiple Environments for an App (https://devcenter.heroku.com/articles/multiple-environments).

Build cache

Buildpacks (https://devcenter.heroku.com/articles/buildpacks) can optionally cache content (such as app dependencies) to greatly speed up future builds.

If you suspect that a build problem is being caused by an error in this cache, you can use the `heroku-repo` (https://github.com/heroku/heroku-repo) plugin to clear it.

Repository size

Although there is not a hard limit on your repository size, very large repositories (over 600 MB) are not recommended; they may cause timeouts and slow pushes overall. Running `heroku apps:info` will show you your repository size.

Common causes of large repositories are binary files checked into the repository (Git is notoriously bad at handling binaries) or constantly-changing development logs. Removing files committed by accident can be done with git filter-branch (http://git-scm.com/book/en/Git-Internals-Maintenance-and-Data-Recovery#_removing_objects), though after running it you will have to push with the `--force` option, which is something that requires coordination among your team.

Other limits

To protect the Git service, Heroku imposes certain limits on Git repository use and content size.

Users are limited to a rolling window of 75 Git requests per hour, per user, per app. Once this limit is reached, Git requests are denied until request levels drop below the limit for a few minutes, with the error message:

```
!  Too many requests for this Git repo. Please try again later.
```

If you reach this limit, ensure there are not automated processes or scripts polling the Git repository.

In addition, the uncompressed size of a checkout of `HEAD` from the repository, combined with the size of restored submodules, cannot exceed 1 GB.

Using subversion or other revision control systems

What if you're already using Subversion or another revision control system to track your source code? Although we believe that Git is one of the best choices available for revision control, you don't need to stop using your current revision control system. Git can be purely a deployment mechanism, existing side-by-side with your other tool.

> You can learn much more about `.gitignore` in **our article on the topic (https://devcenter.heroku.com/articles/gitignore)**.

For example, if you are using Subversion, initialize your Git repository as described above. Then, add a `.gitignore` file to tell Git to ignore your Subversion directories.

```
$ git init
$ echo .svn > .gitignore
$ git add .
$ git commit -m "using git for heroku deployment"
```

Now tell Subversion to ignore Git:

```
$ svn propset svn:ignore .git .
property 'svn:ignore' set on '.'
$ svn commit -m "ignoring git folder (git is used for heroku deployment)"
```

> The `-f` (force flag) is recommended in order to avoid conflicts with other developers' pushes. Since you are not using Git for your revision control, but as a transport only, using the force flag is a reasonable practice.

Each time you wish to deploy to Heroku:

```
$ git add -A
$ git commit -m "commit for deploy to heroku"
...

$ git push -f heroku
```

Additional resources

- Git on Rails (http://railscasts.com/episodes/96) shows common conventions for using Git to track Rails apps.
- Git cheat sheets for web (http://cheat.errtheblog.com/s/git) and print (http://res.cloudinary.com/hy4kyit2a/image/upload/SF_git_cheatsheet.pdf) consumption.
- Git - SVN Crash Course (http://git.or.cz/course/svn.html)
- The Pro Git book (https://git-scm.com/book) is a great resource that covers all of Git.