Reference Manual

Useful for our day to day using our team git work flow

Team:

Equipo-c23.87-webapp

Date of elaboration: 08/01/2025

Content

Commit message prefixes

Using consistent commit message prefixes helps maintain a clear and organized commit history. Here are some common prefixes based on the Conventional Commits specification.

- **feat**: A new feature
- **fix**: A bug fix
- **docs**: Documentation only changes
- **style**: Changes that do not affect the meaning of the code (white-space, formatting, missing semi-colons, etc.)
- **refactor**: A code change that neither fixes a bug nor adds a feature
- **perf**: A code change that improves performance
- **test**: Adding missing tests or correcting existing tests
- **build**: Changes that affect the build system or external dependencies (example scopes: gulp, broccoli, npm)
- **ci**: Changes to our CI configuration files and scripts (example scopes: Travis, Circle, BrowserStack, SauceLabs)
- **chore**: Other changes that don't modify src or test files
- **revert**: Reverts a previous commit

For instance, adding a *.gitignore* file, the chore prefix is appropriate since it involves maintaining the project without affecting the source code directly. Here's an example commit message for adding a *.gitignore* file:

1. Add the file

```
git add .gitignore
```

2. Commit the changes:

```
git commit -m "chore: Add .gitignore file"
```

3. Push the changes to the remote repository:

```
git push –u origin main
```

Useful References

- Conventional Commits (https://www.conventionalcommits.org/en/v1.0.0/)
- List of git commit message prefixes · GitHub (https://gist.github.com/iamskok/2cd92725f2d30b0f52c3)

Pushing to a private remote repository using HTTPS

1. **Generate a Personal Access Token**: If we're using GitHub, generate a personal access token from our GitHub account settings under "Developer settings" > "Personal access tokens".
2. **Clone the Repository**: Use the token to clone our private repository:

```
git clone
https://<username>:<token>@github.com/<username>/<repository>.git
```

! Replace <username>, <token>, and <repository> with our GitHub username, the generated token, and our repository name, respectively.

3. **Push Changes**: After making changes and committing them, push our changes:

```
git push –u origin main
```

Pushing to a private remote repository using SSH

1. **Generate SSH keys**: If we don´t have an SSH key pair, we can generate one using the following command in our terminal or git bash:

```
ssh-keygen –t rsa –b 4096 –C "our_email@example.com"
```

! Let´s break down it:

1. **ssh-keygen**: This is the command used to generate a new SSH key pair.
2. **-t rsa**: The -t option specifies the type of key to create. In this case, rsa indicates that an RSA key pair will be generated. RSA (Rivest-Shamir-Adleman) is a widely used encryption algorithm.
3. **-b 4096**: The -b option specifies the number of bits in the key. Here, 4096 bits are used, which provides a high level of security. The larger the number of bits, the more secure the key, but it also takes longer to generate and use.
4. **-C "our_email@example.com"**: The -C option adds a comment to the key. This is typically used to identify the key, and using our email address is a common practice. This comment is included in the public key file and can help you identify which key is which if we have multiple keys.

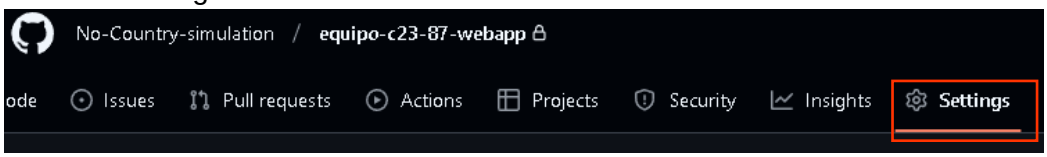
References

What is ssh-keygen & How to Use It to Generate a New SSH Key? (https://www.ssh.com/academy/ssh/keygen)

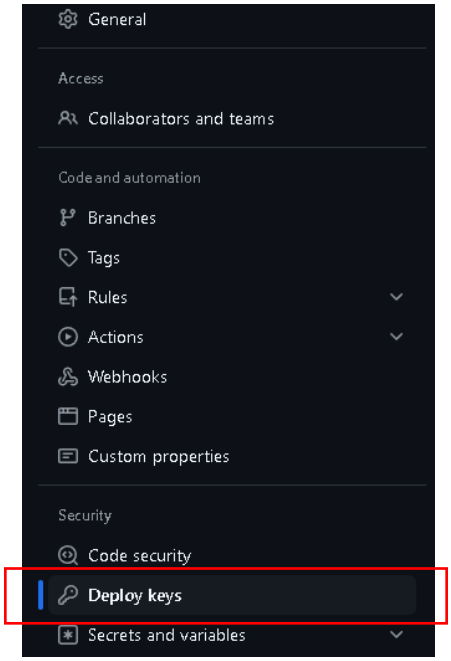
How to use the command ssh-keygen (with examples) (https://commandmasters.com/commands/ssh-keygen-common/)

2. **Add SSH key to GitHub**: Copy the contents of our public key (**c:\Users\<User>\.ssh\id_rsa.pub**) and add it to our GitHub account (or repo):

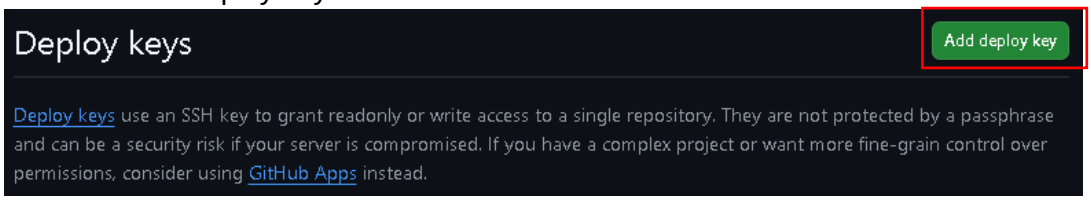
    1. Click on Settings:



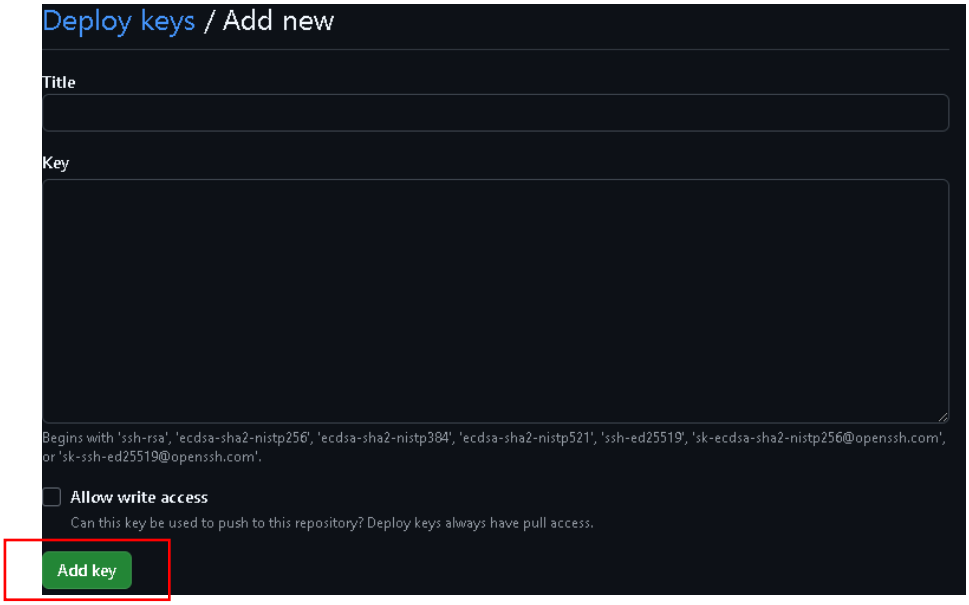
    2. Click on “Deploy keys” in the panel on the right of the “Settings” options:



    3. Click on “Add deploy key” button:



4. Add a title, paste the key of “id_rsa.pub” and click on “Add key” button:



And that’s it, we should now be able to push code from your local repo to the remote one.

❗Verify it if we running a `git push`