
maintained no! (as of 2019)

DPLOY

DPLOY is an FTP/SFTP deployment tool built in node.js

Uploads the latest changes by comparing the version on your server with your git repository.

Install

Install DPLOY and it's dependencies globally by running:

```
1 npm install dploy -g
```

Help

```
1 dploy --help
```

Version

```
1 dploy --version
```

Commands

dploy

Will deploy the first environment that you have on your `dploy.yaml`

dploy install

Will install the `dploy.yaml` file and set up a `post-commit` script on your `.git/hooks` folder so you can *DPLOY* from your commit message as well.

dp`loy` ...rest

Anything else after the `dploy` command will be used as an environment, like this:

```
1 dploy dev stage production
```

In this case *DPLOY* will expect to find **dev**, **stage** and **production** configs on your `dploy.yaml` file.

Basic example

If you only have one server, just name whatever you want and run `dploy`.

```
1 server_name:
2   host: "ftp.myserver.com"
3   user: "user"
4   pass: "password"
5   path:
6     local: "deploy/"
7     remote: "public_html/"
```

Deploying on the command line:

```
1 dploy
```

You can also set the environment that you want to upload:

```
1 dploy server_name
```

Attributes of the `dploy.yaml`

scheme

- Type: `String`
- Default: `ftp`

DPLOY has two available schemes: **ftp** and **sftp**. You must provide this information, because we don't like to play guessing games.

host

- Type: `String`

-
- Default: `none`

port

- Type: `Number`
- Default: 21 when ftp and 22 when sftp

The port that your hosting server is using. Note that the default value is different depending on **scheme** that you are using.

user

- Type: `String`
- Default: `none`

pass

- Type: `String`
- Default: `none`

If you don't set a password and if you are using SFTP, DPLOY will try look for the **privateKey** and **publicKey**.

But if we can't find any of those options, you will be prompted to type the password manually.

privateKey

- Type: `path`
- Default: `none`
- Scheme: `sftp`

When using SFTP, you can set the path of your private key instead of the password. The default locations are usually:

```
1 privateKey: ~/.ssh/id_rsa
2 privateKey: ~/.ssh/id_dsa
```

passphrase

- Type: `String`
- Default: `none`
- Scheme: `sftp`

For an encrypted private key, this is the passphrase used to decrypt it.

publicKey

- Type: `path`
- Default: `none`
- Scheme: `sftp`

It works in the same way of the **privateKey**. The default locations are usually:

```
1 publicKey: ~/.ssh/id_rsa.pub
2 publicKey: ~/.ssh/id_dsa.pub
```

secure

- Type: `mixed`
- Default: `false`
- Scheme: `ftp`

Set this parameter only if you are using FTPS. Set to `true` for both control and data connection encryption, `control` for control connection encryption only, or `implicit` for implicitly encrypted control connection.

secureOptions

- Type: `object`
- Default: `none`
- Scheme: `ftp`

Additional options to be passed together with the `secure` parameter.

revision

- Type: `String`
- Default: `.rev`

To check the different between your local files and what's on the server, we have to create a temporary file with the reference of the last commit you've uploaded. This parameter defines the name of this file.

slots

- Type: `Number`
- Default: `1`

To make the upload faster, you can create multiple connections to your server.

check

- Type: `Boolean`
- Default: `false`

If you set this parameter to `true`, you will be prompted to confirm the list of files before the actual action.

branch

- Type: `String` or `Array`
- Default: `none`

You can set a list of branches that are allowed to deploy to your server. This will also help you to avoid accidental uploads to different servers.

Note that you can also set a string (a single branch), rather than a list.

path.local

- Type: `String`
- Default: `none`

The local folder that you want to upload to the server. If you don't set anything, the entire folder of your project will be uploaded.

path.remote

- Type: `String`
- Default: `none`

The remote folder where your files will be uploaded. If you don't set anything, your files will be uploaded to the root of your server. We **highly recommend** that you set this!

exclude

- Type: `Array`
- Default: `none`

Exclude files that are tracked by git, but that you don't want on your server. You can target individual files or use glob to target multiple files and file types.

- Individual files: `exclude`: `["deploy.yaml", "package.json", "path/to/file.js"]`.
- Using glob: `exclude`: `["*.yaml", "*.json", "path/**/*.js", "**/*.md"]`.

include

- Type: `Object`
- Default: `none`

The **include** parameter is similar to the **exclude**. But instead of an array, it expects an object.

The **key** of your object is what *DPLOY* is gonna search locally and the **value** of your object is the destination on the remote server (this path is relative to the **path.remote**!). Again you can also target individual files or multiple using glob on the key of your object.

```
1 include:
2   "videos/kitty.mp4": "videos/"
3   "videos/*.mp4": "another/folder/inside/remote/path/"
4   "*.json": "data/"
```

Ignore include flag

If you are using the **include** parameter on your `deploy.yaml`, you will note that those files will always be uploaded to the server, no matter if they were modified or not (because they aren't necessarily tracked by git).

In order to avoid re-uploading those files all the time, there's a tag called `--ignore-include` that you can set when calling *DPLOY*.

```
1 deploy stage --ignore-include
```

Or using a shortcut:

```
1 deploy stage -i
```

Catchup flag

If you already have your files on the server (from a previous manual upload or if you somehow deleted the revision file), setting this flag will upload only the revision file and nothing more. It can be used for multiple servers too.

```
1 deploy stage --catchup
```

Or using a shortcut:

```
1 deploy stage -c
```

Multiple environments

Most of the times we have to work on different environments (dev, stage, production...).

With *DPLOY* is really easy to make multiple deploys using a single command. All you need to do is create different configurations on your `deploy.yaml` file, like this:

```
1 dev:
2   host: "dev.myserver.com"
3   user: "dev_user"
4   pass: "dev_password"
5   path:
6     local: "deploy/"
7     remote: "public_html/"
8
9 stage:
10  host: "stage.myserver.com"
11  user: "stage_user"
12  pass: "stage_password"
13  path:
14    local: "deploy/"
15    remote: "public_html/"
16
17 production:
18  host: "myserver.com"
19  user: "production_user"
20  pass: "production_password"
21  path:
22    local: "deploy/"
23    remote: "public_html/"
```

Deploy to **stage** environment only:

```
1 dploy stage
```

Or if you want to upload to more than one environment:

```
1 dploy dev stage production
```

Including and excluding files

This example will upload your local `deploy` folder to your remote `public_html` folder and:

- Will **include** all `.mp4` files inside your `videos` folder to a remote folder named `funny` on your server.
- Will **include** all `json`, `yaml` and `xml` files at your cwd folder to a remote folder named `data`.
- Will **exclude** all `yaml`, `json` from your `deploy` folder.

-
- Will **exclude** all `js` files inside the folder `deploy/path`.
 - Will **exclude** all `md` files from your `deploy` folder.

```
1 server_name:
2   host: "ftp.myserver.com"
3   user: "user"
4   pass: "password"
5   path:
6     local: "deploy/"
7     remote: "public_html/"
8   exclude: ["deploy/*.yaml", "deploy/*.json", "deploy/path/**/*.js",
9     "deploy/**/*.md"]
10  include:
11    "videos/*.mp4": "funny/"
12    "*.json *.yaml *.xml": "data/"
```

Contribute

Feel free to contribute to DPLOY in any way. If you have any issues, questions or suggestions, just create it at the issues page.

If you want to create your own fork, follow the instructions bellow to build **DPLOY**:

build

You need to install the dependencies from npm first and then just use grunt to compile the CoffeeScript:

```
1 grunt
```

watch

You can watch the changes by running the watch task from grunt:

```
1 grunt watch
```

Mentions

DPLOY was inspired by another great tool written in Ruby, called dandelion from Scott Nelson.

License

The MIT License

Copyright (c) 2013 Lean Mean Fighting Machine, Inc. <http://lmfm.co.uk>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.