NAME

git-stash - Stash the changes in a dirty working directory away

SYNOPSIS

git stash list [<options>]
*git stash*sho w [<stash>]
*git stash*drop [-q|--quiet] [<stash>]
git stash(pop | apply) [--index] [-q|--quiet] [<stash>]
*git stash*branc h <branchname> [<stash>]
git stash[sa ve [-p|--patch] [-k|--[no-]keep-index] [-q|--quiet]
[-u|--include-untracked] [-a|--all] [<message>]]
*git stash*clear
*git stash*create [<message>]
*git stash*store [-m|--message <message>] [-q|--quiet] <commit>

DESCRIPTION

Use git stash when you want to record the current state of the working directory and the index, but want to go back to a clean working directory. The command saves your local modifications away and reverts the working directory to match the HEAD commit.

The modifications stashed away by this command can be listed with git stash list, inspected with git stash show, and restored (potentially on top of a different commit) with git stash apply. Calling git stash without any arguments is equivalent to git stash save. A stash is by default listed as WIP on *branchname* ..., but you can give a more descriptive message on the command line when you create one.

The latest stash you created is stored in refs/stash; older stashes are found in the reflog of this reference and can be named using the usual reflog syntax (e.g. stash@{0} is the most recently created stash, stash@{1} is the one before it, stash@{2.hours.ago} is also possible).

OPTIONS

save [-p|--patch] [-k|--[no-]keep-index] [-u|--include-untracked] [-a|--all] [-q|--quiet] [<message>]
Save your local modifications to a new *stash*, and run git reset --hard to revert them. The <message> part is optional and gives the description along with the stashed state. For quickly making a snapshot, you can omit *both* save and <message>, but giving only <message> does not trigger this action to prevent a misspelled subcommand from making an unwanted stash.

If the --keep-index option is used, all changes already added to the index are left intact.

If the --include-untracked option is used, all untracked files are also stashed and then cleaned up with git clean, leaving the working directory in a very clean state. If the --all option is used instead then the ignored files are stashed and cleaned in addition to the untracked files.

With --patch, you can interactively select hunks from the diff between HEAD and the working tree to be stashed. The stash entry is constructed such that its index state is the same as the index state of your repository, and its worktree contains only the changes you selected interactively. The selected changes are then rolled back from your worktree. See the "Interactive Mode" section of **git-add(1)** to learn how to operate the --patch mode.

The --patch option implies --keep-index. You can use --no-keep-index to override this.

list [<options>]
List the stashes that you currently have. Each *stash* is listed with its name (e.g. stash@{0} is the latest stash, stash@{1} is the one before, etc.), the name of the branch that was current when the stash was made, and a short description of the commit the stash was based on.

stash@{0}: WIP on submit: 6ebd0e2... Update git-stash documentation
stash@{1}: On master: 9cc0589... Add git-stash

The command takes options applicable to the *git log* command to control what is shown and

how. See **git-log(1)**.

show [<stash>]
: Show the changes recorded in the stash as a diff between the stashed state and its original parent. When no <stash> is given, shows the latest one. By default, the command shows the diffstat, but it will accept any format known to *git diff* (e.g., git stash show -p stash@{1} to view the second most recent stash in patch form).

pop [--index] [-q|--quiet] [<stash>]
: Remove a single stashed state from the stash list and apply it on top of the current working tree state, i.e., do the inverse operation of git stash save. The working directory must match the index.

  Applying the state can fail with conflicts; in this case, it is not removed from the stash list. You need to resolve the conflicts by hand and call git stash drop manually afterwards.

  If the --index option is used, then tries to reinstate not only the working tree's changes, but also the index's ones. However, this can fail, when you have conflicts (which are stored in the index, where you therefore can no longer apply the changes as they were originally).

  When no <stash> is given, stash@{0} is assumed, otherwise <stash> must be a reference of the form stash@{<revision>}.

apply [--index] [-q|--quiet] [<stash>]
: Like pop, but do not remove the state from the stash list. Unlike pop, <stash> may be any commit that looks like a commit created by stash save or stash create.

branch <branchname> [<stash>]
: Creates and checks out a new branch named <branchname> starting from the commit at which the <stash> was originally created, applies the changes recorded in <stash> to the new working tree and index. If that succeeds, and <stash> is a reference of the form stash@{<revision>}, it then drops the <stash>. When no <stash> is given, applies the latest one.

  This is useful if the branch on which you ran git stash save has changed enough that git stash apply fails due to conflicts. Since the stash is applied on top of the commit that was HEAD at the time git stash was run, it restores the originally stashed state with no conflicts.

clear
: Remove all the stashed states. Note that those states will then be subject to pruning, and may be impossible to recover (see *Examples* below for a possible strategy).

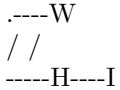
drop [-q|--quiet] [<stash>]
: Remove a single stashed state from the stash list. When no <stash> is given, it removes the latest one. i.e. stash@{0}, otherwise <stash> must be a valid stash log reference of the form stash@{<revision>}.

create
: Create a stash (which is a regular commit object) and return its object name, without storing it anywhere in the ref namespace. This is intended to be useful for scripts. It is probably not the command you want to use; see save above.

store
: Store a given stash created via *git stash create* (which is a dangling merge commit) in the stash ref, updating the stash reflog. This is intended to be useful for scripts. It is probably not the command you want to use; see save above.

DISCUSSION

A stash is represented as a commit whose tree records the state of the working directory, and its first parent is the commit at HEAD when the stash was created. The tree of the second parent records the state of the index when the stash is made, and it is made a child of the HEAD commit. The ancestry graph looks like this:



where H is the HEAD commit, I is a commit that records the state of the index, and W is a commit that records the state of the working tree.

EXAMPLES

Pulling into a dirty tree

When you are in the middle of something, you learn that there are upstream changes that are possibly relevant to what you are doing. When your local changes do not conflict with the changes in the upstream, a simple git pull will let you move forward.

However, there are cases in which your local changes do conflict with the upstream changes, and git pull refuses to overwrite your changes. In such a case, you can stash your changes away, perform a pull, and then unstash, like this:

```
$ git pull
...
file foobar not up to date, cannot merge.
$ git stash
$ git pull
$ git stash pop
```

Interrupted workflow

When you are in the middle of something, your boss comes in and demands that you fix something immediately. Traditionally, you would make a commit to a temporary branch to store your changes away, and return to your original branch to make the emergency fix, like this:

```
# ... hack hack hack ...
$ git checkout -b my_wip
$ git commit -a -m WIP
$ git checkout master
$ edit emergency fix
$ git commit -a -m Fix in a hurry
$ git checkout my_wip
$ git reset --soft HEAD^
# ... continue hacking ...
```

You can use *git stash* to simplify the above, like this:

```
# ... hack hack hack ...
$ git stash
$ edit emergency fix
$ git commit -a -m Fix in a hurry
$ git stash pop
# ... continue hacking ...
```

Testing partial commits

You can use git stash save --keep-index when you want to make two or more commits out of the changes in the work tree, and you want to test each change before committing:

```
# ... hack hack hack ...
$ git add --patch foo # add just first part to the index
$ git stash save --keep-index # save all other changes to the stash
$ edit/build/test first part
$ git commit -m First part # commit fully tested change
$ git stash pop # prepare to work on all other changes
```

```
# ... repeat above five steps until one commit remains ...
$ edit/build/test remaining parts
$ git commit foo -m Remaining parts
```

Recovering stashes that were cleared/dropped erroneously

If you mistakenly drop or clear stashes, they cannot be recovered through the normal safety mechanisms. However, you can try the following incantation to get a list of stashes that are still in your repository, but not reachable any more:

```
git fsck --unreachable |
grep commit | cut -d -f3 |
xargs git log --merges --no-walk --grep=WIP
```

SEE ALSO

git-checkout(1), **git-commit(1)**, **git-reflog(1)**, **git-reset(1)**

GIT

Part of the **git(1)** suite