

[theserverside.com](https://www.theserverside.com)

An example of how to use the git stash command

By Cameron McKenzie, TechTarget Published: 20 Oct 2020

4-5 minutes

[Get started](#) Bring yourself up to speed with our introductory content.

-
-

Let's envision a common developer scenario.

A developer retrieves the latest code from GitHub and begins to make changes to a few source files to enhance a feature, only to be told by the team lead to stop what they're doing to fix some bugs.

Now that developer doesn't want to commit their code because it's experimental and it would break the rest of the build if another developer attempted to merge with it. At the same time, the

developer doesn't want to do a hard reset of the working tree because it might lose all the changes.

If only there was a command that merged the benefits of a hard reset with the persistent nature of a standard commit.

Luckily, there is. It's called the git stash command and it's incredibly useful.

The git stash command explained

Developers who use the git stash command perform the following steps:

1. Start off with a fresh Git commit.
2. Edit some source files.
3. Issue a git stash command.
4. Develop as though the code base was set back to the fresh commit, because it was.
5. Continue to commit code as needed.
6. Call the git stash pop command at any point to apply the shelved files.

Here's a simple example of how to use the git stash command.

First, a developer will initialize a Git repository, add two files to the [Git worktree](#) and issue a commit:

```
git init
echo "A solid start." >> solid.html
touch "This may get flakey." >> flakey.html
git commit -m "Situation normal"
```

A git stash command example

Then, a developer can start to experiment with the flakey.html file and add some highly experimental code.

```
echo "Good but experimental content." >> flakey.html
```

Now imagine that they're told to set aside all the flakey but experimental work and instead add a fix to the solid.html file. Developers wouldn't want to lose the changes in the flakey.html file, but it might break the build if they performed a [Git commit](#). Plus, they don't want to do a [hard Git reset](#) because they'd lose the experimental changes forever.

The right thing to do is to use the git stash command.

Git stash command in action

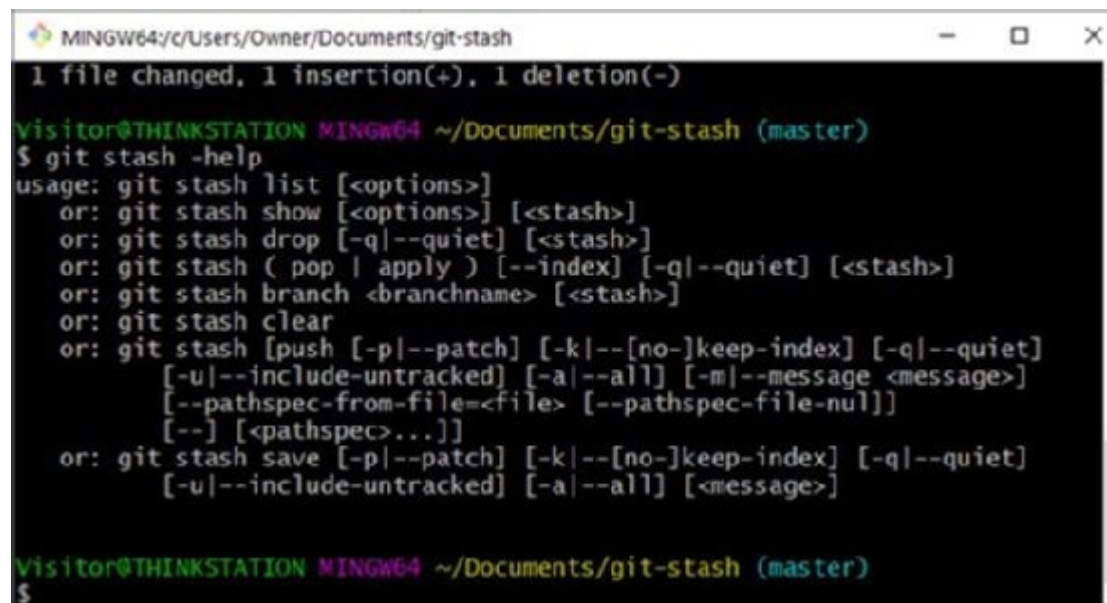
If developers inspect the contents of both the solid.html and flakey.html file after they use the git stash command, they would see that both files have returned to their pre-commit state. When the git stash command runs successfully, [the Git workspace](#) is reset to where it was immediately prior to the last commit.

After the developer issues the git stash command, it's safe to implement new code changes and perform commits as usual. Here's how to fix and commit the solid.html file.

```
echo "This is a solid fix" >> solid.html
```

```
git add .
```

```
git commit -m "Solid file is fixed."
```

A screenshot of a terminal window titled 'MINGW64/c/Users/Owner/Documents/git-stash'. The window shows the output of the 'git stash -help' command. At the top, it says '1 file changed, 1 insertion(+), 1 deletion(-)'. Below that, the prompt is 'Visitor@THINKSTATION MINGW64 ~/Documents/git-stash (master)'. The command '\$ git stash -help' is entered, and the output shows the usage for 'git stash' with various options like 'list', 'show', 'drop', 'pop', 'apply', 'branch', 'clear', 'push', 'save', etc. The prompt '\$' is visible at the bottom of the terminal window.

```
MINGW64/c/Users/Owner/Documents/git-stash
1 file changed, 1 insertion(+), 1 deletion(-)
Visitor@THINKSTATION MINGW64 ~/Documents/git-stash (master)
$ git stash -help
usage: git stash list [<options>]
or: git stash show [<options>] [<stash>]
or: git stash drop [-q|--quiet] [<stash>]
or: git stash ( pop | apply ) [--index] [-q|--quiet] [<stash>]
or: git stash branch <branchname> [<stash>]
or: git stash clear
or: git stash [push [-p|--patch] [-k|--[no-]keep-index] [-q|--quiet]
      [-u|--include-untracked] [-a|--all] [-m|--message <message>]
      [--pathspec-from-file=<file> [--pathspec-file-nul]]
      [--] [<pathspect>...]
or: git stash save [-p|--patch] [-k|--[no-]keep-index] [-q|--quiet]
      [-u|--include-untracked] [-a|--all] [<message>]

Visitor@THINKSTATION MINGW64 ~/Documents/git-stash (master)
$
```

The git stash command can shelve temporary changes, and then

later reapply those changes to the workspace.

Git stash pop command use

Now that the fix is in, it's time to get back the experimental changes made to the flakey.html file. All of those changes are stored locally in the stash. Developers can resurrect those changes and reapply them to their current workspace with the git stash pop command.

This command brings back the content that was previously stashed, namely the changes to the flakey.html file. If we now inspect all the files in the workspace, we will see that the experimental changes made to the flakey.html file are restored, and the changes to the solid.html file made after the git stash command was issued are still intact.

Whenever developers need to [temporarily shelve](#) some code changes with the intent on returning to that code sometime in the near future, use the git stash command. It's just what developers need in that situation.