Dealing With a Scenario

Look into a scenario when one would need to stash away changes for any reason.

We'll cover the following

Scenario

Here is a basic example of a change I want to “stash”:

```
mkdir lgthw_git_stash_1
cd lgthw_git_stash_1
git init
echo 'Some content' > file1
git add file1
git commit -am initial
echo 'Some changes I am not sure about' >> file1
```

Terminal 1

Terminal

Now imagine I'm in the middle of some work, and Alice lets me know that there's an important update to the code that I need to pull from BitBucket.

First, you can see what changes you have made locally with `git diff`:

```
8   git diff
```

Stashing changes

To store away these changes locally, run `git stash`. Run the following three commands in the terminal given below.

```
9   git stash
```

A quick `git status` confirms that your working directory is "clean":

```
10  git status
```

What happened to your change? If you are interested, you can run the following command to view the graph:

```
11  git log --graph --all --decorate
```

Terminal 1

Terminal

As you can see, `git stash` has committed the state of the index and then committed the local change to the `refs/stash` branch and merged them as

a child of the `HEAD` on a new `refs/stash` branch.

Don’t worry too much about the details yet; it has basically stored all the changes you’ve made (but not *committed*), ready to be re-applied.

The `stash` branch is a special one that is kept local to your repository. The “commit” message `WIP on master` and `index on master` is added automatically for you.

The `master` branch is still where it was before you stashed, and the `HEAD` pointer is pointed at the `master` branch.

I can now do my other work (in this case, pulling the latest changes from a remote. Remember, we will cover remotes in a later section) without concern for whether it conflicts with those changes.

Retrieving stash list

You can see your “stash list” by running this command:

```
12  git stash list
```

Terminal 1

Terminal

Popping stashed work

Once ready, you can re-apply those changes on the same codebase by running `git stash pop`:

```
13  git stash pop
```

It “pops” the zero-numbered change off the stash stack and restores the changes I stashed, applied to wherever I’ve ended up.

What's a Stack?

A stack is a computer science concept of a series of items stored in such a way that you can "push" an item to the top of it for any number of times and then "pop" an item off the top until it's empty. When you pop, you only get the most recent item that was pushed on. This means that when you pop your Git stash, you get the most recent one you pushed (which is likely to be what you wanted).

Back

Introduction: Git Stash

Next

Choosing Your Stash

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/dealing-with-a-scenario__git-stash__learn-git-the-hard-way)