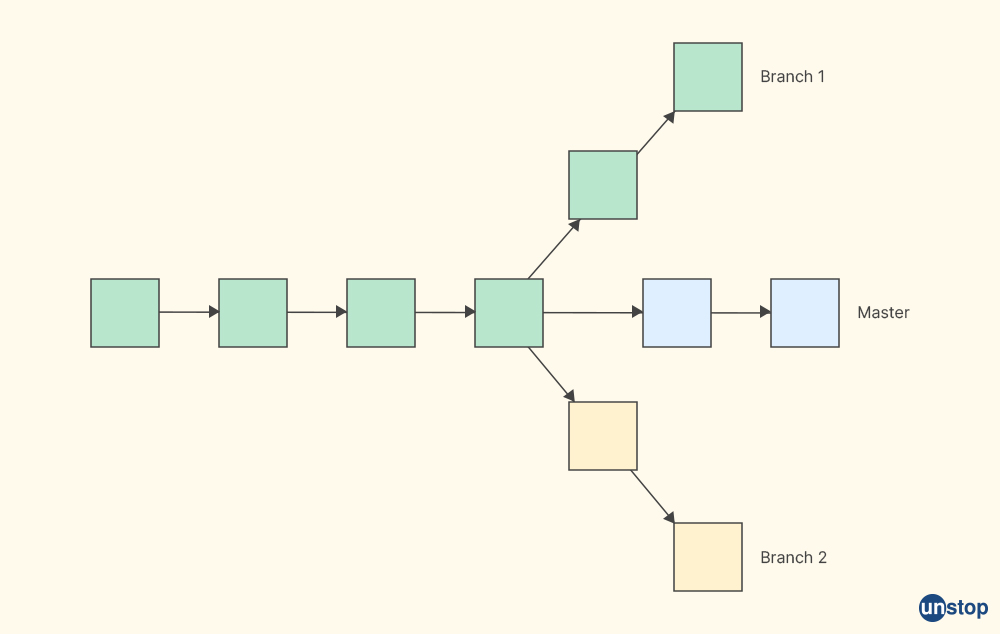
Git is a powerful version control system that helps developers to manage their codebase efficiently. Git Branches are one of the fundamental features that make Git so useful. Git Branches allow developers to create parallel versions of their codebase, where they can experiment with new features, make changes, and fix bugs without affecting the main codebase. This article will cover Git Branches in detail, explaining what they are, how to create and manage them, and how they can be used to streamline the development process.



**What Is Branch In Git?**

A Git Branch is a parallel version of the main codebase. In other words, it is a separate timeline of the codebase where developers can make changes without affecting the main codebase. A branch in Git basically represents an isolated line of development. Every Git Branch has its own set of pointers to commits, which means that developers can make changes to one branch without affecting other branches making the master branch free from questionable code. Git Branches are used to create new features, fix bugs, and test new ideas without disturbing the stability of the main codebase. It is a feature present in most modern version control systems.

**Why Are Git Branches Essential?**

Git branches are essential for several reasons, which are:

* They allow developers to work on different parts of the codebase simultaneously without interfering with each other's everyday development process. For example, if one developer is working on a new feature, they can create a branch for that feature and work on it independently without worrying about other changes being made to the main codebase. They can also check the previous versions, everyday workflow, project history, repository history, code base, and current commit in the feature branch.
* Branches allow developers to experiment with new ideas and approaches without affecting the disk space and stability of the main codebase. If a new feature is not working as expected, it can be abandoned or refined in the branch without affecting the main codebase.
* Git Branches enable developers to collaborate on code changes effectively. By creating a separate branch for a specific feature or bug fix, developers can work together to make changes in the lines of development and review each other's unstable code before merging it back into the main codebase.

**How To Create A New Git Branch?**

Creating a new Git Branch is a straightforward process. Instead of copying files from directory to directory, Git stores a branch as a reference to a previous commit. First, the developer must decide on a name for the new branch head. This name should be descriptive and should reflect the purpose of the new branch head. For example, if a new branch is being created to add a new feature to the codebase, the name of the branch could be "feature/add-new-feature." Once the name of the new branch has been decided, the developer can create the new branch using the Git command:

**git branch <branch-name>**

For example, to create a new branch named "feature/add-new-feature," the developer would use the following command:

**git branch feature/add-new-feature**

After the new branch has been created, the developer can switch to the new branch using the Git command:

**git checkout <branch-name>**

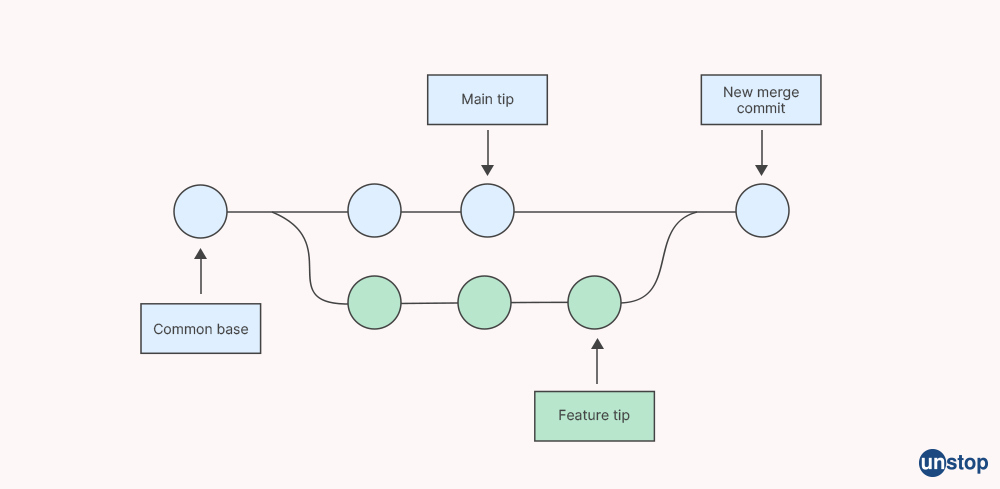
For example, to switch to the new branch named "feature/add-new-feature," the developer would use the following command:

**git checkout feature/add-new-feature**

**Managing Git Branches**

Once a new Git Branch has been created, the developer can start making changes to the codebase in that branch. As the development process progresses, the developer may need to merge changes from one branch into another. For example, if a bug is discovered in the main codebase, the developer may need to create a new branch to fix the bug. Once the bug has been fixed, the developer can merge the changes from the bug-fix branch back into the main codebase.

**Merging Git Branches**



Merging Git Branches is the process of combining the changes from one branch into another to a local or remote repository. When merging Git Branches, Git uses a "three-way merge" algorithm to determine which changes should be kept and which should be discarded. The three-way merge algorithm compares the changes in three different versions of the codebase: the two branches being merged and the common ancestor of the two branches present in the remote repository.

To merge one branch into another in a remote repository, the developer must first switch to the branch that will receive the changes. For example, if the changes in the "feature/add-new-feature" branch need to be merged into the main codebase, the developer would switch to the default branch using the Git command:

**git checkout <main-branch>**

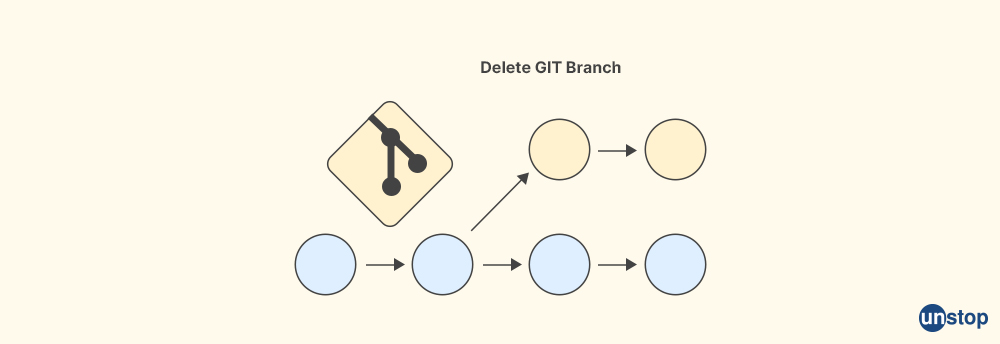
Once the developer has switched to the main branch, they can use the Git Merge command:

**git merge <branch-to-merge>**

For example, to merge the changes from the "feature/add-new-feature" branch into the main codebase, the developer would use the following command:

**git merge feature/add-new-feature**

**How To Delete A Git Branch?**



In Git, you can delete a branch using the ***git branch -d*** command. This command deletes the specified branch, but it will fail if the branch has unmerged changes. If you want the branch deletion regardless of whether it has unmerged changes, you can use the ***git branch -D*** command instead.

Here are the steps to delete a Git branch:

1. First, make sure that you are not currently on the branch that you want to delete. You can use the ***git branch*** command to list all branches in your repository, and the currently active base branch will be marked with an asterisk (\*).

**$ git branch**  
**master**  
**\* feature-branch**  
**new-branch**

1. Switch to a different branch by using the ***git checkout*** command. Replace ***[branch-name]*** with the name of the branch you want to switch to.

**$ git checkout master**

1. Delete the branch using the ***git branch -d*** command. Replace ***[branch-name]*** with the name of the branch you want to delete.

**$ git branch -d feature-branch**

If the branch has unmerged changes and you still want to delete it, use the ***git branch -D*** command instead:

**$ git branch -D feature-branch**

After you delete a branch, it will no longer appear in the output of the ***git branch*** command. However, the branch's commit history will still be preserved in the repository's commit history.

**Git Branch: Commands**

Git branch [commands](https://unstop.com/blog/git-stash) are used to manage branches in a Git repository. Here are some commonly used Git branch operations and commands :

1. **git branch:** This command lists all the branches in the current Git repository.
2. **git branch <branch-name>:** This command creates a new branch with the specified name.
3. **git checkout <branch-name>**: This Git checkout command switches to the specified master branch.
4. **git checkout -b <branch-name>**: This Git checkout command creates a new branch with the specified name and switches to it.
5. **git merge <branch-name>**: This command merges the specified branch into the current branch. It is used to check the merged status of the master branch.
6. **git branch -d <branch-name>**: This command lists the current branch for deletion i.e. it deletes the unneeded branch.
7. **git branch -m <old-branch-name> <new-branch-name>**: The -M <Branch> command renames the specified upstream branch from the old name to the new name.
8. **git branch -a**: This command lists all local and remote branches from the master in the current Git repository.
9. **git branch -r**: This command lists all remote branches after creation in the current Git repository.
10. **git branch -v**: This command lists all branches with the last commit message in the current directory.
11. **git branch --merged**: This command lists all branches that have been merged into the master branch.
12. **git branch --no-merged**: This command lists all branches that have not been merged into the default branches.
13. **git branch -t**: This command allows you to create a new local branch that is set up to track a start-point branch. This means that when you push or pull changes to the previous commit object and from the remote-tracking branches, Git will automatically know which branch to use.

GIT Merge

Merging is an essential [Git](https://phoenixnap.com/kb/what-is-git) operation that combines changes from two branches. Its primary purpose is to integrate changes made in one [branch](https://phoenixnap.com/kb/git-create-new-branch) (the source branch) into another (the target branch) and share those changes with other developers.

**This tutorial shows how to merge a Git branch into the master (main) branch.**

Prerequisites

* Git installed (follow our tutorials to [install Git on Ubuntu](https://phoenixnap.com/kb/how-to-install-git-on-ubuntu), [macOS](https://phoenixnap.com/kb/install-git-on-mac), [Windows](https://phoenixnap.com/kb/how-to-install-git-windows), [CentOS 7](https://phoenixnap.com/kb/how-to-install-git-on-centos-7), or [CentOS 8](https://phoenixnap.com/kb/how-to-install-git-centos-8)).
* A [Git repository](https://phoenixnap.com/kb/what-is-a-git-repository).

## Merge a Git Branch into Master

The **master** (or **main**) branch in Git is a repository's default and primary branch. It usually represents the latest stable version of the project's code. Merging another branch into **master** allows multiple developers to work on different features or fixes concurrently and then bring those changes together into a single branch.

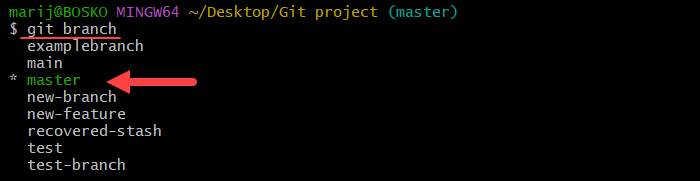
**Note:** Two terms describe the main branch - **master** and **main**. The more neutral term **main** has been more prevalent in recent years due to discussions about inclusivity. Previously, the only term for the main branch was **master**.

Follow the steps below to merge a branch into **master**:

### Step 1: List All Git Branches

List all the branches in your local Git repository using the **git branch** command:

git branch



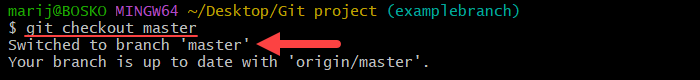
The output shows we are currently on the **master** branch and lists all the other branches in the repository.

**Note:** See how to [list remote branches in Git](https://phoenixnap.com/kb/git-list-remote-branches).

### Step 2: Switch to Master

Ensure you are on the branch you want to merge into. In our case, the **master** branch. Use the **git switch** or **git checkout** command to [switch to the master branch](https://phoenixnap.com/kb/git-switch-branch#ftoc-heading-2) if you are not already on it:

git checkout master



The command switches to the **master** branch.

### Step 3: Merge Branch into Master

After switching, use the **git merge** command to merge another branch into **master**. The merge creates a merge commit, bringing together multiple lines of development while preserving the history of the source branch.

Since merging is a type of commit, it also requires a commit message. There are two ways to specify the commit message:

* Using the **merge** command.
* Specifying the commit message in a text editor.

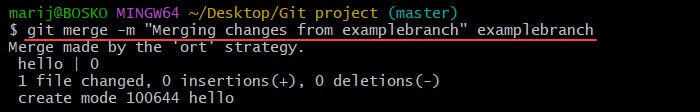
**1. Specify Commit Message Right Away**

To specify the **merge** message right away, use the following syntax:

git merge -m "Your merge commit message" [source\_branch]

* The **-m** option is used to specify a commit message.
* Replace **"Your merge commit message"** with the message you want to use for the merge commit. Enclose the message in double quotes.
* **[source\_branch]** is the name of the branch you want to merge into your current branch.

For example:



The command merges the **examplebranch** branch into the **master** branch and automatically sets the commit message to the one specified in the double quotes.

**Note:** See how to [resolve potential merge conflicts](https://phoenixnap.com/kb/how-to-resolve-merge-conflicts-in-git) or [change a commit message later](https://phoenixnap.com/kb/git-change-commit-message).

**2. Specify Commit Message Separately**

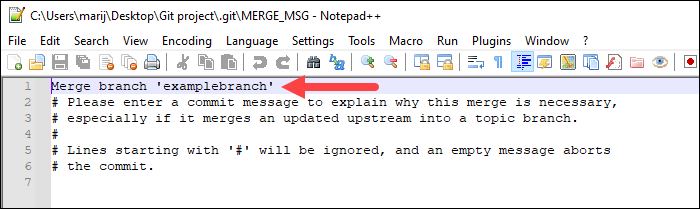
Alternatively, use the following syntax to specify the commit message separately:

git merge [source\_branch]

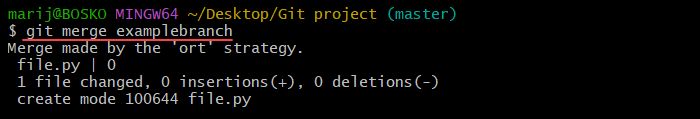
For example:

git merge examplebranch

The command starts the merge process and opens the default [text editor](https://phoenixnap.com/kb/best-linux-text-editors-for-coding), prompting you to enter a commit message for the merge:



In Windows, the Notepad++ editor opens, or whichever is the default one on your system. Specify the merge message, save and close the file, and the merge process completes:



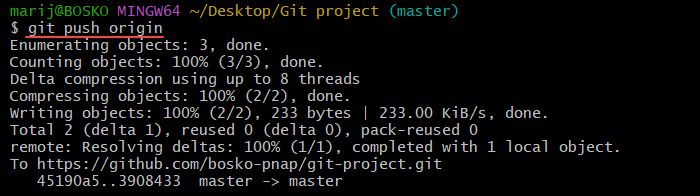
### Step 4: Push Changes

The final step is to [push the local changes](https://phoenixnap.com/kb/git-push-to-remote-branch) to the remote repository so everyone working on the project can [fetch](https://phoenixnap.com/kb/git-fetch) the latest version. The syntax is:

git push [remote\_name]

Replace **[remote\_name]** with the name of the remote repository. For example, if your remote repository is **origin**, run the following command:

git push origin



The command pushes the changes to the remote repository, where they become available to everyone working on the project.

Conclusion

This tutorial showed how to merge a Git branch into the **master** branch. Merging is an essential Git procedure that allows users to bring together multiple lines of development.

**Generate SSH Keys for Authentication**

Introduction

Secure Shell ([SSH](https://phoenixnap.com/kb/what-is-ssh)) is a cryptographic network protocol that ensures the security and integrity of data transmission over the Internet. In [Git](https://phoenixnap.com/kb/what-is-git), SSH boosts data transmission security and represents a powerful tool for authentication and data exchange. The protocol facilitates a secure and efficient version control workflow.

**This tutorial will walk you through setting up SSH and cloning Git repositories using SSH.**

Prerequisites

* Git installed ([install Git on Ubuntu](https://phoenixnap.com/kb/how-to-install-git-on-ubuntu), [macOS](https://phoenixnap.com/kb/install-git-on-mac), [Windows](https://phoenixnap.com/kb/how-to-install-git-windows), [CentOS 7](https://phoenixnap.com/kb/how-to-install-git-on-centos-7), or [CentOS 8](https://phoenixnap.com/kb/how-to-install-git-centos-8)).
* A remote and local [Git repository](https://phoenixnap.com/kb/what-is-a-git-repository).
* An account with administrator privileges.

## Set up SSH

To establish an [SSH connection](https://phoenixnap.com/kb/how-does-ssh-work#ftoc-heading-4), you must create a pair of keys (private and public), share the public key with the service you want to connect to, and set up an SSH agent. The connection requires the user to sign in once, and the SSH agent handles the rest of the authentication throughout the session.

Follow the steps below to [set up SSH](https://phoenixnap.com/kb/setup-passwordless-ssh) on [Linux](https://phoenixnap.com/kb/what-is-linux) or in Git Bash on Windows:

### Step 1: Generate SSH Key Pair

The first step is to generate the SSH key pair. Unix systems have a built-in SSH module, while Windows does not. However, it is possible to use the [Git Bash](https://phoenixnap.com/kb/what-is-git-bash) terminal on Windows to generate the keys.

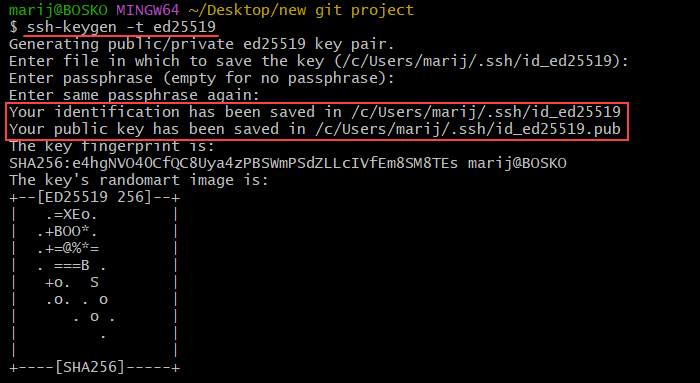
**Note:** If you are on Windows and don't have Git Bash, check out our tutorial for other ways to [generate an SSH key](https://phoenixnap.com/kb/generate-ssh-key-windows-10).

Open a bash terminal on Linux (**Ctrl** + **Alt** + **T**) or Git Bash on Windows and use the following syntax to generate the SSH keys:

ssh-keygen -t ed25519 -C "[info]"

* The **-t** flag allows you to specify the key type. The most commonly used key type for Git is **ed25519**.
* The **-C** flag is optional and provides additional information about the key, such as its purpose or the creator.

For example:



The command prompts you for a location in which to save the files and for a passphrase. To keep the default values, leave everything blank and press **Enter** to confirm. The command creates a pair of keys whose default file names are id\_ed25519 and id\_ed25519.pub and saves them in the specified location.

**Note:** If you have previously created SSH keys, the **ssh-keygen** command asks if you want to overwrite the existing key. In that case, it is better to create a custom-named SSH key. To do so, specify the **-f** flag followed by the path to the key with your custom key name.

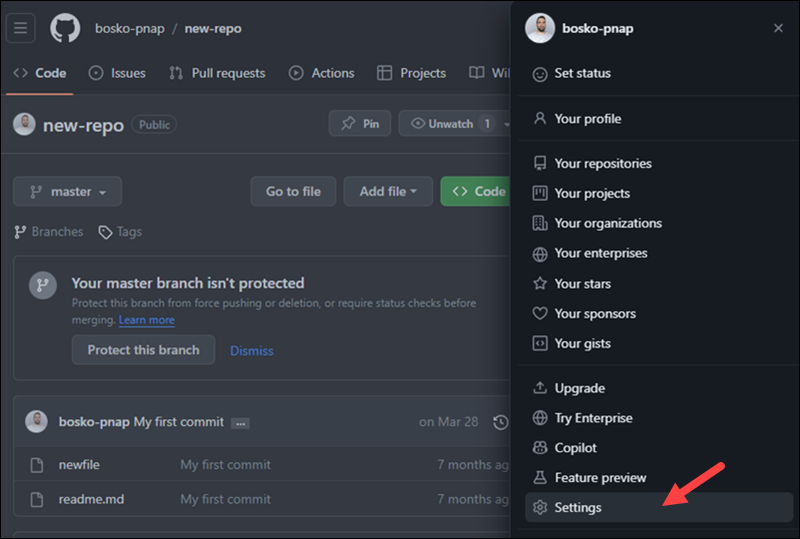
### Step 2: Add Key To GitHub

After generating the key pair, add the public key to the service you want to connect to over SSH. In this tutorial, we will use GitHub.

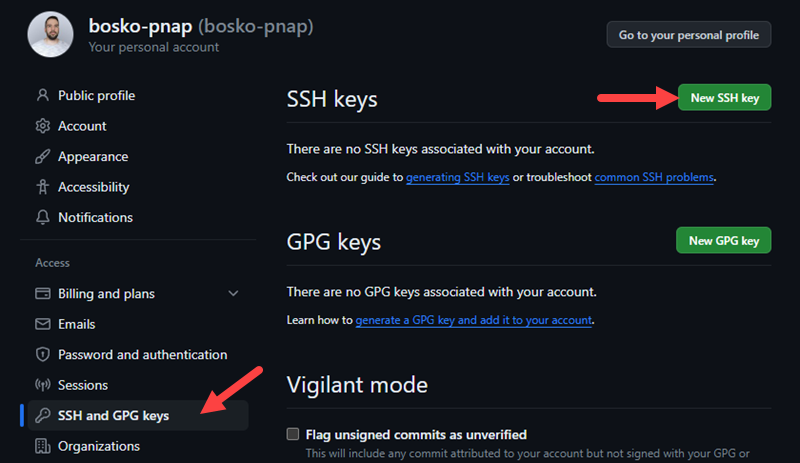
Follow the steps below:

1. [Log in to your GitHub](https://github.com/login) account.

2. In the top right corner, click your account image and select **Settings**.

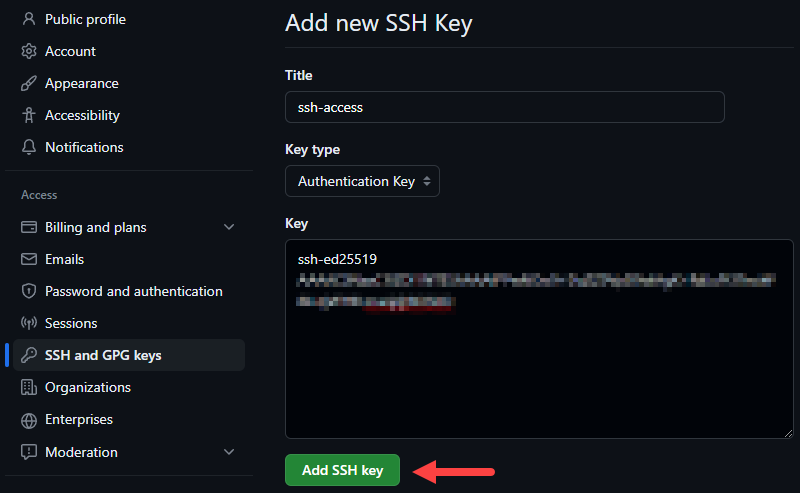


3. Click the **SSH and GPG keys** section and select the **New SSH** **key** button.

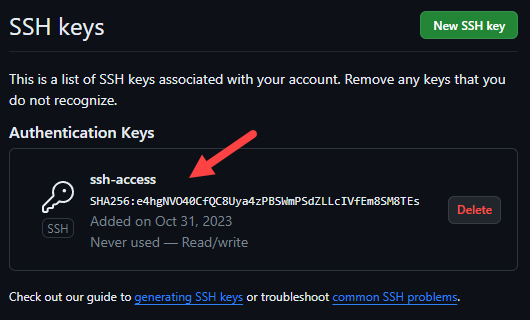


4. On the Add new SSH Key page, provide a name for your SSH key. Since you can assign multiple keys to your account, give them descriptive names for easier navigation. Make sure to add the [public key](https://phoenixnap.com/kb/ssh-with-key) you have previously generated, **not the private one**. Copy the key contents to the designated field.

For example:



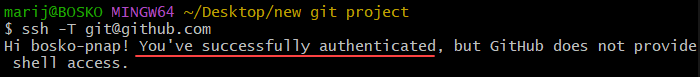
After adding the key, it should appear in the **Authentication Keys** section:



### Step 3: Test the Connection

After adding a new key, test your connection to make sure everything works as it's supposed to. In the terminal or Git Bash, run the following command:

ssh -T git@github.com



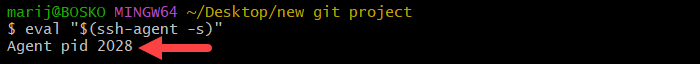
The command output states that you have successfully authenticated. However, in case you get an error saying "[permission denied](https://phoenixnap.com/kb/ssh-permission-denied-publickey)," make sure that you have correctly set up everything. If the error persists, you can delete the keys (locally and in GitHub), generate a new pair, and try again.

### Step 4: Check Local SSH Agent

The SSH agent is part of the SSH toolkit. It comes by default in the Unix-like systems and in Git Bash. The SSH agent holds the private key from the key pair we have previously generated.

Before adding the private key to the SSH agent, use the following command to make sure that it is running:

eval "$(ssh-agent -s)"



The output shows that the SSH agent is up and running and displays the process ID.

**Note:** If you are on Linux, prepend the command with [**sudo**](https://phoenixnap.com/kb/linux-sudo-command).

### Step 5: Add Keys to SSH Agent

The next step is to add the keys to the SSH agent. Use the following syntax:

ssh-add [path-to-private-key]

Specify the entire path to the private key you generated earlier.

For example:

Adding the private SSH key to the SSH agent.

The command adds the key to the agent, and you are now ready to clone the [repository](https://phoenixnap.com/glossary/what-is-a-repository).

## Clone Repository Using SSH Protocol

To clone a Git repository using SSH, use the **git clone** command followed by a valid SSH [URL](https://phoenixnap.com/glossary/url-definition-meaning). The command syntax is:

git clone git@host:username/repository.git

* **host** is the [domain](https://phoenixnap.com/glossary/what-is-a-domain) name or the [IP address](https://phoenixnap.com/glossary/what-is-an-ip-address) of the hosting server. In our case, it is github.com.
* **username** is your user account on GitHub.
* **repository** is the name of the Git repository you want to clone.

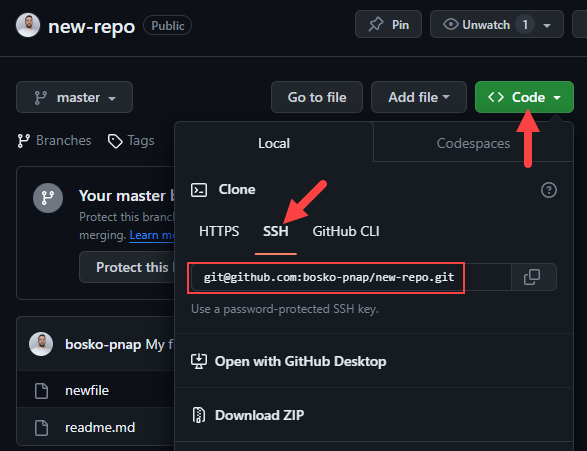
You can find all the information on your repository page. Follow the steps below:

### Step 1: Go to Repository Page

Log in to your account on GitHub and go to the repository page you want to clone.

### Step 2: Obtain SSH URL

On the repository page, click the **<> Code** button to obtain the SSH URL:

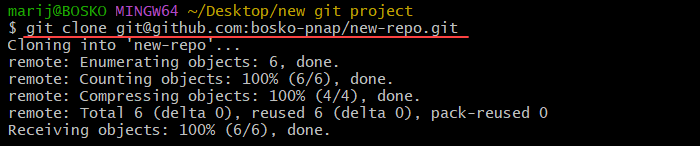


Make sure to select the **SSH** option and copy the code snippet.

### Step 3: Clone Repository

Paste the SSH URL as an argument to the **git clone** command in Git Bash. For example:

git clone git@github.com:bosko-pnap/new-repo.git



The command clones the repository to your local machine over SSH.

## Cloning with SSH vs. HTTPS

Cloning a Git repository with [SSH instead of HTTPS](https://phoenixnap.com/kb/git-ssh-vs-https) offers advantages in terms of [authentication](https://phoenixnap.com/glossary/what-is-authentication) and security. When opting for HTTPS, Git requires you to input your username and password during the authentication process, which can be a potential security risk if not managed carefully.

On the other hand, SSH provides a more secure method for cloning repositories. Instead of transmitting sensitive credentials over the network, SSH uses cryptographic keys for authentication. This approach enhances security and eliminates the need to enter login details repeatedly.

With SSH, only the machines with the corresponding key file can access the repositories, reducing the likelihood of unauthorized access. Even if the SSH key file gets stolen, the potential damage is limited since it doesn't grant access to the entire account. Additionally, you can easily revoke and replace the compromised key, further improving the security of your Git workflow.

Conclusion

This tutorial has outlined the steps required to create an SSH key pair and add it to your GitHub account. The instructions showed how to clone repositories with improved security and reduced reliance on sensitive credentials.

Next, learn the difference between [SSH and SSL](https://phoenixnap.com/kb/ssh-vs-ssl) or see how [SSH differs from Telnet](https://phoenixnap.com/kb/telnet-vs-ssh).