GIT: From Beginner To Fearless

GIT Stash Activity:
Sometimes you just don't want to commit and you need to get the changes back

Brian Gorman, Author/Instructor/Trainer

Introduction

I have a background that includes too many years in .Net to count. My first version control system was Visual Source Safe 6, which, if any of you have used or encountered, left more than a little bit to be desired.

Fast forward to 2012-2015 when I was using Team Foundation Server and Visual Studio Team System (now Azure DevOps or TeamCity or something like that) for professional development (Git/BitBucket for personal). Anyway, one of my favorite features to use in this centralized source-control system was a feature called "Shelving." Shelving was great because I could put my changes "on the shelf" and basically reset the repository back to the last commit. Anytime I wanted to, I could then pull my changes "off the shelf" and start working with them again.

As such, I was super excited to learn about a feature in Git called "Stashing." My first reaction was "I can't wait to learn about that, because I'm going to use it all the time. It will be the most important feature that Git has to offer, I have no doubt.

Ok, so guess what. It's not. It's great, but it's mostly unnecessary if you don't want to use it. The simple fact of the matter is that creating a branch in Git is so inexpensive and easy to do, that it is easier to just check out and commit my changes to a new branch than to put them "on the shelf" with a stash command.

Add into that – using stash out of the box is a bit counter-intuitive, and without proper planning can quickly become pretty useless, confusing, and un-memorable.

There is one feature that I have found, however, that is perfect for stashing. The feature is when I want to repeatedly do something that I'm not going to check in and I want to be able to do that on any branch, at any time. Also, it's a change, not a command, obviously.

In this activity, we'll go over that one scenario while getting familiar with how the stash command works.

Let's gets to stashing!

Step 1: Start with a copy of my StashActivity repository [has the web.config file, etc]

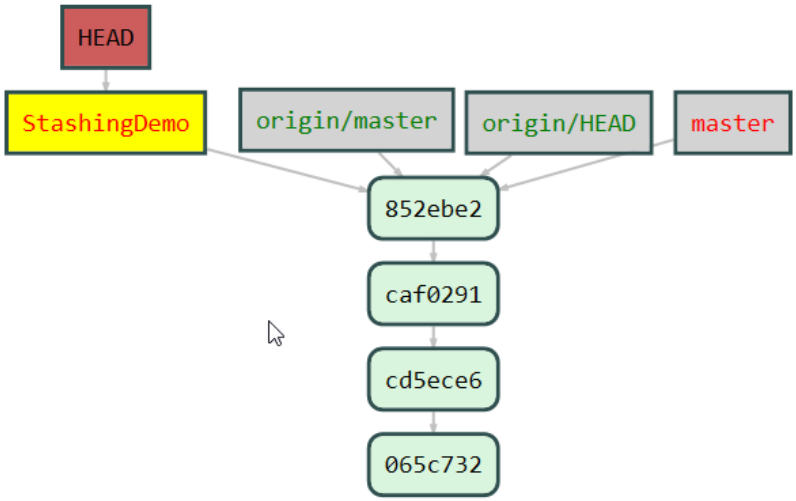
a) Start with a copy of the repo and create a feature branch.

First fork [then clone local] my repo [it has the coveted web.config file in it]
If you don't want to fork go to the link and download the files and use them to create your own repo:
[git clone https://github.com/majorguidancesolutions/GFBTFStashingActivity.git <folder>]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder
$ git clone https://github.com/majorguidancesolutions/GFBTFStashingActivity.git
GitStashingActivity
Cloning into 'GitStashingActivity'...
remote: Counting objects: 14, done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 14 (delta 3), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (14/14), done.
```

[git checkout –b feature-branch]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (master)
$ git checkout -b StashingDemo
Switched to a new branch 'StashingDemo'
```



Nothing spectacular here. The main thing is we have that web.config to play with, and some other files as well.

Notes

b) Modify the web.config file to use your local db settings.

In case you are not a .net developer, the web.config file is traditionally a simple configuration file that sets up the project and configurable variables [like connection string info] for .Net projects. We're going to simulate this file [not an actual one, and no real db connection, etc]. Our main goal will be to stash our local connection string info while only have the test version of the file in the repository.

[code web.config]
[change the connection string settings from test to local

```
<add name="AppContext"
         connectionString="Server=local_server;Database=dbApp_Local;
                                User
Id=local_user;Password=local_user_pwd;"
         providerName="System.Data.SqlClient"/>
    <add name="AuthContext"

connectionString="Server=local_auth_server;Database=dbAuth_Local;
                                User
Id=local_auth_user;Password=local_auth_user_pwd;"
         providerName="System.Data.SqlClient"/>
```

[git status]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (Stashin
o)
$ git status
On branch StashingDemo
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:   web.config

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

Instead of committing, we are going to stash these changes. Doing this will add it to the stash and reset the repo.

Step 2: Simple Stashing

Now that we have the changes we want to use multiple times in our repository across branches, we stash the change:

a) Add the change to stash

[git stash]

[git status]

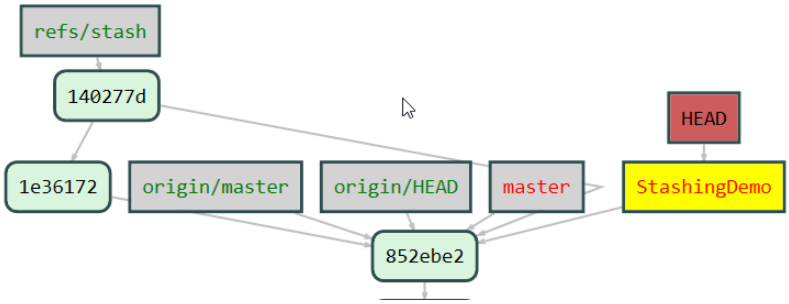
```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
o)
$ git stash
Saved working directory and index state WIP on StashingDemo: 852ebe2 Create read
me.txt

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
o)
$ git status
On branch StashingDemo
nothing to commit, working tree clean
```

b) List the stash

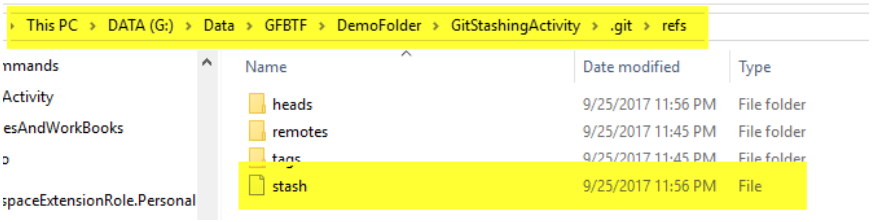
[git stash list]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
o)
$ git stash list
stash@{0}: WIP on StashingDemo: 852ebe2 Create readme.txt
```

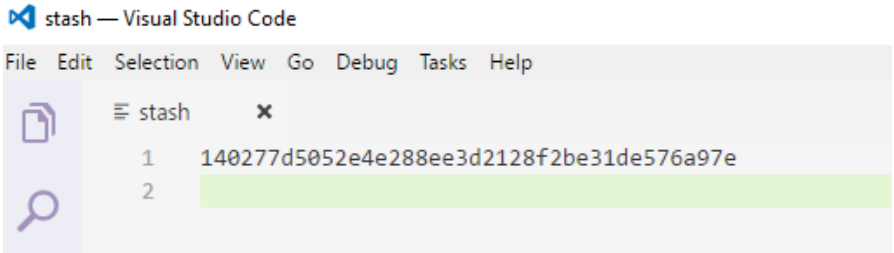
So that isn't very useful. Also, look at the state of our repo



There is some kind of weird thing for refs/stash. What is that?



Open in code:



So there is a commit id in that file. It essentially gives us a pointer to the changes we made on that stash.

Note back up to the list, that the stash points to the main commit.

c) Get changes back and remove from stash

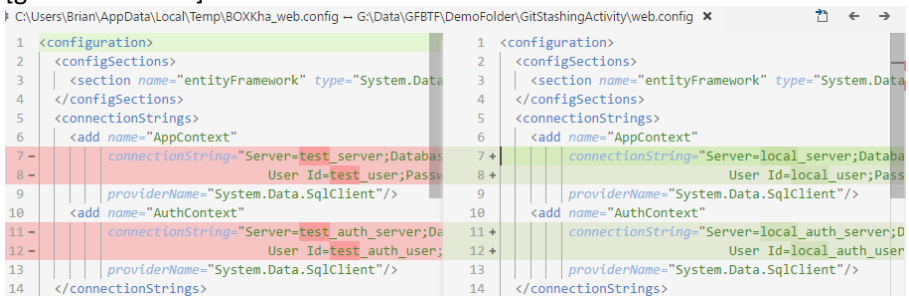
[git stash pop]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
o)
$ git stash pop
On branch StashingDemo
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:   web.config

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (140277d5052e4e288ee3d2128f2be31de576a97e)

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
o)
$ git stash list

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
```

So popping removes from the stash and applies to local (would have to resolve any conflicts of course).

Compare to head to see that changes are as expected:

[git difftool head]



Perfect.

Re-add the stash:

[git stash save web-config-changes]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
o)
$ git stash save web-config-changes
Saved working directory and index state On StashingDemo: web-config-changes

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
o)
$ git stash list
stash@{0}: On StashingDemo: web-config-changes

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
```

So naming is possible – we can then know which one to get later when we want to put the changes back in place.

d) Get Changes back and leave in stash

[git stash apply]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
o)
$ git stash apply
On branch StashingDemo
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:   web.config

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

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
o)
$ git stash list
stash@{0}: On StashingDemo: web-config-changes
```

So getting our config changes back and leaving the changes in stash is possible.

Step 3: Advanced stash operations

We've seen one simple stash, and know how to pop and apply, as well as list. But what happens when there are multiple stash entries and we want to put just one in place? What about untracked files?

a) Stash all changes, including untracked files.

[git reset –hard head]

[touch aNewFile.txt]

[code info.txt]

[git status]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDe
o)
$ git status
On branch StashingDemo
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:   info.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        aNewFile.txt

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

[git stash save -u "A new file and modified info.txt"]

[git status]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
o)
$ git stash save -u "A new file and modified info.txt"
Saved working directory and index state On StashingDemo: A new file and modified
 info.txt

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
o)
$ git status
On branch StashingDemo
nothing to commit, working tree clean
```

[git stash list]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
o)
$ git stash list
stash@{0}: On StashingDemo: A new file and modified info.txt
stash@{1}: On StashingDemo: web-config-changes
```

I wish I could just apply from the names I gave, but it's not that easy. Instead, I have to use the refs.

[git stash apply stash@{1}]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
o)
$ git stash apply stash@{1}
On branch StashingDemo
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:   web.config

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

I could re-add to stash and get another entry that is the same as stash@{1}, but that would be pointless. Also note that this is like a stack – Last in = first out. So stash@{0} is always the top of the 'stack', and would be what is popped. As you add more to the stash, the original entries get pushed farther down. This means stash@{1} won't necessarily always be my web.config, which is another important reason to name them.

b) Checking out a branch from stash:

[git reset –hard head]

[git status]

[git stash list]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
o)
$ git reset --hard head
HEAD is now at 852ebe2 Create readme.txt

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
o)
$ git status
On branch StashingDemo
nothing to commit, working tree clean

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
o)
$ git stash list
stash@{0}: On StashingDemo: A new file and modified info.txt
stash@{1}: On StashingDemo: web-config-changes

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
```

[git stash branch feature-changes]

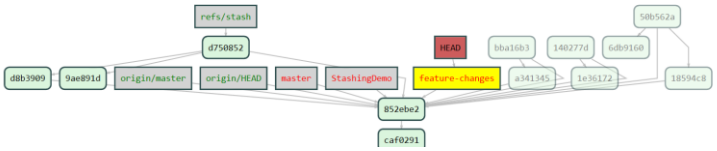
```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (StashingDem
o)
$ git stash branch feature-changes
Switched to a new branch 'feature-changes'
On branch feature-changes
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   info.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        aNewFile.txt

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (50b562a862d884a27b13df8fc5e9e1d30348360e)

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (feature-cha
nges)
$ git stash list
stash@{0}: On StashingDemo: web-config-changes
```

Notice that it took the first stash only by default. The branch also changed.

[git stash save -u “A new file and modified info.txt”]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (feature-cha
nges)
$ git stash save -u "A New file and modified info.txt"
Saved working directory and index state On feature-changes: A New file and modif
ied info.txt

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (feature-cha
```

Our repo is getting pretty messy with all this stashing going on…



[git stash list]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (feature-cha
nges)
$ git stash list
stash@{0}: On feature-changes: A New file and modified info.txt
stash@{1}: On StashingDemo: web-config-changes
```

c) Removing from stash without applying or popping

[git stash drop stash@{0}]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivi
nges)
$ git stash drop stash@{0}
Dropped stash@{0} (d750852437c510e65c0668c639c8ea415cda2a88)
```

Keep our changes by applying them, then clearing stash

[git stash apply stash@{0}]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (feature-cha
nges)
$ git stash apply stash@{0}
On branch feature-changes
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:   web.config

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

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (feature-cha
nges)
$ git stash list
stash@{0}: On StashingDemo: web-config-changes
```

[git stash clear]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (feature-cha
nges)
$ git stash clear

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (feature-cha
nges)
$ git stash list
```

[git stash save web-config-changes]

```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (feature-cha
nges)
$ git stash save web-config-changes
Saved working directory and index state On feature-changes: web-config-changes

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/GitStashingActivity (feature-cha
nges)
$ git stash list
stash@{0}: On feature-changes: web-config-changes
```

This concludes our stashing activity.

Closing Thoughts

In this activity we saw how it is possible to stash changes and get them back at a later time.

We also learned how we can save the stash with a name to help us remember what is stashed, and how to apply them while keeping the stash intact.

Stashing is useful for small changesets that need to be applied across multiple branches repeatedly. If the changes are fairly major and/or are specific to a branch, I would recommend just committing the changes to a branch rather than working with stash.

Also, some really good news for visual studio users is that VS2019 now includes stashing as part of the Git integration tools in VS out of the box. This is a great improvement over VS2017, which did not have stashing built-in.

Take a few minutes to make some notes about the various commands we've learned about in this activity, and practice using them.

Notes