**Cloning with HTTPS URLs (recommended)**

The https:// clone URLs are available on all repositories, public and private. These URLs work everywhere--even if you are behind a firewall or proxy. In certain cases, if you'd rather use SSH, you might be able to use [SSH over the HTTPS port](https://help.github.com/articles/using-ssh-over-the-https-port).

When you git clone, git fetch, git pull, or git push to a remote repository using HTTPS URLs on the command line, you'll be asked for your GitHub username and password.

If you have enabled [two-factor authentication](https://help.github.com/articles/securing-your-account-with-two-factor-authentication-2fa), or if you are accessing an organization that uses [SAML single sign-on](https://help.github.com/articles/about-authentication-with-saml-single-sign-on), you must [provide a personal access token](https://help.github.com/articles/creating-a-personal-access-token-for-the-command-line) instead of entering your password for HTTPS Git.

Adding a remote

To add a new remote, use the git remote add command on the terminal, in the directory your repository is stored at.

The git remote add command takes two arguments:

* A remote name, for example, origin
* A remote URL, for example, https://github.com/user/repo.git

For example:

git remote add origin https://github.com/*user*/*repo*.git

# Set a new remote

git remote -v

# Verify new remote

origin https://github.com/*user*/*repo*.git (fetch)

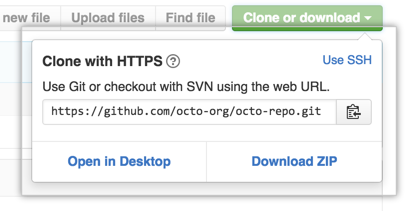
origin https://github.com/*user*/*repo*.git (push)

##Not sure which URL to use? Check out "[Which remote URL should I use?](https://help.github.com/articles/which-remote-url-should-i-use)"

Which remote URL should I use?

There are several ways to clone repositories available on GitHub.

When you view a repository while signed in to your account, the URLs you can use to clone the project onto your computer are available below the repository details:



##For information on setting or changing your remote URL, see "[Changing a remote's URL](https://help.github.com/articles/changing-a-remote-s-url)."

Changing a remote's URL

The git remote set-url command changes an existing remote repository URL.

**Tip:** For information on the difference between HTTPS and SSH URLs, see "[Which remote URL should I use?](https://help.github.com/articles/which-remote-url-should-i-use)"

The git remote set-url command takes two arguments:

* An existing remote name. For example, origin or upstream are two common choices.
* A new URL for the remote. For example:
  + If you're updating to use HTTPS, your URL might look like:
  + https://github.com/*USERNAME*/*REPOSITORY*.git
  + If you're updating to use SSH, your URL might look like:
  + git@github.com:*USERNAME*/*REPOSITORY*.git

**Switching remote URLs from SSH to HTTPS**

1. Open Git Bash.
2. Change the current working directory to your local project.
3. List your existing remotes in order to get the name of the remote you want to change.
4. git remote -v
5. origin git@github.com:*USERNAME/REPOSITORY*.git (fetch)
6. origin git@github.com:*USERNAME/REPOSITORY*.git (push)
7. Change your remote's URL from SSH to HTTPS with the git remote set-url command.
8. git remote set-url origin https://github.com/*USERNAME*/*REPOSITORY*.git
9. Verify that the remote URL has changed.
10. git remote -v
11. # Verify new remote URL
12. origin https://github.com/*USERNAME/REPOSITORY*.git (fetch)
13. origin https://github.com/*USERNAME/REPOSITORY*.git (push)

The next time you git fetch, git pull, or git push to the remote repository, you'll be asked for your GitHub username and password.

* If you have [two-factor authentication](https://help.github.com/articles/securing-your-account-with-two-factor-authentication-2fa) enabled, you must [create a personal access token](https://help.github.com/articles/creating-a-personal-access-token-for-the-command-line) to use instead of your GitHub password.
* You can [use a credential helper](https://help.github.com/articles/caching-your-github-password-in-git) so Git will remember your GitHub username and password every time it talks to GitHub.

**Switching remote URLs from HTTPS to SSH**

1. Open Git Bash.
2. Change the current working directory to your local project.
3. List your existing remotes in order to get the name of the remote you want to change.
4. git remote -v
5. origin https://github.com/*USERNAME/REPOSITORY*.git (fetch)
6. origin https://github.com/*USERNAME/REPOSITORY*.git (push)
7. Change your remote's URL from HTTPS to SSH with the git remote set-url command.
8. git remote set-url origin git@github.com:*USERNAME*/*REPOSITORY*.git
9. Verify that the remote URL has changed.
10. git remote -v
11. # Verify new remote URL
12. origin git@github.com:*USERNAME/REPOSITORY*.git (fetch)
13. origin git@github.com:*USERNAME/REPOSITORY*.git (push)

**Troubleshooting**

You may encounter these errors when trying to changing a remote.

**No such remote '[name]'**

This error means that the remote you tried to change doesn't exist:

git remote set-url sofake https://github.com/octocat/Spoon-Knife

fatal: No such remote 'sofake'

Check that you've correctly typed the remote name.

Renaming a remote

Use the git remote rename command to rename an existing remote.

The git remote rename command takes two arguments:

* An existing remote name, for example, origin
* A new name for the remote, for example, destination

**Example**

These examples assume you're [cloning using HTTPS](https://help.github.com/articles/which-remote-url-should-i-use/#cloning-with-https-urls-recommended), which is recommended.

git remote -v

# View existing remotes

origin https://github.com/*OWNER*/*REPOSITORY*.git (fetch)

origin https://github.com/*OWNER*/*REPOSITORY*.git (push)

git remote rename origin destination

# Change remote name from 'origin' to 'destination'

git remote -v

# Verify remote's new name

destination https://github.com/*OWNER*/*REPOSITORY*.git (fetch)

destination https://github.com/*OWNER*/*REPOSITORY*.git (push)

**Troubleshooting**

You may encounter these errors when trying to rename a remote.

**Could not rename config section 'remote.[old name]' to 'remote.[new name]'**

This error means that the remote you tried the old remote name you typed doesn't exist.

You can check which remotes currently exist with the git remote -v command:

git remote -v

# View existing remotes

origin https://github.com/*OWNER*/*REPOSITORY*.git (fetch)

origin https://github.com/*OWNER*/*REPOSITORY*.git (push)

**Remote [new name] already exists.**

This error means that the remote name you want to use already exists. To solve this, either use a different remote name, or rename the original remote.

Removing a remote

Use the git remote rm command to remove a remote URL from your repository.

The git remote rm command takes one argument:

* A remote name, for example, destination

**Example**

These examples assume you're [cloning using HTTPS](https://help.github.com/articles/which-remote-url-should-i-use/#cloning-with-https-urls-recommended), which is recommended.

git remote -v

# View current remotes

origin https://github.com/*OWNER/REPOSITORY*.git (fetch)

origin https://github.com/*OWNER/REPOSITORY*.git (push)

destination https://github.com/*FORKER/REPOSITORY*.git (fetch)

destination https://github.com/*FORKER/REPOSITORY*.git (push)

git remote rm destination

# Remove remote

git remote -v

# Verify it's gone

origin https://github.com/*OWNER/REPOSITORY*.git (fetch)

origin https://github.com/*OWNER/REPOSITORY*.git (push)

**Note**: git remote rm does not delete the remote repository from the server. It simply removes the remote and its references from your local repository.

**Troubleshooting**

You may encounter these errors when trying to remove a remote.

**Could not remove config section 'remote.[name]'**

This error means that the remote you tried to delete doesn't exist:

git remote rm sofake

error: Could not remove config section 'remote.sofake'

Check that you've correctly typed the remote name.

Changing a commit message

If a commit message contains unclear, incorrect, or sensitive information, you can amend it locally and push a new commit with a new message to GitHub. You can also amend a commit to add a co-author.

**Rewriting the most recent commit message**

You can change the most recent commit message using the git commit --amend command.

In Git, the text of the commit message is part of the commit. Changing the commit message will change the commit ID--i.e., the SHA1 checksum that names the commit. Effectively, you are creating a new commit that replaces the old one.

**Commit has not been pushed online**

If the commit only exists in your local repository and has not been pushed to GitHub, you can amend the commit message with the git commit --amend command.

1. On the command line, navigate to the repository that contains the commit you want to amend.
2. Type git commit --amend and press **Enter**.
3. In your text editor, edit the commit message and save the commit. You can also add a co-author by adding a trailer to the commit. For more information, see "[Creating a commit with multiple authors](https://help.github.com/articles/creating-a-commit-with-multiple-authors)."

The new commit and message will appear on GitHub the next time you push.

You can change the default text editor for Git by changing the core.editor setting. For more information, see "[Basic Client Configuration](https://git-scm.com/book/en/Customizing-Git-Git-Configuration#_basic_client_configuration)" in the Git manual.

**Amending older or multiple commit messages**

If you have already pushed the commit to GitHub, you will have to force push a commit with an amended message.

We strongly discourage force pushing, since this changes the history of your repository. If you force push, people who have already cloned your repository will have to manually fix their local history. For more information, see "[Recovering from upstream rebase](https://git-scm.com/docs/git-rebase#_recovering_from_upstream_rebase)" in the Git manual.

**Amending the message of the most recently pushed commit**

1. Follow the [steps above](https://help.github.com/articles/changing-a-commit-message#commit-has-not-been-pushed-online) to amend the commit message.
2. Use the push --force command to force push over the old commit.
3. git push --force *example-branch*

**Amending the message of older or multiple commit messages**

If you need to amend the message for multiple commits or an older commit, you can use interactive rebase, then force push to change the commit history.

1. On the command line, navigate to the repository that contains the commit you want to amend.
2. Use the git rebase -i HEAD~n command to display a list of the last n commits in your default text editor.
3. git rebase -i HEAD~3 # Displays a list of the last 3 commits on the current branch

The list will look similar to the following:

pick e499d89 Delete CNAME

pick 0c39034 Better README

pick f7fde4a Change the commit message but push the same commit.

# Rebase 9fdb3bd..f7fde4a onto 9fdb3bd

#

# Commands:

# p, pick = use commit

# r, reword = use commit, but edit the commit message

# e, edit = use commit, but stop for amending

# s, squash = use commit, but meld into previous commit

# f, fixup = like "squash", but discard this commit's log message

# x, exec = run command (the rest of the line) using shell

#

# These lines can be re-ordered; they are executed from top to bottom.

#

# If you remove a line here THAT COMMIT WILL BE LOST.

#

# However, if you remove everything, the rebase will be aborted.

#

# Note that empty commits are commented out

1. Replace pick with reword before each commit message you want to change.
2. pick e499d89 Delete CNAME
3. reword 0c39034 Better README
4. reword f7fde4a Change the commit message but push the same commit.
5. Save and close the commit list file.
6. In each resulting commit file, type the new commit message, save the file, and close it.
7. Force-push the amended commits.
8. git push --force