GIT: From Beginner To Fearless

GIT Reset and Clean Activity: Reset changes and clean up your repo/working directory

Brian Gorman, Author/Instructor/Trainer

Introduction

The command [git reset] is a command that can be used to reset your current repository to the state it was in at any particular commit. The commit you may be targeting is the last commit [throwing away your current changes, for example]. The commit you may be targeting could also be a few commits down the chain in history. Going back in history can be somewhat dangerous however, so it is critical to use caution when resetting back to a previous commit that is deep in history. Additionally, if the commit chain is public, and other developers rely on this commit history, then you should probably try to find another way to "reset" your code, unless it is simply unavoidable.

In this activity, we're going to look at different types of resets and some scenarios where we would want to use a soft reset [fairly harmless] to a hard reset [red flag: can be very dangerous]. Reset is one of the few commands that gives us the ability to really wreck our repository, but it should still not induce panic and fear. Always remember, if you are scared to do something, you can fork a repo and try it there, with no risk of your changes causing the main repo to become corrupt.

The [git clean] command gives us a lot of power, and can be used to recursively wipe out files and folders. For that reason you may want to do a dry run or practice your command on a secondary copy of the repository in order to avoid problems. The clean command gives a lot of options, so doing research and running dry runs before performing the actual clean may be your best friends when it comes time to do some cleanup.

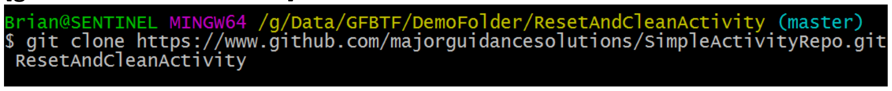
Let's gets started!

Step 1: Resetting your current branch to the most recent [HEAD] commit:

a) Start with any repo, make sure you have the latest in master, and create a feature branch.

First clone the repo if it doesn't exist:

[git clone <link> <folder>]



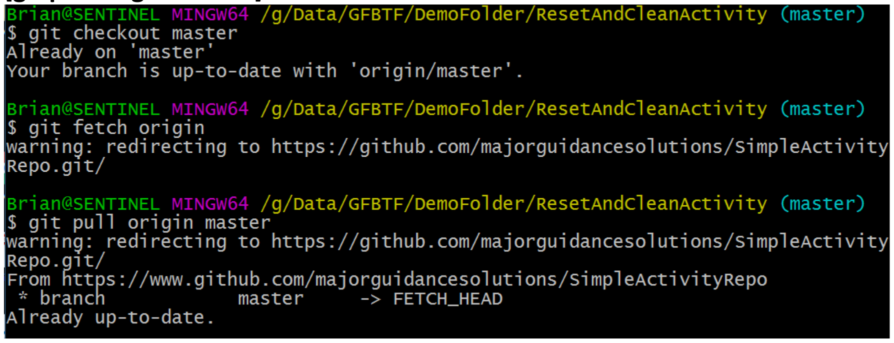
```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActivity (master)
$ git clone https://www.github.com/majorguidancesolutions/SimpleActivityRepo.git
 ResetAndCleanActivity
```

If you didn't clone, make sure master is up to date

[**git checkout master**]

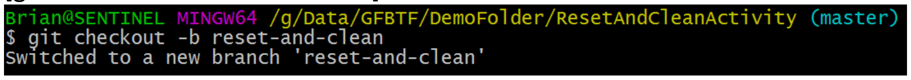
[**git fetch origin**]

[**git pull origin master**]



```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActivity (master)
$ git checkout master
Already on 'master'
Your branch is up-to-date with 'origin/master'.

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActivity (master)
$ git fetch origin
warning: redirecting to https://github.com/majorguidancesolutions/SimpleActivity
Repo.git/

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActivity (master)
$ git pull origin master
warning: redirecting to https://github.com/majorguidancesolutions/SimpleActivity
Repo.git/
From https://www.github.com/majorguidancesolutions/SimpleActivityRepo
 * branch            master     -> FETCH_HEAD
Already up-to-date.
```

[git checkout –b reset-and-clean]

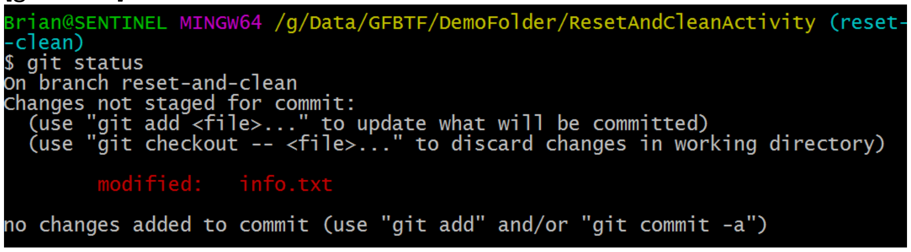


```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActivity (master)
$ git checkout -b reset-and-clean
Switched to a new branch 'reset-and-clean'
```

b) Make some changes and then perform a soft reset

[code info.txt]

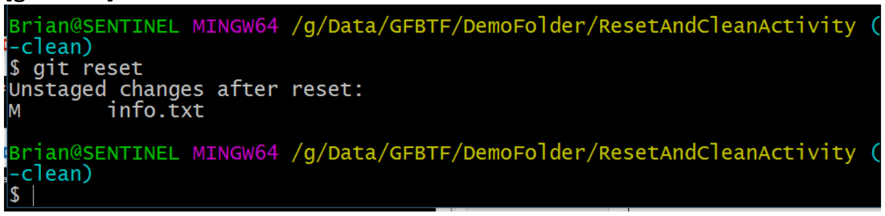
[git status]



```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActivity (reset-
-clean)
$ git status
On branch reset-and-clean
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

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

Notes

Now reset the state of the repo:
[git reset]

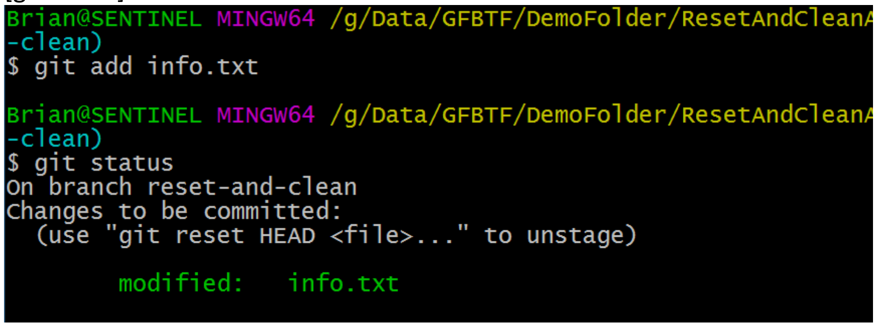


```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActivity (
-clean)
$ git reset
Unstaged changes after reset:
M       info.txt

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActivity (
-clean)
$ |
```

Nothing happens. So [reset] didn't actually remove the changes. That is good to know. So what does reset do in this case? Nothing. If the changes are staged for commit, then something would have happened.

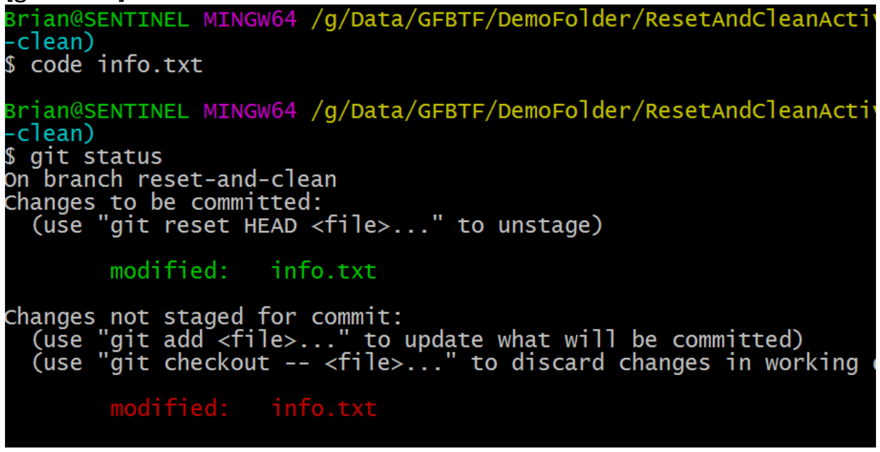
c) Stage a change, make another change, perform a soft reset.

[git add info.txt]
[git status]



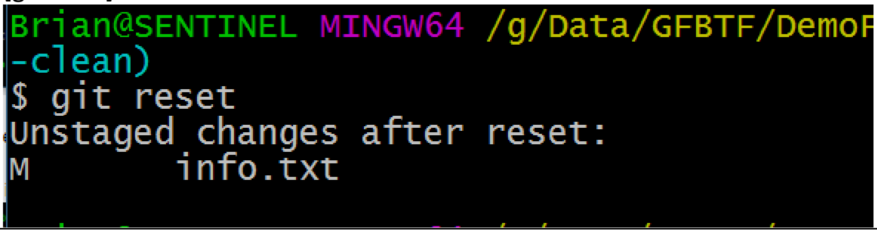
```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanA
-clean)
$ git add info.txt

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanA
-clean)
$ git status
On branch reset-and-clean
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   info.txt
```

[code info.txt]
[git status]



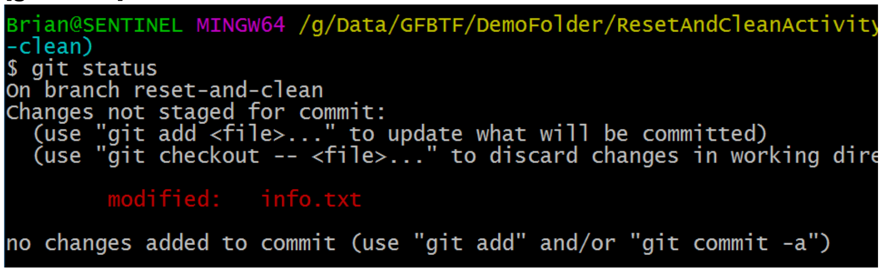
```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActi
-clean)
$ code info.txt

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActi
-clean)
$ git status
On branch reset-and-clean
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   info.txt

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

        modified:   info.txt
```

Now perform the reset
[git reset]



```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoF
-clean)
$ git reset
Unstaged changes after reset:
M       info.txt
```

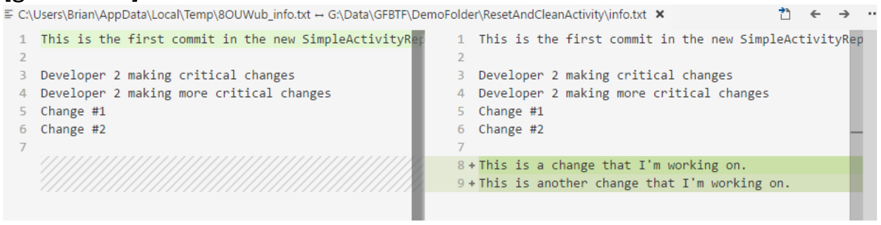
[git status]



```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActivity
-clean)
$ git status
On branch reset-and-clean
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working dire

        modified:   info.txt

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

Check the file. Changes are still there, but now there is nothing that is staged for commit.

[git difftool]



```
C:\Users\Brian\AppData\Local\Temp\8OUWub_info.txt ↔ G:\Data\GFBTF\DemoFolder\ResetAndCleanActivity\info.txt
```

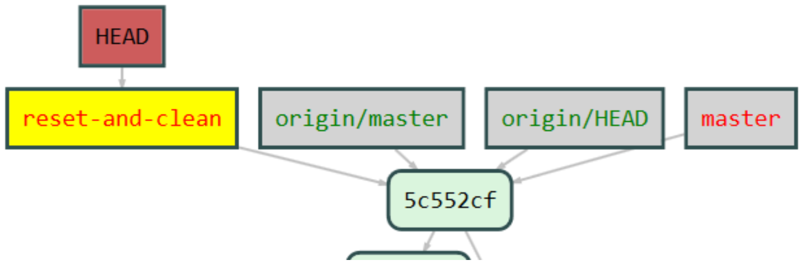
```
This is the first commit in the new SimpleActivityRep

Developer 2 making critical changes
Developer 2 making more critical changes
Change #1
Change #2
```

```
This is the first commit in the new SimpleActivityRep

Developer 2 making critical changes
Developer 2 making more critical changes
Change #1
Change #2

+ This is a change that I'm working on.
+ This is another change that I'm working on.
```

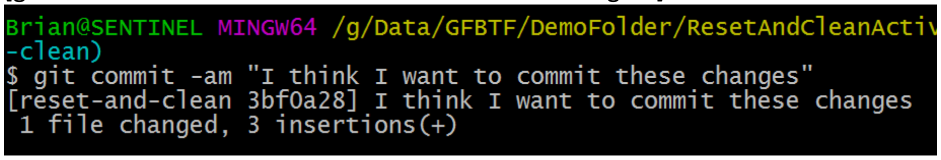
Step 2: Resetting to a previous commit:

Sometimes we commit our changes and then decide we don't want the commit anymore. Let's take a look at how we might be able to do that:

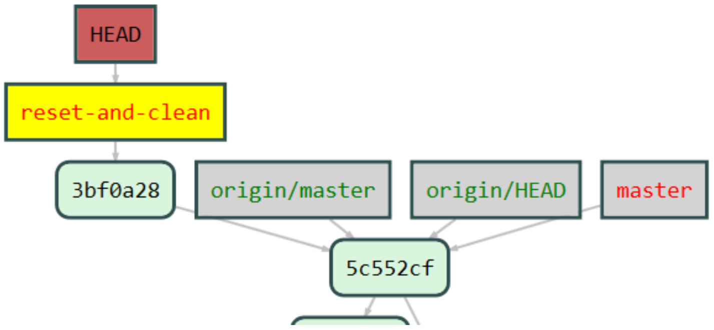
a) Commit the previous changes



[git commit -am "I think I want to commit these changes"]



```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActiv
-clean)
$ git commit -am "I think I want to commit these changes"
[reset-and-clean 3bf0a28] I think I want to commit these changes
 1 file changed, 3 insertions(+)
```



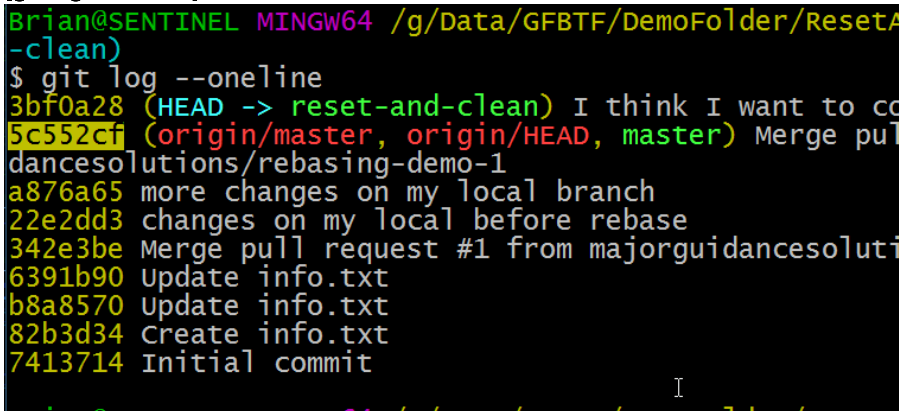
b) Reset back to the previous commit

Now we decide that we don't want to have that commit after all. What do we do?

We can reset back to the previous commit [5c552cf in this case]

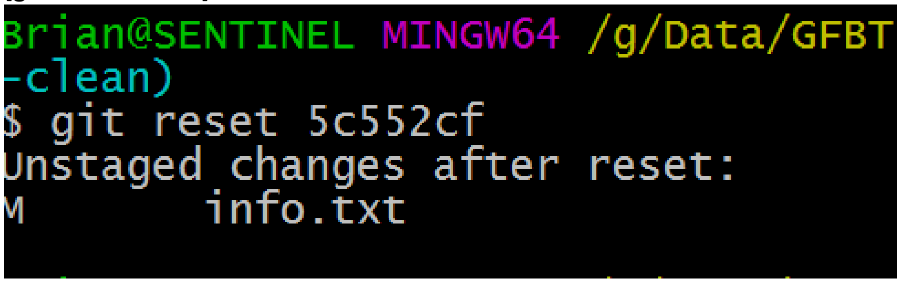
Using git log we can see the commit history as well [if not using GitVis]

[git log --oneline]



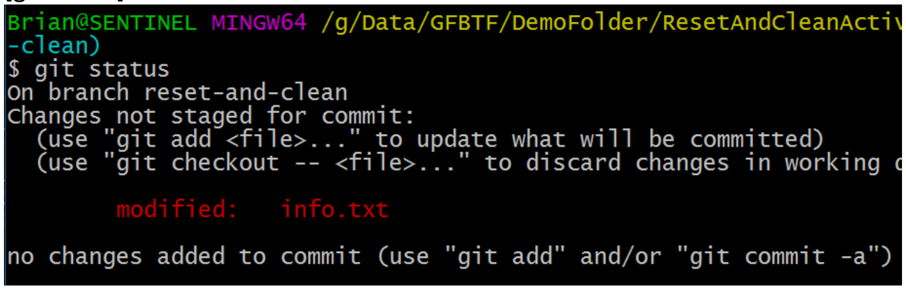
```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetA
-clean)
$ git log --oneline
3bf0a28 (HEAD -> reset-and-clean) I think I want to co
5c552cf (origin/master, origin/HEAD, master) Merge pul
dancesolutions/rebasing-demo-1
a876a65 more changes on my local branch
22e2dd3 changes on my local before rebase
342e3be Merge pull request #1 from majorguidancesoluti
6391b90 Update info.txt
b8a8570 Update info.txt
82b3d34 Create info.txt
7413714 Initial commit
```

[git reset 5c552cf]

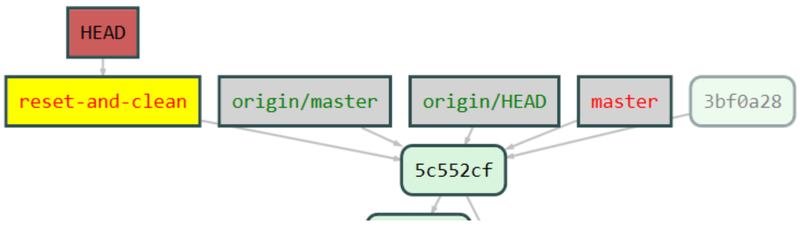


```
Brian@SENTINEL MINGW64 /g/Data/GFBT
-clean)
$ git reset 5c552cf
Unstaged changes after reset:
M       info.txt
```

[git status]



```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActiv
-clean)
$ git status
On branch reset-and-clean
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working d

        modified:   info.txt

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



c) Cleanup the unreachable commit

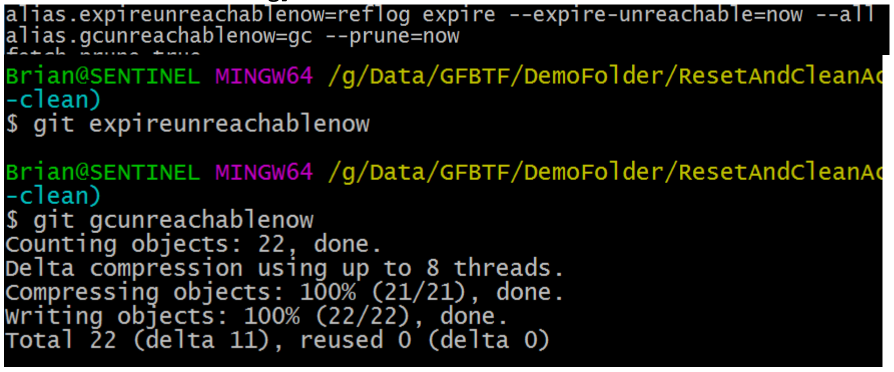
To clean up the commits we just need to make sure we have the reflog set to expire our commits and then run the garbage collector. I have these commands aliased, but in case you don't and you want to run these [or want the commands for later reference], here they are:

[**git reflog expire –expire-unreachable=now –all**]

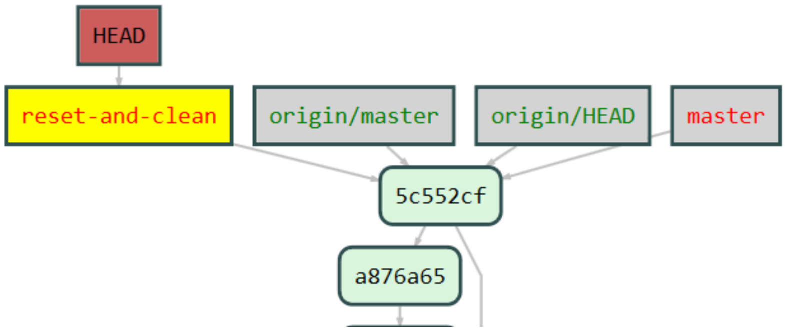
And

[**git gc –prune=now**]

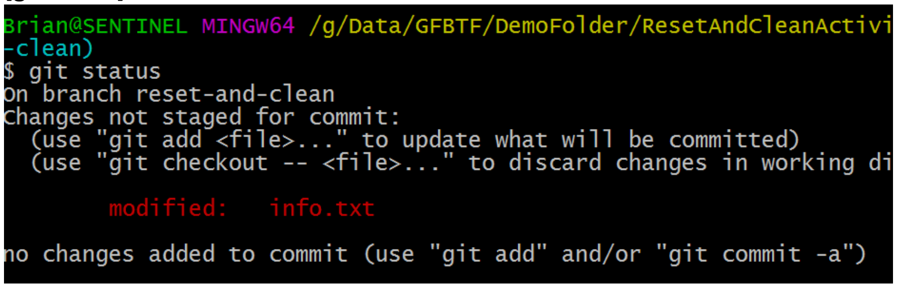
And here are my aliases in my global config [check out the aliasing activity for more info about aliasing]:



```
alias.expireunreachablenow=reflog expire --expire-unreachable=now --all
alias.gcunreachablenow=gc --prune=now
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanAc
-clean)
$ git expireunreachablenow

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanAc
-clean)
$ git gcunreachablenow
Counting objects: 22, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (21/21), done.
Writing objects: 100% (22/22), done.
Total 22 (delta 11), reused 0 (delta 0)
```



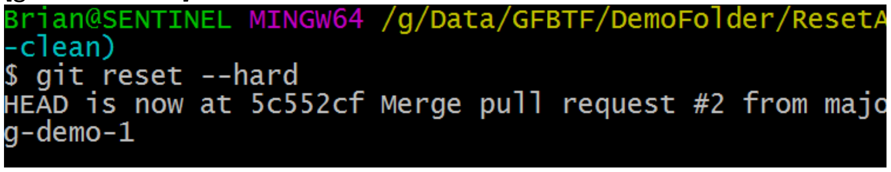
Even though we moved back to a previous commit, we didn't lose the changes that were in the commit that we were on
[git status]



```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActivi
-clean)
$ git status
On branch reset-and-clean
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working di

        modified:   info.txt

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

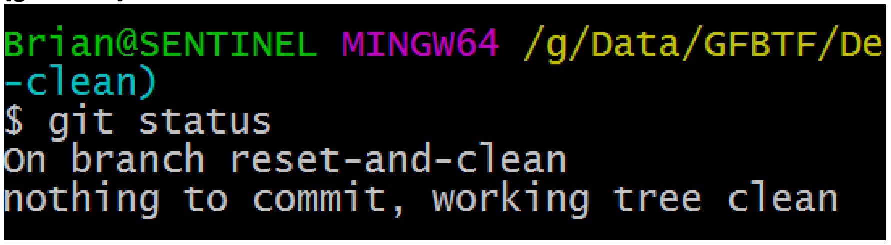
Step 3: Performing a hard reset:

a) Reset back to HEAD with a hard reset. This will remove any changes, so we only want to do this when we are sure that we don't mind losing changes.
[git reset --hard]



```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetA
-clean)
$ git reset --hard
HEAD is now at 5c552cf Merge pull request #2 from majo
g-demo-1
```

[git status]



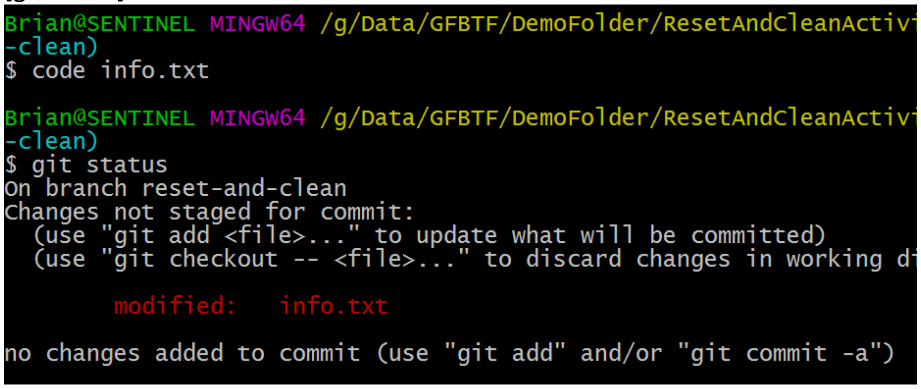
```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/De
-clean)
$ git status
On branch reset-and-clean
nothing to commit, working tree clean
```

Step 4: Performing a hard reset when untracked files are present:

When all of the files are tracked, performing a hard reset will not remove the untracked files. Sometimes we want that to happen as well.

a) Make some changes to info.txt
Before you begin, if you are on a repo that has any important files, you might want to do a quick backup. Our clean operation we are about to run is going to wipe the slate for untracked files

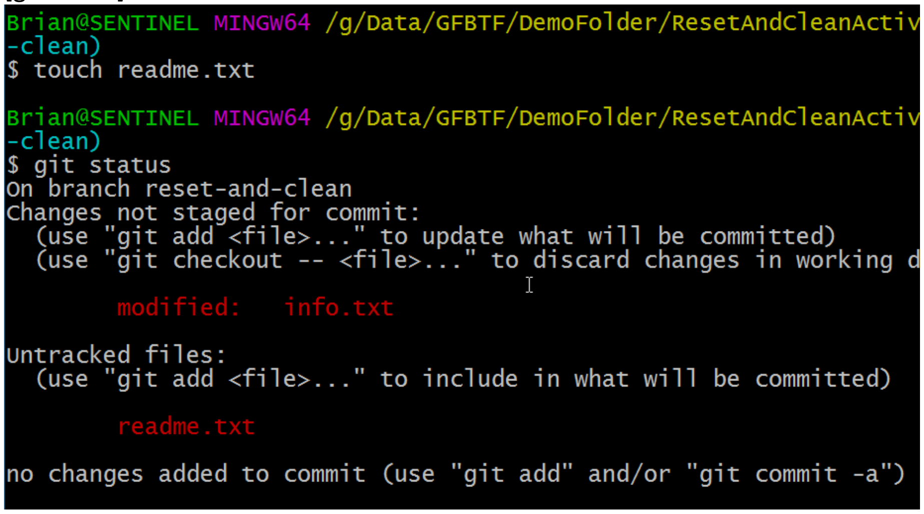
[code info.txt]
[git status]



```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActivi
-clean)
$ code info.txt

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActivi
-clean)
$ git status
On branch reset-and-clean
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working di

        modified:   info.txt

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

a) Add a second text file that is going to remain untracked

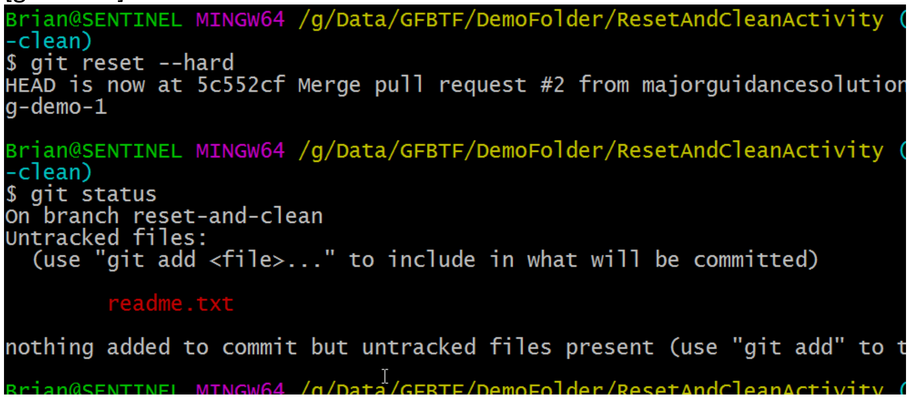
[touch readme.txt]
[git status]



```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActiv
-clean)
$ touch readme.txt

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActiv
-clean)
$ git status
On branch reset-and-clean
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working d

        modified:   info.txt

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

        readme.txt

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

b) Perform a hard reset back to HEAD

[git reset –hard]
[git status]



```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActivity (
-clean)
$ git reset --hard
HEAD is now at 5c552cf Merge pull request #2 from majorguidancesolution
g-demo-1

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActivity (
-clean)
$ git status
On branch reset-and-clean
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        readme.txt

nothing added to commit but untracked files present (use "git add" to t

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActivity (
```

The readme.txt file is still there. We need to run a clean to get rid of it.

Step 5: Cleaning our repo with [git clean]:

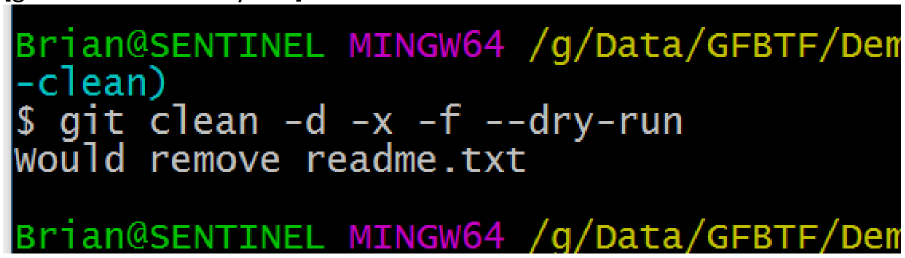
Sometimes there are files that end up in our repo that we don't want anymore. Other times we create a file and don't want it anymore. In either case, we need an easy way to clean up our directory to get our working directory to line up with what the repo says it should have at the latest commit.

a) Perform a full clean

one last warning. Issuing this command will wipe out all files that aren't in the repo from this folder and in it's subfolders.

Since the clean command is destructive, let's do a dry-run to make sure it won't hurt us too badly:

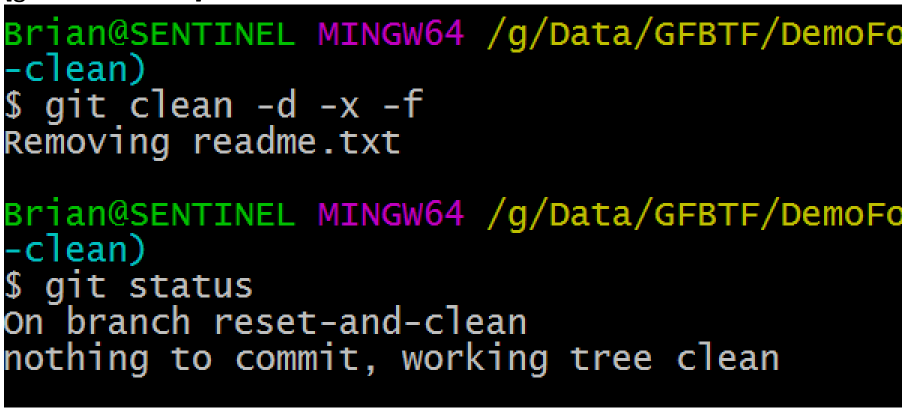
[git clean -d -x -f --dry-run]



```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/Dem
-clean)
$ git clean -d -x -f --dry-run
Would remove readme.txt

Brian@SENTINEL MINGW64 /g/Data/GFBTF/Dem
```

That looks ok to me. Let's do it:

[git clean -d -x -f]



```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFo
-clean)
$ git clean -d -x -f
Removing readme.txt

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFo
-clean)
$ git status
On branch reset-and-clean
nothing to commit, working tree clean
```

So this points out that we can use clean anytime (not just after a reset) to just get our files and folders cleaned up on our local repository.

b) Perform an interactive clean

We've seen that the clean command can be destructive. What if the dry-run command above had listed one file that we wanted to keep? In that case we couldn't have used the -f option. For this reason, there is an interative option.

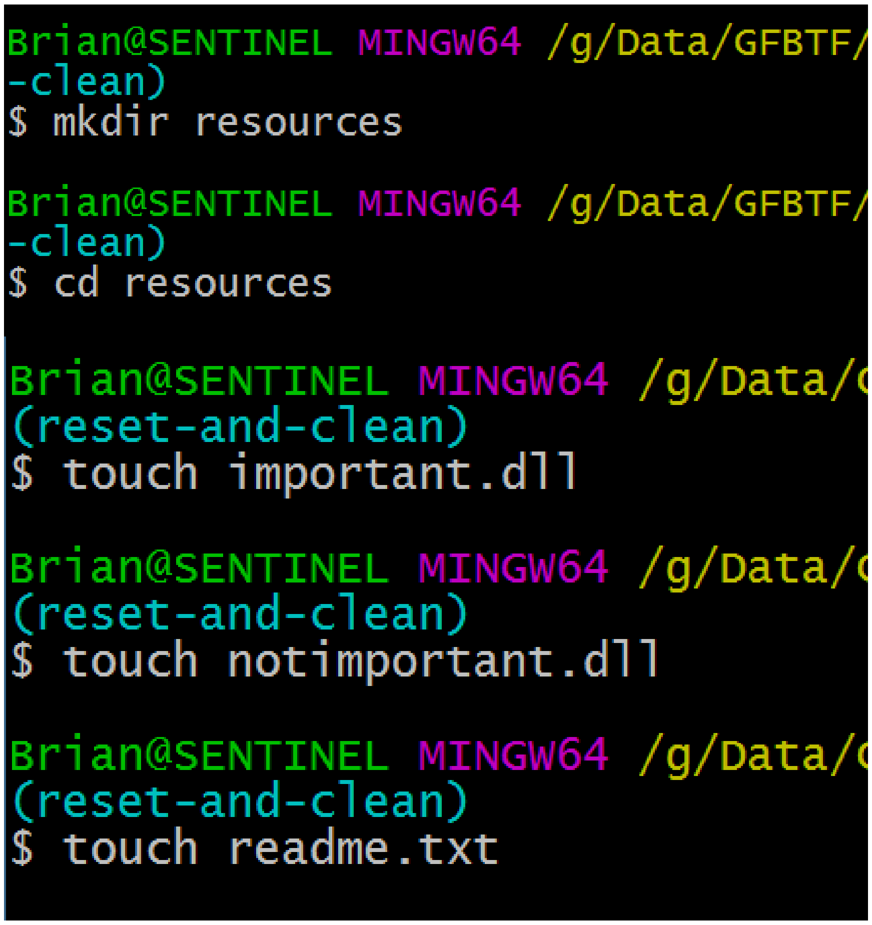
[mkdir resources]

[cd resources]

[touch important.dll]

[touch notimportant.dll]

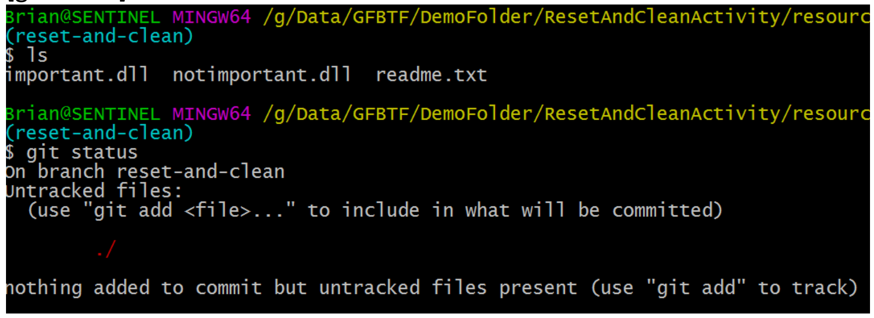
[touch readme.txt]



```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/
-clean)
$ mkdir resources

Brian@SENTINEL MINGW64 /g/Data/GFBTF/
-clean)
$ cd resources

Brian@SENTINEL MINGW64 /g/Data/(
(reset-and-clean)
$ touch important.dll

Brian@SENTINEL MINGW64 /g/Data/(
(reset-and-clean)
$ touch notimportant.dll

Brian@SENTINEL MINGW64 /g/Data/(
(reset-and-clean)
$ touch readme.txt
```

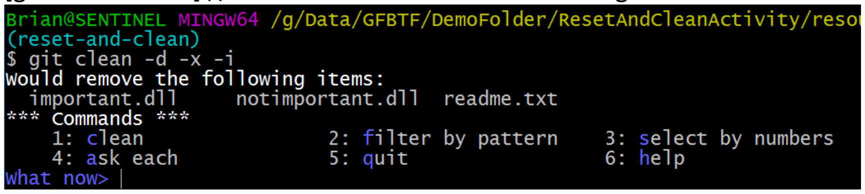
[ls]

[git status]



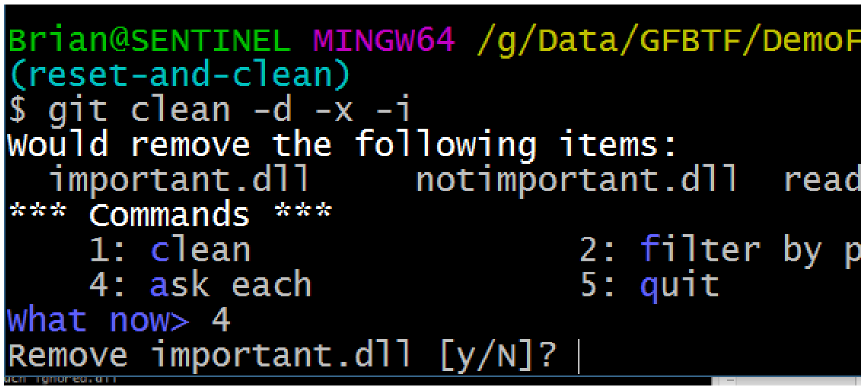
```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActivity/resourc
(reset-and-clean)
$ ls
important.dll  notimportant.dll  readme.txt

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActivity/resourc
(reset-and-clean)
$ git status
On branch reset-and-clean
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        ./

nothing added to commit but untracked files present (use "git add" to track)
```

[git clean -d -x -i] // i => interactive x => cleans even ignored files



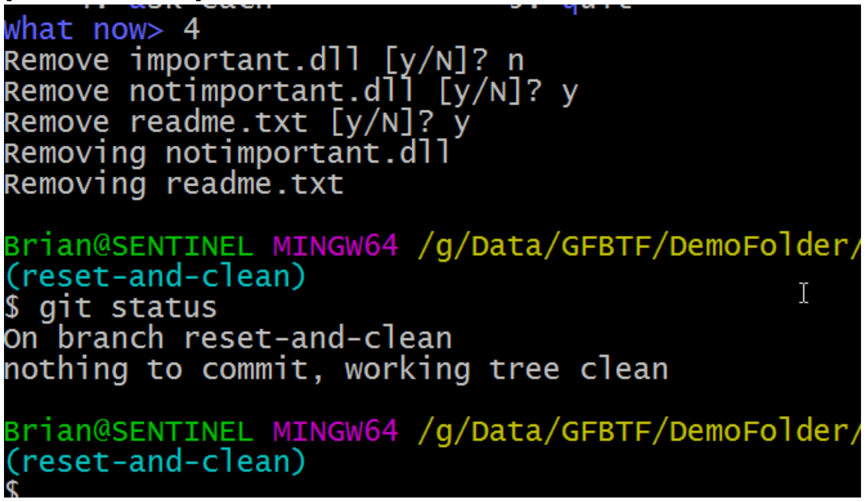
```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/ResetAndCleanActivity/reso
(reset-and-clean)
$ git clean -d -x -i
Would remove the following items:
  important.dll     notimportant.dll  readme.txt
*** Commands ***
    1: clean                2: filter by pattern    3: select by numbers
    4: ask each             5: quit                 6: help
What now> |
```

[choose option 4] //you can play with the others if you want



```
Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoF
(reset-and-clean)
$ git clean -d -x -i
Would remove the following items:
  important.dll      notimportant.dll   read
*** Commands ***
    1: clean                    2: filter by p
    4: ask each                 5: quit
What now> 4
Remove important.dll [y/N]?
```

[keep important.dll]
[remove the rest]



```
What now> 4
Remove important.dll [y/N]? n
Remove notimportant.dll [y/N]? y
Remove readme.txt [y/N]? y
Removing notimportant.dll
Removing readme.txt

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/
(reset-and-clean)
$ git status
On branch reset-and-clean
nothing to commit, working tree clean

Brian@SENTINEL MINGW64 /g/Data/GFBTF/DemoFolder/
(reset-and-clean)
$
```

This concludes our reset and clean activity.

Closing Thoughts

In this activity, we have seen how we can use the fairly safe "soft reset" in order to reset our repository back to the state it was in at the last commit without losing any changes.

We've also seen how we can reset back to any other commit in our history. It's very important to remember, however, that anytime there is a history re-write capability, we should be very careful not to do this against a commit history that is public.

We wrapped up the activity with a look at using the [git clean] command to clear out our working directory of files that are untracked, which can happen on a hard reset when we've added files, or a build added files, etc.

It is equally important to remember that a git clean operation is destructive to our working directory, so we examined the --dry-run capability, as well as doing an interactive clean with the –i flag.

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

Notes

____

____

____

____

____

____

____

____

____

____

____

____