Architecture & Deployment

2025-2026 v0.1.0 on branch main Rev: 6d218297357081f922e8a7dd1d5cd8471c27fa79

Git Cheatsheet

If you want to learn more about Git, read the Pro Git book (online & free).

Table of contents

Best practices

- **Commit early and often, perfect later** (Seth Robertson)

  Git only takes full responsibility for your data when you commit. If you fail to commit and then do something poorly thought out, you can run into trouble. Additionally, having periodic checkpoints means that you can understand how you broke something.

- **Writing a good commit message** (GitKraken)

If by taking a quick look at previous commit messages, you can discern what each commit does and why the change was made, you're on the right track. But if your commit messages are confusing or disorganized, then you can help your future self and your team by improving your commit message practices with help from this article.

- **Conventional Commits**

  If you want to go further, look at *Conventional Commits*, a specification for adding human and machine readable meaning to commit messages.

- Choose a **branching workflow**:
  - A successful branching model (for large teams)
  - A successful branching model considered harmful
  - Branch-per-feature
  - Trunk-based development
- Enable Git Rerere

One-time configuration

Configure your identity

You must configure identity using your user name and e-mail address. This is important because every Git commit uses this information, and it's immutably baked into every commit you make. **You should obviously replace your `"John Doe"` and `john.doe@example.com` with your own information.**

```
$> git config --global user.name "John Doe"
$> git config --global user.email john.doe@example.com
```

Automagically exclude annoying `.DS_Store` files from your commits (macOS only)

You can create a global ignore file in your home directory to ignore them:

```
$> echo ".DS_Store" >> ~/.gitignore
```

Run the following command to configure Git to use this file. You only have to do it once on each machine:

```
$> git config --global core.excludesfile ~/.gitignore
```

Frequent operations

Create a new empty repository

```
$> cd /path/to/projects
$> mkdir my-new-project
$> cd my-new-project
$> git init
```

Put an existing project on GitHub

```
$> cd /path/to/projects/my-project
$> git init
```

If you **don't want to commit some files**, create a `.gitignore` file listing them each on one line, e.g.

```
*.log
node_modules
```

Commit the project's files:

```
$> git add --all
$> git commit -m "Initial commit"
```

Create your new repository on GitHub, copy the SSH clone URL (e.g. `git@github.com:MyUser/my-project.git`), and add it as a remote:

```
$> git remote add origin git@github.com:MyUser/my-project.git
```

Push your `main` branch and track it (with the `-u` option):

```
$> git push -u origin main
```

Push my latest changes to the GitHub repository

Commit and push your changes:

```
$> git add --all
$> git commit -m "My changes"
$> git push origin main
```

If GitHub rejects your push, you should **pull the latest changes** first.

Pull the latest changes from the GitHub repository

If you have uncommitted change (check with `git status`), stage and commit them:

```
$> git add --all
$> git commit -m "My changes"
```

Pull the changes:

```
$> git pull
```

If you've worked on the same files, there might be a **merge**. **If there is a merge conflict**, resolve it and complete the merge with `git commit`.

Add an SSH key to my GitHub account

See Adding a new SSH key to your GitHub account.

↑ Back to top