Guide to Git - Common Git Commands

Overview

You can access all of Git™'s functionality via the command line. This document lists common commands and options that may assist you when you learn Git.

Important:

- This document is **not** comprehensive. It intentionally omits information about intermediate and advanced Git functionality. For more information about additional commands and options, read Git's documentation.
- The terminology in this document assumes basic familiarity with the command line.

Common Git commands

Notes:

- To resolve Git errors, read our Guide to Git - Troubleshooting documentation.
- For general Linux commands, read our Getting Started with Linux Commands documentation.
- For help to access the command line in order to run these commands, read our How to Access the Command Line documentation.
- You can use cPanel's *Terminal* interface (*cPanel* >> *Home* >> *Advanced* >> *Terminal*) to access the command line from within the cPanel interface.
- You can add the `--help` option to any Git command in order to view the manual page for that command.

git clone

This command clones a repository into a new directory, creates remote-tracking branches, and forks a new working branch from the cloned repository's active branch.

```
git clone repositoryurl
```

In the example above, `repositoryurl` represents the URL of the repository that you wish to clone.

Notes:

- Use the `git fetch` command to update the new repository's remote-tracking branches.
- Use the `git pull` command to merge the remote master branch into the current master branch.
- cPanel's *Git™ Version Control* interface (*cPanel >> Home >> Files >> Git™ Version Control*) provides the URL to use to clone each of your account's repositories.

git add

This command adds the current version of a file to the index of staged content for the next commit.

```
git add [options] filepath
```

In the example above, `filepath` represents the file's absolute path **or** its path relative to the current working directory.

- To stage uncommitted changes for **all** tracked files, run this command with either of the `-a` or `-u` options (and without a specified file path).
- This command **only** stages the current changes for the current commit. The next time that you create a commit, you **must** run the command for the file again in order to stage any new changes.

git commit

This command creates a new commit for the currently-staged changes.

```
git commit [options]
```

When you run this command (without the `-m` option), Git immediately displays a text file, in which you can enter and save your commit message.

- To automatically stage modified and deleted files before Git creates the commit, run this command with the `-a` option.
- To specify a short commit message directly from the command line, run this command with the `-m` option. For example:

```
git commit -m "Commit message here."
```

Note:
To stage changes for inclusion in a commit, use the `git add` or `git rm` commands or provide individual filepaths as arguments to this command.

`git checkout`

This command sets the specified branch as the current working branch.

```
git checkout [options] branchname
```

In the example above, `branchname` represents the branch to check out.

- To check out only a specified file, run this command with a file path instead of a branch name.

  ```
  git checkout mybranch files/templates/2.html
  ```

  In this example, `mybranch` represents the branch that contains the version of the file that you wish to check out and `files/templates/2.html` represents the file to check out. If you run this command, the system will replace the `files/templates/2.html` file's contents in the current local working branch with the file's contents from the `mybranch` branch.

  > **Note:**
  > If you omit the branch name, Git will check out that file from the HEAD of the current branch.

- To create a new branch with the specified branch name and then check it out, run this command with the `-b` option.
- To forcibly change branches, run the command with the `-f` option. This option causes Git to overwrite local changes in order to match the working tree to the branch's HEAD commit.

`git rm`

This command removes files or directories from Git's index and working tree.

```
git rm [options] files_or_dirs
```

In the example above, `files_or_dirs` represents the paths to the files or directories to remove, relative to the repository's main directory.

> **Important:**
> - To run this command, the specified file **cannot** contain uncommitted changes.
> - This command **cannot** retain the file in the index **and** remove the file from the working tree. To do this, use BASH's `rm` command.
> - If you specify a directory name, you **must** also use the `-r` option. This option allows the command to recursively remove the files in that directory.

`git fetch`

This command downloads branches, tags, and their histories from one or more other repositories.

```
git fetch [options] remotename
```

In the example above, `remotename` represents the name of the remote repository.

`git pull`

This command fetches and merges changes from a local branch or a remote or local repository. With most options, this command combines the `git fetch` and `git merge` commands.

```
git pull [options] repo-or-branch
```

In the example above, `repo-or-branch` represents the branch name or the repository name or URL.

`git push`

This command adds your committed changes to the specified repository and branch.

```
git push [options] repository branch
```

In the example above, `repository` represents the repository name or URL and `branch` represents the remote branch on that repository.

- If you do **not** specify a repository, the command performs one of the following actions:
  - If your current branch's configuration includes a remote repository, the command adds your changes to that repository.
  - If your current branch's configuration does **not** include a remote repository, the command adds your changes to the `origin` repository.

> **Important:**
> You **must** explicitly specify a repository in order to specify a branch. If you do **not** specify a branch, the command adds your changes to the remote repository's current branch.

- To push **all** commits from **all** local branches to their upstream repositories, run this command with the `--all` option.
- To add the specified repository to the branch as its upstream repository, run this command with the `--set-upstream` option.
  - This allows you to omit the repository on subsequent pushes to upstream.
  - You **must** specify a remote repository when you use this option.

> **Note:**
> cPanel's *Git Version Control* feature (*cPanel >> Home >> Files >> Git Version Control*) automatically adds a `post-receive` hook that each push to cPanel-managed repositories triggers. For more information, read our Guide to Git - Deployment documentation or Git's githooks documentation.

`git branch`

This command creates, lists, or deletes branches.

```
git branch [options] branchname
```

In the example above, `branchname` represents the branch name.

- To create a new branch, run this command with the desired branch name.

> **Important:**
> Git does **not** automatically check out new branches when you create them. You **must** also run the `git checkout` command in order to check out your new branch.

- To retrieve a list of existing local branches, run this command without a branch name. Use the `-a` option to retrieve a list of both local and remote branches.
- To set the upstream branch for a specified branch, run this command with the `-u` option.
- To rename a specified branch, run this command with the `-m` option and the current and new branch names. For example:

```
git branch -m oldbranch newbranch
```

In this example, `oldbranch` represents the current branch name and `newbranch` represents the new branch name.

- To delete a specified branch, run this command with the `-d` option.

git merge

This command combines the history of one or more commits into the history of the current branch.

```
git merge [options]
```

> **Note:**
> The `git pull` command automatically performs this action.

git blame

This command displays the specified file with the author, most-recent change date, and commit SHA-1 for each line of the file.

```
git blame [options] filepath
```

In the example above, `filepath` represents the file's absolute path **or** its path relative to the current working directory.

When you run this command without additional options, the output will resemble the following example:

```
5f033c48d84a (Jane E. Coder        2017-05-24 18:25:53 -0600  1) <!DOCTYPE
HTML>
5f033c48d84a (Jane E. Coder        2017-05-24 18:25:53 -0600  2) <html>
5f033c48d84a (Jane E. Coder        2017-05-24 18:25:53 -0600  3)
5f033c48d84a (Jane E. Coder        2017-05-24 18:25:53 -0600  4) <body>
54222e949682 (John B. Developer    2018-01-08 10:57:07 +0000  5)
<p>Here's some text.</p>
5f033c48d84a (Jane E. Coder        2017-05-24 18:25:53 -0600  6)
5f033c48d84a (Jane E. Coder        2017-05-24 18:25:53 -0600  7) <script>
```

In this example, on May 24th, 2017, Jane E. Coder committed changes to the file. On January 8th, 2018, John B. Developer committed changes to the file. Because John committed changes after Jane, this output does **not** display any changes that Jane made to line 5 **or** the history of the other lines before Jane's commit.

git clean

This command removes untracked files (files that Git does not manage) from the working tree.

```
git clean [options]
```

- To display a list of untracked files to remove but **not** remove them, run this command with the `-n` option.
- To also remove untracked directories, run this command with the `-d` option.

git config

This command retrieves or updates Git's global and repository settings in its configuration files.

```
git config [options]
```

Git stores your settings in the following files:

- `/path-to-git/etc/gitconfig` — Global settings.
- `/path-to-git/config` — Repository settings.
- `/home-directory/.gitconfig` — A user configuration file.
- `/home-directory/.config/git/config` — A user configuration file.

Notes:

- In the paths above, `path-to-git` represents the Git installation's absolute path and `home-directory` represents a cPanel account's home directory (for example, the `/home/user/.gitconfig` file.).
- If both user configuration files exist and their values conflict, the system uses the values in the `.gitconfig` file.
- This command accepts many options for each of Git's configurable settings. To use this command, read Git's git config documentation.

git diff

This command compares changes between two commits, a commit and the current working tree, two branches or working trees, or two files.

```
git diff [options]
```

By default, this command returns a comparison of the working tree and your last commit (the changes that Git would commit if you ran the `git commit -a` command).

You may wish to use the following common options:

- To view a comparison of two branches, run the following command, where `branch1` and `branch2` represent the branches to compare:

  ```
  git diff branch1..branch2
  ```

- To view a comparison of two commits, run the following command, where `FirstSHA` and `SecondSHA` represent the SHA-1 values for the two commits:

  ```
  git diff FirstSHA..SecondSHA
  ```

- To only view differences between two versions of one file in a working tree, branch, or commit, specify that filepath as an argument. For example:

  ```
  git diff branch1..branch2 filename
  ```

  In the example above, `branch1` and `branch2` represent the branches from which Git will compare the contents of the `filename` file.

git grep

This command searches the current working tree for one or more patterns (generally, strings or regular expressions).

```
git grep [options] "pattern"
```

In the example above, `pattern` represents the data to query.

- To perform a case-insensitive search, run this command with the `-i` option.
- To use Perl-Compatible Regular Expressions (PCREs) in your patterns, run this command with the `--perl-regexp` option. cPanel & WHM's implementation of Git automatically includes the necessary dependencies for this option.
- To return **only** files that include **all** of the specified patterns (when you run the command with multiple patterns), run this command with the `--all-match` option. For example:

  ```
  git grep --all-match "string one" "string two" "string three"
  ```

  This example would return files that contain `string one`, `string two`, **and** `string three`, but would **not** return files that only contain `string two`.
- To return file paths relative to the repository's main directory rather than the current directory, run this command with the `--full-name` option.

git log

This command queries the commit logs for your current branch.

```
git log [options]
```

- To view only results from a specific range of commits, run the following command:

  ```
  git log FirstSHA..SecondSHA
  ```

  In this example, `FirstSHA` and `SecondSHA` represent the SHA-1 values for the two commits.

  > **Note:**
  > If you do not specify a range of commits to query, this command queries all commits between the `origin` commit and HEAD for the current branch.

- To view only a specific number of the most recent log entries, run the following command, where `num` represents the number of entries to return:

  ```
  git log -num
  ```

- To view only log entries before or after a specific date, run one of the following commands, where `date` represents the specified date:

  ```
  git log --before=date
  git log --after=date
  ```

  For date formatting options, read Git's git log documentation.

- To view only log entries for commits from a specific author, run the following command, where `authorname` represents the author's name in their `.gitconfig` file:

  ```
  git log --author=authorname
  ```

- To view only log entries that contain a specific pattern (generally, a string), run the following command, where `pattern` represents the pattern to query:

  ```
  git log --grep=pattern
  ```

  If you include multiple patterns to query, use the `--all-match` option to limit output to log entries that match **all** of the specified patterns.

Notes:
- This command also accepts formatting options from the `git diff` command.
- If you only require summarized commit log information, you may wish to use the `git shortlog` command.

`git revert`

This command reverts existing commits within a specified range and then allows you to edit their commit messages.

```
git revert [options] commit1..commit2
```

In the example above, `commit1` and `commit2` represent the SHA-1 values for the range of commits to revert.

Important:
To run this command, your working tree **cannot** contain uncommitted changes.

`git shortlog`

This command produces a shortened version of the output of the `git log` command. You may wish to use this command if, for example, you need to generate a list of changes for release notes or a change log.

```
git shortlog [options]
```

`git stash`

This command uses several options to create, manage, and retrieve sets of changes (stashes). When you run this command without specified options, it defaults to `git stash save` functionality.

```
git stash [options]
```

Use the following options to manage stashes:

- To create a new stash and return the current branch to its state in the HEAD commit, run this command with the `save` option.

  Note:
  When you use this option, you can also use the `-message` option to add a description to the stash. For example:

  ```
  git stash save -message "Description"
  ```

  In the example above, `Description` represents the stash description.

- To return a comparison of stashed changes with the HEAD commit when you created the stash, run this command with the `show mystash` option.
- To list your current stashes, run this command with the `list` option.
- To apply stashed changes to the current working tree and **and** remove the stash, run this command with the `pop mystash` option.
- To apply stashed changes to the current working tree but **not** remove the stash, run this command with the `apply mystash` option.
- To remove all stashes, run this command with the `clear` option.

Note:
In the options above, `mystash` represents the reflog entry or stash index for the desired stashed changes.

Additional documentation

Suggested documentation For cPanel users For WHM users For developers

- Guide to Git - Deployment
- Guide to Git - For System Administrators
- Guide to Git
- Guide to Git - Common Git Commands
- Guide to Git - Host Git Repositories on a cPanel Account

Error rendering macro 'contentbylabel' : parameters should not be empty

Error rendering macro 'contentbylabel' : parameters should not be empty

Content by label

There is no content with the specified labels