How to name and retrieve a stash by name in git?

Asked 7 years, 2 months ago Active 20 days ago Viewed 423k times

1227

401

I was always under the impression that you could give a stash a name by doing `git stash save stashname` , which you could later on apply by doing `git stash apply stashname` . But it seems that in this case all that happens is that `stashname` will be used as the stash description.

Is there no way to actually name a stash? If not, what would you recommend to achieve equivalent functionality? Essentially I have a small stash which I would periodically like to apply, but don't want to always have to hunt in `git stash list` what its actual stash number is.

git git-stash

edited Apr 12 '16 at 18:16
Geoffrey Hale
6,180 3 30 37

asked Jun 29 '12 at 21:33
Suan
12.3k 12 42 59

34 `git stash push -m stashname` is the current syntax. `git stash save stashname` has been deprecated. – SherylHohman Apr 25 '18 at 18:08

1 git stash push -m stashname doesn't work in 2.8.0.windows.1. – Jac Jun 10 '18 at 10:29

19 Answers

609

This is how you do it:

```
git stash save "my_stash"
git stash list
```

This will list down all your stashes.

To apply a stash and remove it from the stash stack, type:

```
git stash pop stash@{n}
```

To apply a stash and keep it in the stash stack, type:

```
git stash apply stash@{n}
```

Where `n` is the index of the stashed change.

edited Mar 28 at 15:29
mbds
7 4

answered Mar 4 '13 at 8:18
Sri Murthy Upadhyayula
7,231 1 12 20

67 This does not answer the question. By default you end up with a bunch of numbers for your stash , but this doesnt answer how you can put a name to identify easily. – GoodSp33d Sep 5 '14 at 9:15

8 OP is explicitly trying to avoid the awkwardly named stash@{n} names for custom name. `git stash apply <custom-name>` – stewSquared Jan 13 '17 at 21:13

5 Doesn't answer the question about retrieving a stash by name. – nullsteph Feb 2 '17 at 15:46

24 `git stash push -m my_stash` is the current syntax. `git stash save my_stash` has been deprecated. – SherylHohman Apr 25 '18 at 18:11

14 It is not irrelevant. It is useful. – Gayan Weerakutti May 2 '18 at 9:26

273

`git stash save` is **deprecated** as of 2.15.x/2.16, instead you can use `git stash push -m "message"`

You can use it like this:

```
git stash push -m "message"
```

In order to retrieve the stash you can use: `git stash list` . This will output a list like this, for example:

```
stash@{0}: On develop: perf-spike
stash@{1}: On develop: node v10
```

Then you simply use `apply` giving it the `stash@{index}` :

```
git stash apply stash@{1}
```

References [git stash man page]

edited Mar 14 at 18:04
jlafay
9,213 12 64 90

answered Mar 29 '18 at 15:08
EssaidiM
2,957 1 11 16

7 docs showing `push` rather than `save` syntax: git stash push – SherylHohman Apr 25 '18 at 18:04

17 This is the real answer. Unfortunately, there are a ton of old answers above it. – malan Dec 12 '18 at 19:37

1 For more on the newer `git stash push` : stackoverflow.com/a/47231547/6309 – VonC Jan 28 at 15:03

11 2019 newcomers upvote – pttsky Feb 6 at 14:37

89

You can turn a stash into a branch if you feel it's important enough:

```
git stash branch <branchname> [<stash>]
```

from the man page:

This 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, then drops the `<stash>` if that completes successfully. 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

You can later rebase this new branch to some other place that's a descendent of where you were when you stashed.

edited Apr 11 at 8:52
Kasun Siyambalapitiya
1,668 3 21 37

answered Jun 29 '12 at 22:34
Adam Dymitruk
93k 13 125 132

1 Since branches are pretty cheap in git, this suggestion is most useful to me. – Jayan Jan 29 '16 at 1:32

4 Sure, but this doesn't help if you want to keep re-applying this stash in different branches later on, like the OP is asking. You would have to cherry-pick its head. – stewSquared Jan 13 '17 at 21:15

@stewSquared why wouldn't a rebase --onto work? – Gauthier Jul 2 '18 at 6:49

@AdamDymitruk Is there any way to perform this while keeping the stash without poping. (like in `git stash apply`) – Kasun Siyambalapitiya Apr 11 at 7:28

48

If you are just looking for a lightweight way to save some or all of your current working copy changes and then reapply them later at will, consider a patch file:

```
# save your working copy changes
git diff > some.patch

# re-apply it later
git apply some.patch
```

Every now and then I wonder if I should be using stashes for this and then I see things like the insanity above and I'm content with what I'm doing :)

answered Nov 10 '17 at 16:06
Pat Niemeyer
2,157 21 27

3 this is a game changer! I'm done with stashes. – Siddhartha May 29 at 19:42

1 This is it! Thank you. I have also updated my .gitignore to ignore .patch files and I am all set to have as many patches as I want. – LINGS Jun 24 at 14:27

40

Stashes are not meant to be permanent things like you want. You'd probably be better served using tags on commits. Construct the thing you want to stash. Make a commit out of it. Create a tag for that commit. Then roll back your branch to `HEAD^` . Now when you want to reapply that stash you can use `git cherry-pick -n tagname` (`-n` is `--no-commit`).

answered Jun 29 '12 at 21:39
Lily Ballard
155k 23 341 322

1 Definitely like this approach, feels a bit cleaner to just have a `named commit` hanging out somewhere. Only mild annoyance is that it doesn't get committed upon cherry pick and stays in the diff, which means it will need to be manually not checked in during the next commit. – Aditya M P Dec 15 '16 at 3:50

1 This is the closest. I think I'll make some aliases for this. I don't like using the description as a "name". – stewSquared Jan 13 '17 at 21:17

Shame it adds to index and you have to reset, someone should patch a `--no-stage` option! Related: stackoverflow.com/questions/32333383/… – Ciro Santilli 新疆改造中心法轮功六四事件 Sep 19 '18 at 8:16

19

I have these two functions in my `.zshrc` file:

```
function gitstash() {
    git stash push -m "zsh_stash_name_$1"
}

function gitstashapply() {
    git stash apply $(git stash list | grep "zsh_stash_name_$1" | cut -d: -f1)
}
```

Using them this way:

```
gitstash nice

gitstashapply nice
```

edited Jul 9 '18 at 5:59

answered Jun 6 '18 at 18:53
iWheelBuy
2,947 2 24 53

8

Alias

```
sapply = "!f() { git stash apply \"$(git stash list | awk -F: --posix -vpat=\"$*\" \"$ 0 ~ pat {print $ 1; exit}\")\"; }; f"
```

Usage

```
git sapply "<regex>"
```

- compatible with Git for Windows

Edit: I sticked to my original solution, but I see why majority would prefer Etan Reisner's version (above). So just for the record:

```
sapply = "!f() { git stash apply \"$(git stash list | grep -E \"$*\" | awk \"{ print $
1; }\" | sed -n \"s/://;1p\")\"; }; f"
```

edited Dec 22 '13 at 20:56

answered Nov 26 '13 at 22:51
Vlastimil Ovčáčík
1,321 16 25

Using `awk -F: '{print $1}'` would eliminate the need for the sed entirely. Also why wrap this in a function? And using `awk -F: -vpat="$*" '$0 ~ pat {print $1}'` should allow dropping the grep as well. Though might require slightly different quoting for the pattern. – Etan Reisner Nov 27 '13 at 2:36

@EtanReisner: your snippet outputs more than one line. – Vlastimil Ovčáčík Nov 27 '13 at 20:11

Make the action `{print $1; exit}` to quit after the first matched line. – Etan Reisner Nov 27 '13 at 20:50

@EtanReisner: After some testing I could get rid of the sed, but wrapper and grep stays. – Vlastimil Ovčáčík Nov 27 '13 at 23:56

You do not need the grep though like I said the pattern quoting might differ without it. I'm assuming by wrapper you mean the shell function? You never explained why you think you need that so I can't comment on whether you actually do but I believe you quite likely don't. (You might need to manually invoke a shell instead of git stash directly but possibly not even that.) – Etan Reisner Nov 28 '13 at 3:09

8

It's unfortunate that `git stash apply stash^{/<regex>}` doesn't work (it doesn't actually search the stash list, see the comments under the accepted answer).

Here are drop-in replacements that search `git stash list` by regex to find the first (most recent) `stash@{<n>}` and then pass that to `git`

```
# standalone (replace <stash_name> with your regex)
(n=$(git stash list --max-count=1 --grep=<stash_name> | cut -f1 -d":") ; if [[ -n "$n"
]] ; then git stash show "$n" ; else echo "Error: No stash matches" ; return 1 ; fi)
(n=$(git stash list --max-count=1 --grep=<stash_name> | cut -f1 -d":") ; if [[ -n "$n"
]] ; then git stash apply "$n" ; else echo "Error: No stash matches" ; return 1 ; fi)
```

```
# ~/.gitconfig
[alias]
  sshow = "!f() { n=$(git stash list --max-count=1 --grep=$1 | cut -f1 -d":") ; if [[ -n
"$n" ]] ; then git stash show "$n" ; else echo "Error: No stash matches $1" ; return 1 ;
fi }; f"
  sapply = "!f() { n=$(git stash list --max-count=1 --grep=$1 | cut -f1 -d":") ; if [[ -
n "$n" ]] ; then git stash apply "$n" ; else echo "Error: No stash matches $1" ; return
1 ; fi }; f"

# usage:

$ git sshow my_stash
 myfile.txt | 1 +
 1 file changed, 1 insertion(+)

$ git sapply my_stash
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   myfile.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Note that proper result codes are returned so you can use these commands within other scripts. This can be verified after running commands with:

```
echo $?
```

Just be careful about variable expansion exploits because I wasn't sure about the `--grep=$1` portion. It should maybe be `--grep="$1"` but I'm not sure if that would interfere with regex delimiters (I'm open to suggestions).

4,063 3 43 65 3,299 35 67

6

This answer owes much to Klemen Slavič. I would have just commented on the accepted answer but I don't have enough rep yet :(

You could also add a git alias to find the stash ref and use it in other aliases for show, apply, drop, etc.

```
[alias]
    sgrep = "!f() { ref=$(git --no-pager stash list | grep "$1" | cut -d: -f1 | head -
n1); echo ${ref:-<no_match>}; }; f"
    sshow = "!f() { git stash show $(git sgrep "$1") -p; }; f"
    sapply = "!f() { git stash apply $(git sgrep "$1"); }; f"
    sdrop = "!f() { git stash drop $(git sgrep "$1"); }; f"
```

Note that the reason for the `ref=$( ... ); echo ${ref:-<no_match>};` pattern is so a blank string is not returned which would cause sshow, sapply and sdrop to target the latest stash instead of fail as one would expect.

answered Jan 11 '18 at 1:30
Nathanael
109 1 3

1 This works for me while the accepted answer doesn't seem to work (see my commend on the accepted answer) – Jan Rüegg Jan 11 '18 at 9:29

4

Alias This might be a more direct syntax for Unix-like systems without needing to encapsulate in a function. Add the following to ~/.gitconfig under [alias]

```
sshow = !sh -c 'git stash show stash^{/$*} -p' -
sapply = !sh -c 'git stash apply stash^{/$*}' -
ssave = !sh -c 'git stash save "${1}"' -
```

Usage: sapply *regex*

Example: git sshow MySecretStash

The hyphen at the end says take input from standard input

Rohanthewiz
629 6 7

4

Use a small bash script to look up the number of the stash. Call it "gitapply":

```
NAME="$1"
if [[ -z "$NAME" ]]; then echo "usage: gitapply [name]"; exit; fi
git stash apply $(git stash list | grep "$NAME" | cut -d: -f1)
```

Usage:

```
gitapply foo
```

...where foo is a substring of the name of the stash you want.

answered Feb 19 '18 at 12:56
Will Sheppard
1,894 22 37

3

use `git stash push -m aNameForYourStash` to save it. Then use `git stash list` to learn the **index of the stash** that you want to apply. Then use `git stash pop --index 0` to pop the stash and apply it.

note: I'm using *git version 2.21.0.windows.1*

answered Sep 5 at 13:40
canbax
432 6 13

Your answer is nominally what the top-rated answer would be, taking into account this comment on the current syntax for `git stash {push,save}` – Michael Sep 9 at 12:36

2

```
<#
.SYNOPSIS

Restores (applies) a previously saved stash based on full or partial stash name.

.DESCRIPTION
Restores (applies) a previously saved stash based on full or partial stash name and then
optionally drops the stash. Can be used regardless of whether "git stash save" was done
or just "git stash". If no stash matches a message is given. If multiple stashes match a
message is given along with matching stash info.

.PARAMETER message
A full or partial stash message name (see right side output of "git stash list"). Can
also be "@stash{N}" where N is 0 based stash index.

.PARAMETER drop
If -drop is specified, the matching stash is dropped after being applied.

.EXAMPLE
Restore-Stash "Readme change"
Apply-Stash MyStashName
Apply-Stash MyStashName -drop
Apply-Stash "stash@{0}"
#>
function Restore-Stash  {
    [CmdletBinding()]
    [Alias("Apply-Stash")]
    PARAM (
        [Parameter(Mandatory=$true)] $message,
        [switch]$drop
    )

    $stashId = $null

    if ($message -match "stash@{") {
        $stashId = $message
    }

    if (!$stashId) {
        $matches = git stash list | Where-Object { $_ -match $message }

        if (!$matches) {
            Write-Warning "No stashes found with message matching '$message' - check git
stash list"
            return
        }
```

```
            return $matches
        }

        $parts = $matches -split ':'
        $stashId = $parts[0]
    }

    git stash apply ''$stashId''

    if ($drop) {
        git stash drop ''$stashId''
    }
}
```

More details here

answered Jul 20 '17 at 19:35
Geoffrey Hudik
390 3 8

2

Use `git stash save NAME` to save.

Then... you can use this script to choose which to apply (or pop):

```
#!/usr/bin/env ruby
#git-stash-pick by Dan Rosenstark

# can take a command, default is apply
command = ARGV[0]
command = "apply" if !command
ARGV.clear

stashes = []
stashNames = []
`git stash list`.split("\n").each_with_index { |line, index|
    lineSplit = line.split(": ");
    puts "#{index+1}. #{lineSplit[2]}"
    stashes[index] = lineSplit[0]
    stashNames[index] = lineSplit[2]
    realIndex = input.to_i - 1
    puts "\n\nDoing #{command} to #{stashNames[realIndex]}\n\n"

    puts `git stash #{command} #{stashes[realIndex]}`
end
```

I like being able to see the names of the stashes and choose. Also I use Zshell and frankly didn't know how to use some of the Bash aliases above ;)

Note: As Kevin says, you should use tags and cherry-picks instead.

edited Jul 20 '17 at 20:31
Michael
4,063 3 43 65

answered Jun 8 '17 at 15:56
Dan Rosenstark
43.6k 51 246 390

2

What about this?

```
git stash save stashname
git stash apply stash^{/stashname}
```

answered May 7 at 9:40
AdamB
123 9

1 It sounds like something like that *used to be* the accepted answer, but has since been deleted. – Michael May 8 at 14:09

Hm, then why it was deleted? – AdamB May 9 at 15:16

I don't know, since I did not post the answer and do not have 10,000 reputation, but I presume it has something to do with the comments saying it doesn't work: *It's unfortunate that `git stash apply stash^{/<regex>}` doesn't work (it doesn't actually search the stash list, see the comments under the accepted answer).* – Michael May 9 at 15:19

Late to the party here, but if using VSCode, a quick way to do so is to open the command palette (CTRL / CMD + SHIFT + P) and type "Pop Stash", you'll be able to retrieve your stash by name without the need to use git CLI

376 3 11

1

in my fish shell

```
function gsap
  git stash list | grep ": $argv" | tr -dc '0-9' | xargs git stash apply
end
```

use

gsap name_of_stash

answered Dec 26 '18 at 8:18
Matsumoto Kazuya
154 1 5

1

`git stash apply` also works with other refs than `stash@{0}`. So you can use ordinary **tags** to get a persistent name. This also has the advantage that you cannot accidentaly `git stash drop` or `git stash pop` it.

So you can define an alias `pstash` (aka "persistent stash") like this:

```
git config --global alias.pstash '!f(){ git stash && git tag "$1" stash && git stash
drop; }; f'
```

Now you can create a tagged stash:

```
git pstash x-important-stuff
```

and `show` and `apply` it again as usual:

```
git stash show x-important-stuff
git stash apply x-important-stuff
```

48.8k 11 72 102

1

For everything besides the stash creation, I'd propose another solution by introducing fzf as a dependency. Recommend taking 5 minutes of your time and get introduced to it, as it is over-all great productivity booster.

Anyway, a related excerpt from their examples page offering stash searching. It's very easy to change the sciptlet to add additional functionality (like stash application or dropping):

```
fstash() {
    local out q k sha
    while out=$(
            git stash list --pretty="%C(yellow)%h %>(14)%Cgreen%cr %C(blue)%gs" |
            fzf --ansi --no-sort --query="$q" --print-query \
                --expect=ctrl-d,ctrl-b); do
        mapfile -t out <<< "$out"
        q="${out[0]}"
        k="${out[1]}"
        sha="${out[-1]}"
        sha="${sha%% *}"
        [[ -z "$sha" ]] && continue
        if [[ "$k" == 'ctrl-d' ]]; then
            git diff $sha
        elif [[ "$k" == 'ctrl-b' ]]; then
            git stash branch "stash-$sha" $sha
            break;
        else
            git stash show -p $sha
        fi
    done
}
```

edited Jun 14 at 9:09

answered Sep 10 '16 at 11:58
laur
146 12