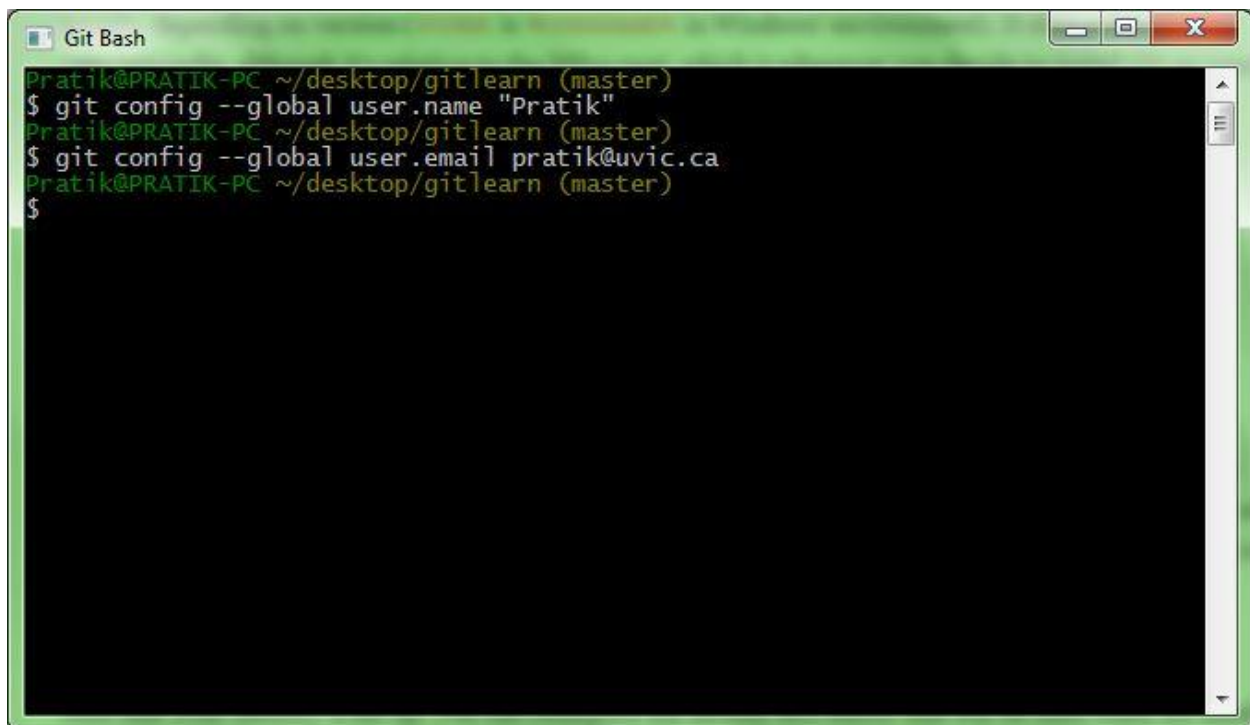


LAB MANUAL GIT

BY PRATIK JAIN

STARTING WITH GIT :

To start with git bash, we need to authenticate ourselves so that for every commit, git can use this information.



```
Git Bash
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git config --global user.name "Pratik"
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git config --global user.email pratik@uvic.ca
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$
```

We can remove -- global option to override this information for some other projects.

GIT EDITOR

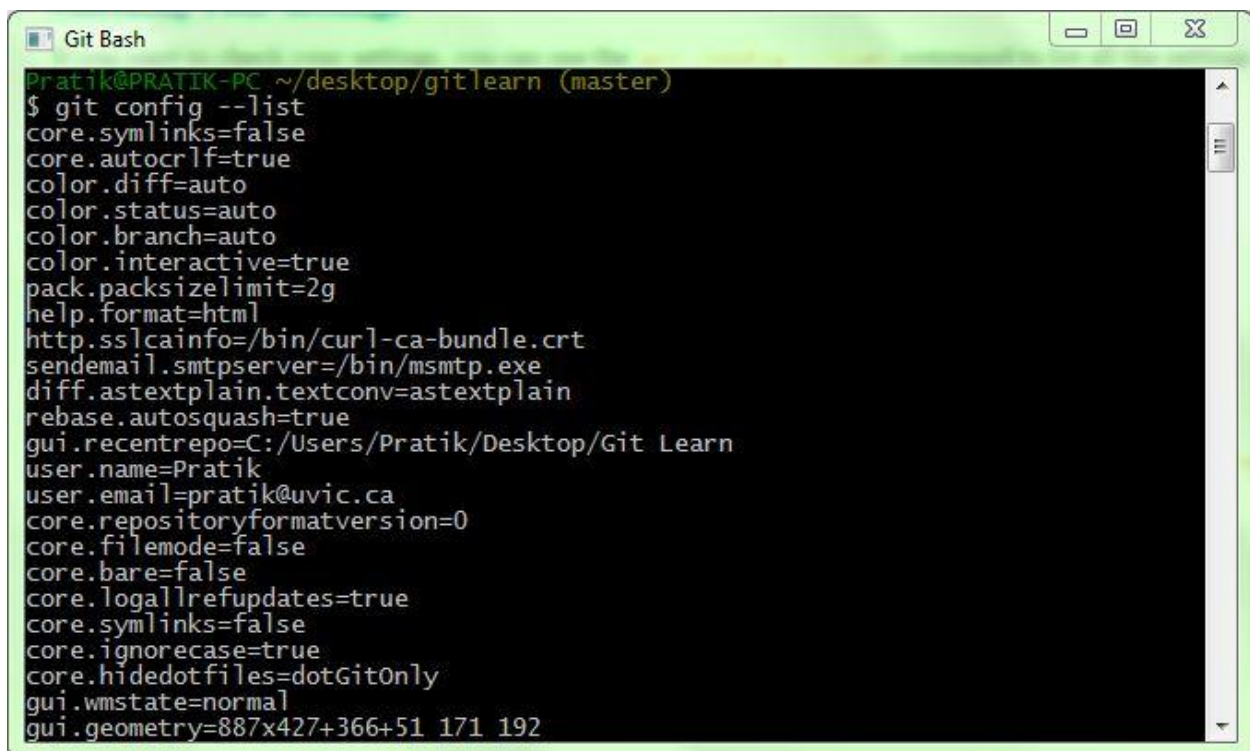
By default Git uses default text editor which is Vi or Vim but you can change editor such as emacs by following command.

```
git config --global core.editor emacs
```

GIT CONFIG

It will show all the attributes of configuration file which we have set.

```
git config --list
```



```
Git Bash
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git config --list
core.symlinks=false
core.autocrlf=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
pack.packsizelimit=2g
help.format=html
http.sslcainfo=/bin/curl-ca-bundle.crt
sendemail.smtpserver=/bin/msmtp.exe
diff.astextplain.textconv=astextplain
rebase.autosquash=true
gui.recentrepo=C:/Users/Pratik/Desktop/Git Learn
user.name=Pratik
user.email=pratik@uvic.ca
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
core.hidedotfiles=dotGitOnly
gui.wmstate=normal
gui.geometry=887x427+366+51 171 192
```

GIT HELP

It will open a browser window as a help for config command.

```
git help config
```

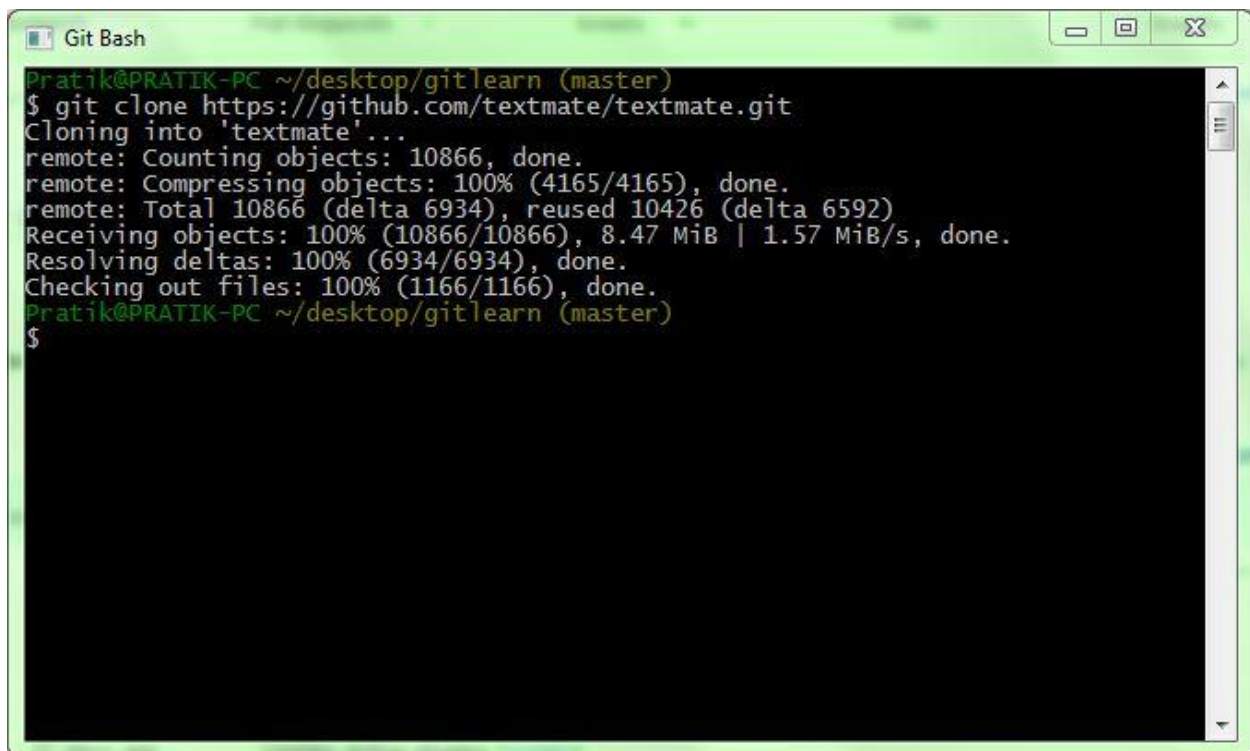
CREATING REPOSITORY

It will create .git directory in your project directory.

git init

CLONING REPOSITORY

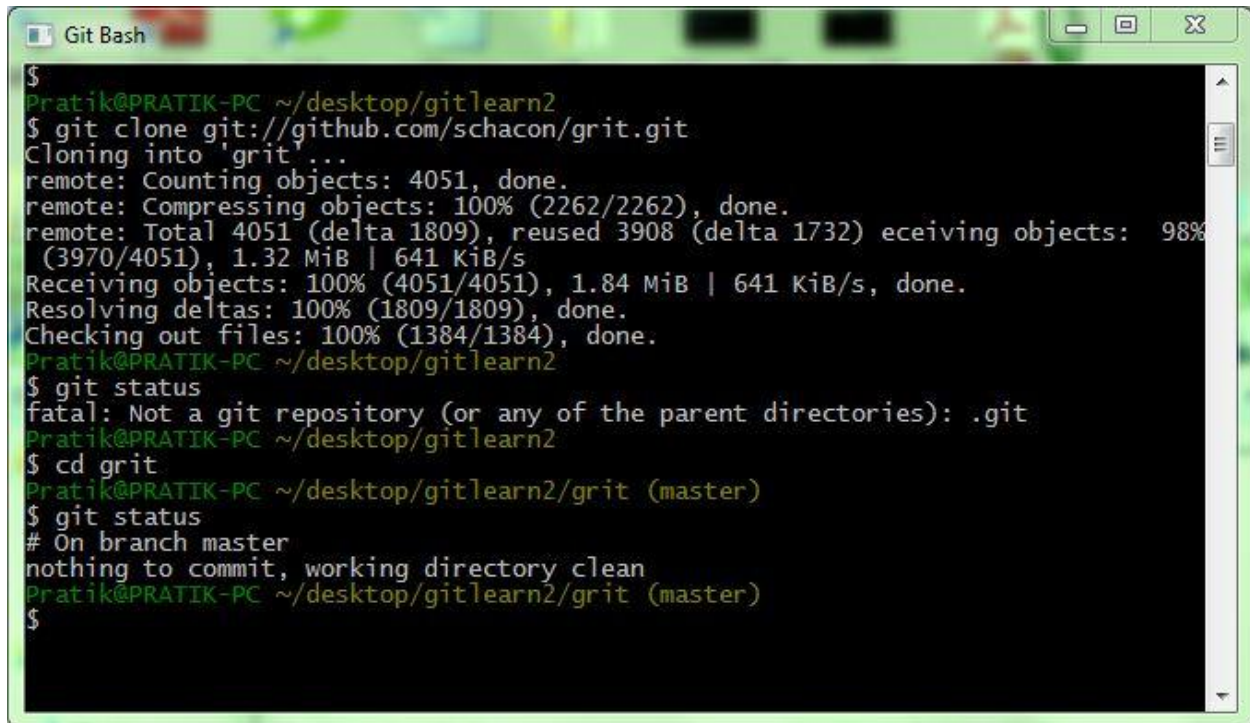
Instead of using https://, you can use git:// protocol or user@server:/path.git, which uses the SSH transfer protocol.

A screenshot of a Git Bash terminal window. The window title is "Git Bash". The prompt is "Pratik@PRATIK-PC ~/desktop/gitlearn (master)". The user enters the command "\$ git clone https://github.com/textmate/textmate.git". The terminal output shows the cloning process: "Cloning into 'textmate'...", "remote: Counting objects: 10866, done.", "remote: Compressing objects: 100% (4165/4165), done.", "remote: Total 10866 (delta 6934), reused 10426 (delta 6592)", "Receiving objects: 100% (10866/10866), 8.47 MiB | 1.57 MiB/s, done.", "Resolving deltas: 100% (6934/6934), done.", "Checking out files: 100% (1166/1166), done.", and finally "Pratik@PRATIK-PC ~/desktop/gitlearn (master)" followed by a "\$" prompt.

```
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git clone https://github.com/textmate/textmate.git
Cloning into 'textmate'...
remote: Counting objects: 10866, done.
remote: Compressing objects: 100% (4165/4165), done.
remote: Total 10866 (delta 6934), reused 10426 (delta 6592)
Receiving objects: 100% (10866/10866), 8.47 MiB | 1.57 MiB/s, done.
Resolving deltas: 100% (6934/6934), done.
Checking out files: 100% (1166/1166), done.
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$
```

GIT STATUS

If you run this command, just after cloning repository. You will see there is nothing to commit all files are tracked and unmodified.



```
Git Bash
$
Pratik@PRATIK-PC ~/desktop/gitlearn2
$ git clone git://github.com/schacon/grit.git
Cloning into 'grit' ...
remote: Counting objects: 4051, done.
remote: Compressing objects: 100% (2262/2262), done.
remote: Total 4051 (delta 1809), reused 3908 (delta 1732) receiving objects: 98%
(3970/4051), 1.32 MiB | 641 KiB/s
Receiving objects: 100% (4051/4051), 1.84 MiB | 641 KiB/s, done.
Resolving deltas: 100% (1809/1809), done.
Checking out files: 100% (1384/1384), done.
Pratik@PRATIK-PC ~/desktop/gitlearn2
$ git status
fatal: Not a git repository (or any of the parent directories): .git
Pratik@PRATIK-PC ~/desktop/gitlearn2
$ cd grit
Pratik@PRATIK-PC ~/desktop/gitlearn2/grit (master)
$ git status
# On branch master
nothing to commit, working directory clean
Pratik@PRATIK-PC ~/desktop/gitlearn2/grit (master)
$
```

You can use `-s` option for checking status in short and concise way.

It shows result which has below terminology.

? Untracked files

A Added files

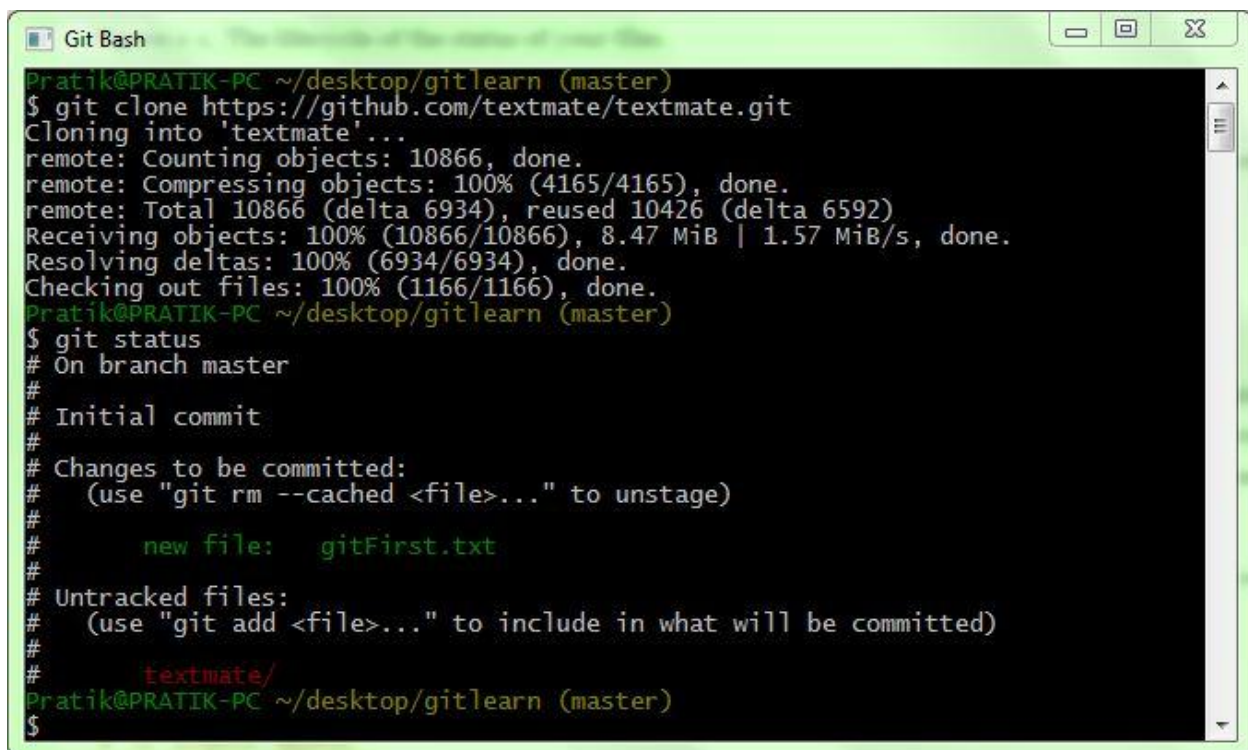
M Modified files

R Rename

TRACKING NEW FILES

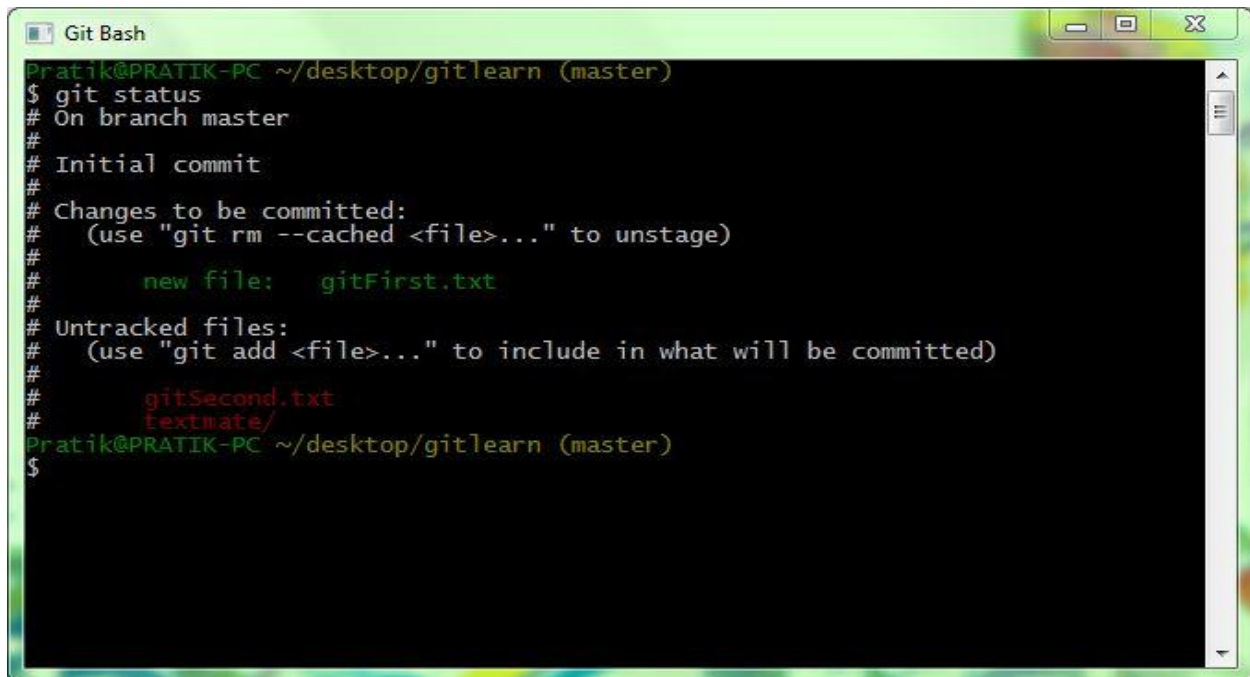
Create a first file `gitFirst.txt` and check status. We need to add this new `gitFirst.txt` file and make it ready to commit but we can see clone repository or file `textmate` still untracked.

```
git add gitFirst.txt
```



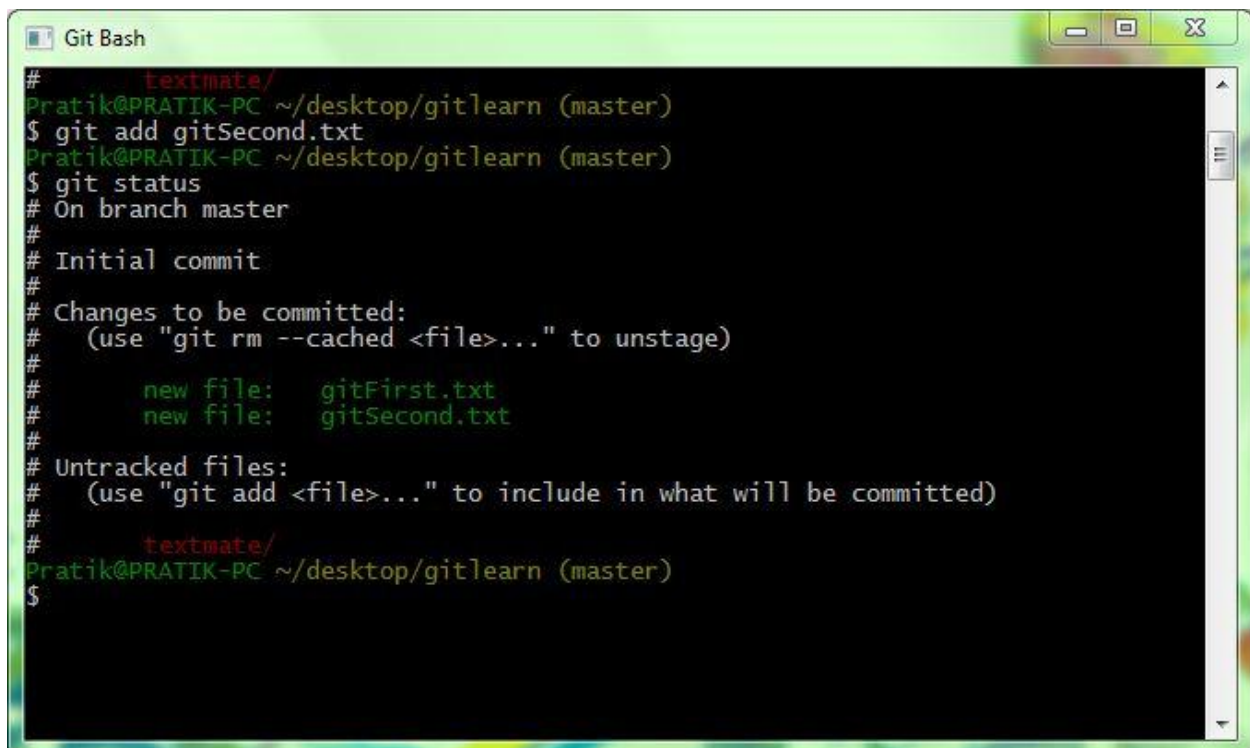
```
Git Bash
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git clone https://github.com/textmate/textmate.git
Cloning into 'textmate'...
remote: Counting objects: 10866, done.
remote: Compressing objects: 100% (4165/4165), done.
remote: Total 10866 (delta 6934), reused 10426 (delta 6592)
Receiving objects: 100% (10866/10866), 8.47 MiB | 1.57 MiB/s, done.
Resolving deltas: 100% (6934/6934), done.
Checking out files: 100% (1166/1166), done.
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   gitFirst.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       textmate/
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$
```

Create a file `second.txt` in project folder and check status again.



```
Git Bash
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   gitFirst.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       gitSecond.txt
#       textmate/
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$
```

Add file `gitSecond.txt` to stage for commit.

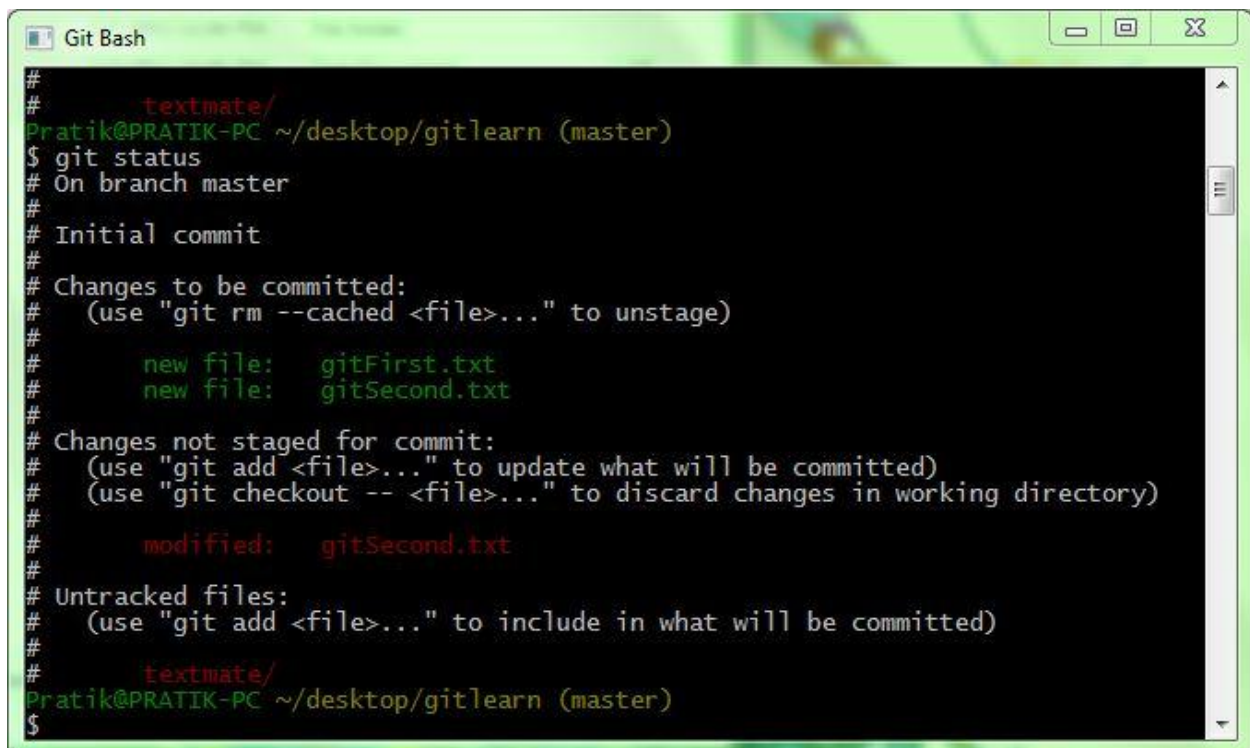


```
Git Bash
#       textmate/
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git add gitSecond.txt
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   gitFirst.txt
#       new file:   gitSecond.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       textmate/
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$
```

Modify gitSecond.txt file and check git status again.

While checking again, you will find gitSecond.txt is in both staged and unstaged area.

Git stage file as you run git add command, so if you commit now git will send a gitSecond.txt version which you added it earlier and is in staged area.



```
Git Bash
#
#      textmate/
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   gitFirst.txt
#       new file:   gitSecond.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 directory)
#
#       modified:   gitSecond.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#      textmate/
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$
```

After each modification you need to add a file.

GIT IGNORING

There can be bunch of files, in your project directory which you want git to ignore.

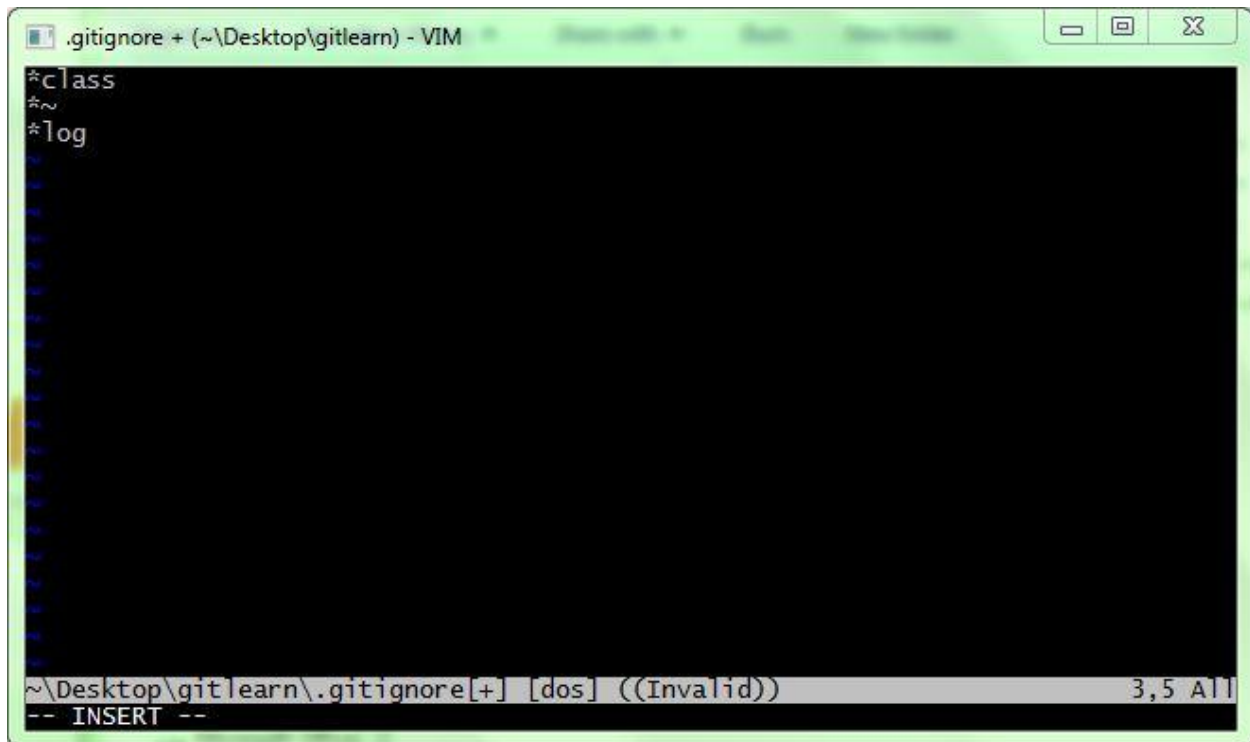
Git should not show them to add or as untracked. These are basically auto generated files like log files, or temporary files.

To ignore those files create .gitignore named file listing patterns to match files.



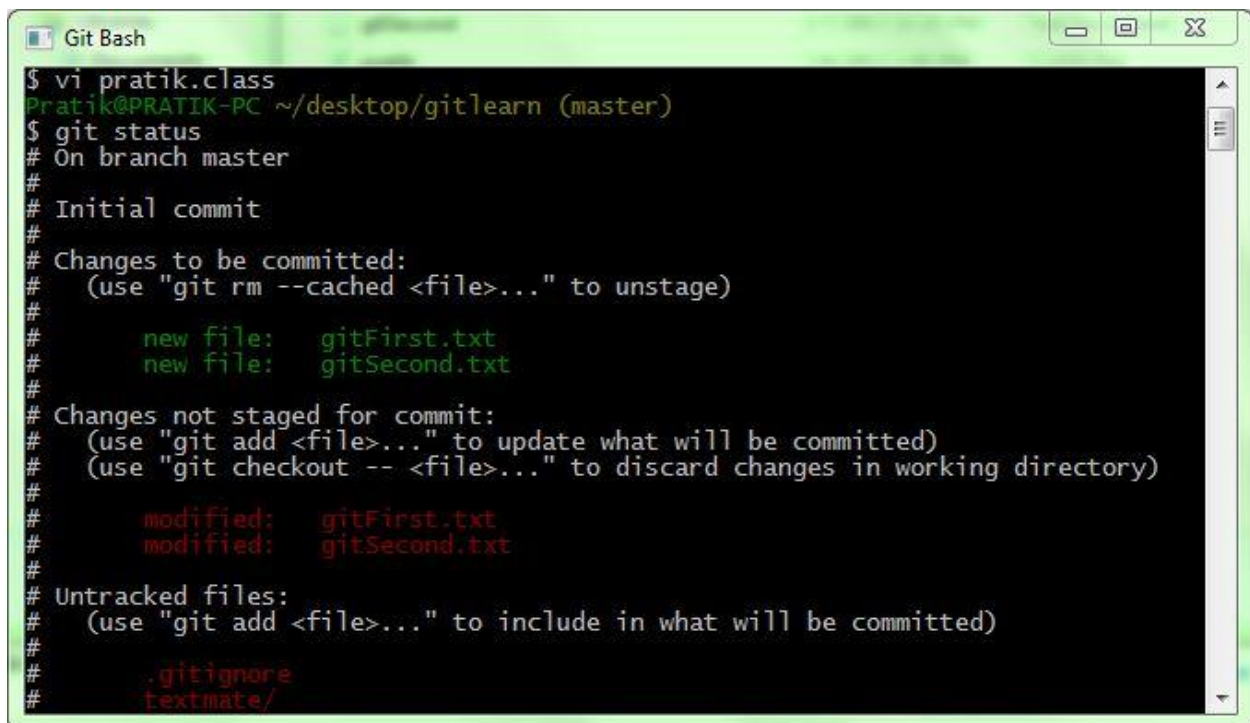
```
Git Bash
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ vi .gitignore
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$
```


In `.gitignore` file list all patterns which you want git to ignore.



```
.gitignore + (~\Desktop\gitlearn) - VIM
^class
^~
^log
-- INSERT --
```

Create a `.class` file to check that git is ignoring those class files or not.



```
Git Bash
$ vi pratik.class
Pratik@PRATIK-PC ~/\desktop/gitlearn (master)
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   gitFirst.txt
#       new file:   gitSecond.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 directory)
#
#       modified:   gitFirst.txt
#       modified:   gitSecond.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       .gitignore
#       textmate/
```

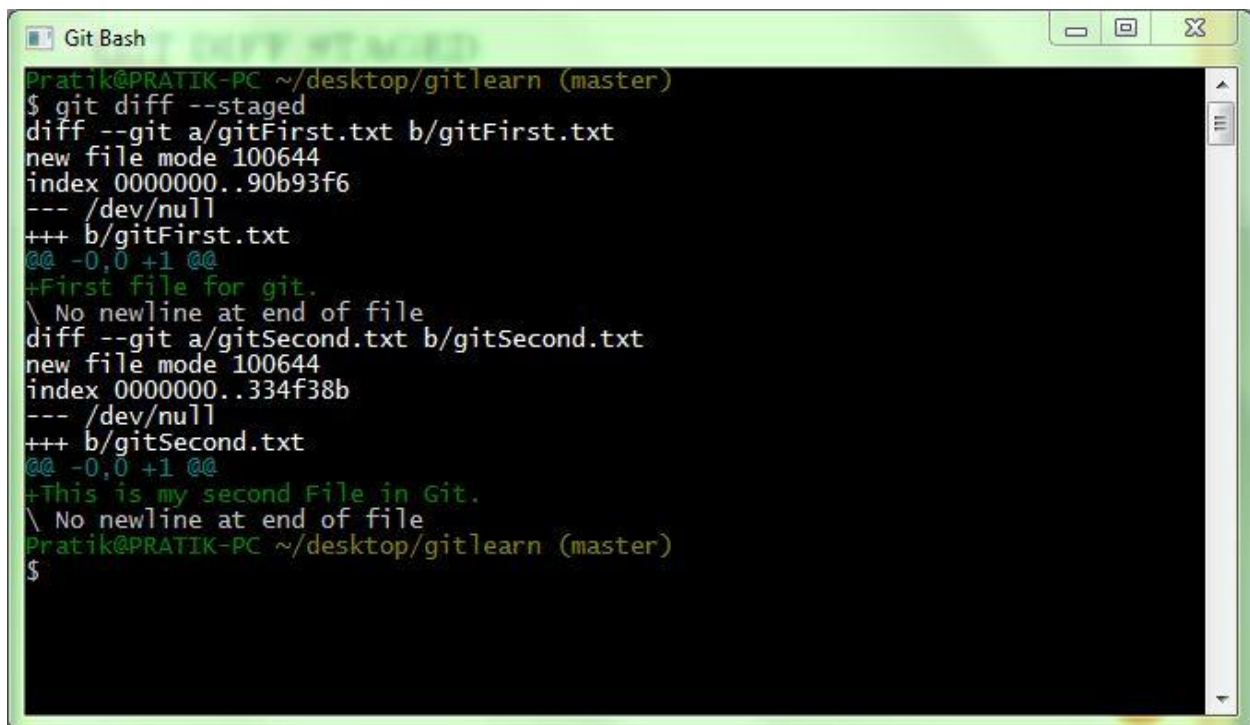
GIT DIFF

You can check your staged and unstaged changes through *git status* but *git diff* will give details of what is changed in files.

To see what you have changed but not yet staged, type *git diff* with no other arguments.

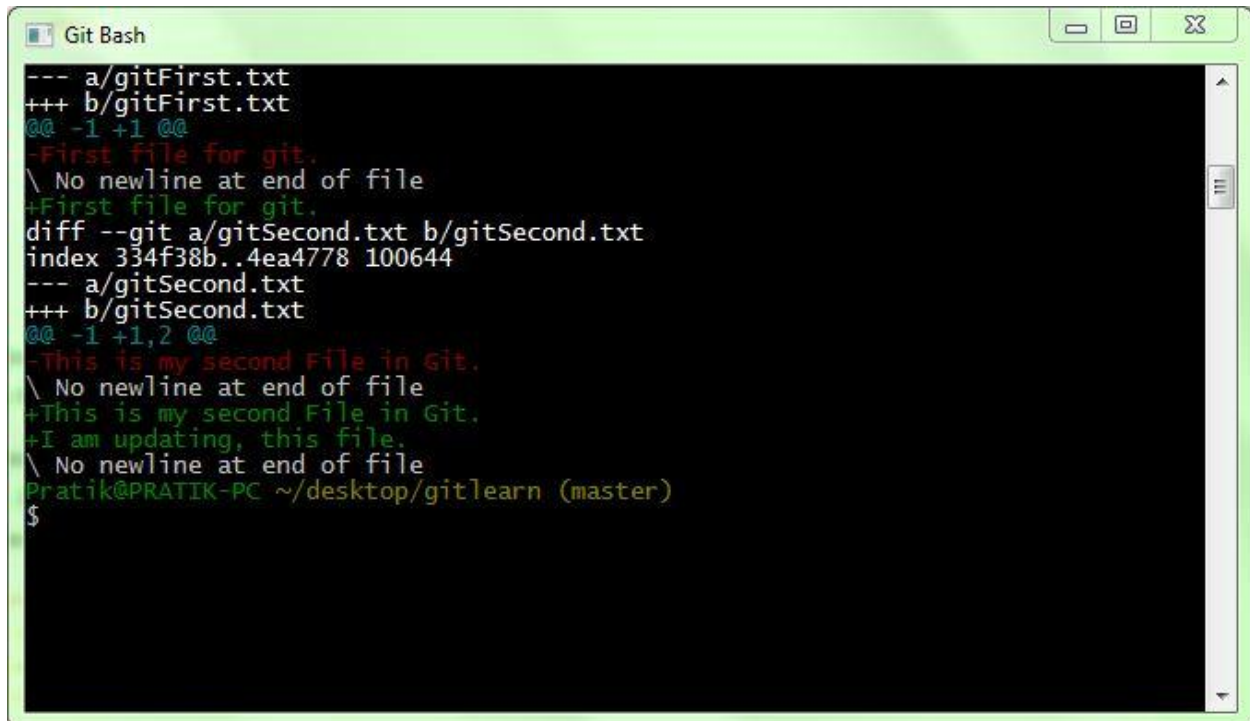
Git diff --staged

It gives the details of files which you have staged and will go into next commit.



```
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git diff --staged
diff --git a/gitFirst.txt b/gitFirst.txt
new file mode 100644
index 0000000..90b93f6
--- /dev/null
+++ b/gitFirst.txt
@@ -0,0 +1 @@
+First file for git.
\ No newline at end of file
diff --git a/gitSecond.txt b/gitSecond.txt
new file mode 100644
index 0000000..334f38b
--- /dev/null
+++ b/gitSecond.txt
@@ -0,0 +1 @@
+This is my second File in Git.
\ No newline at end of file
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$
```

It's important to note that *git diff* by itself doesn't show all changes made since your last commit — only changes that are still unstaged. If you have staged all your changes, git diff will give you no output.

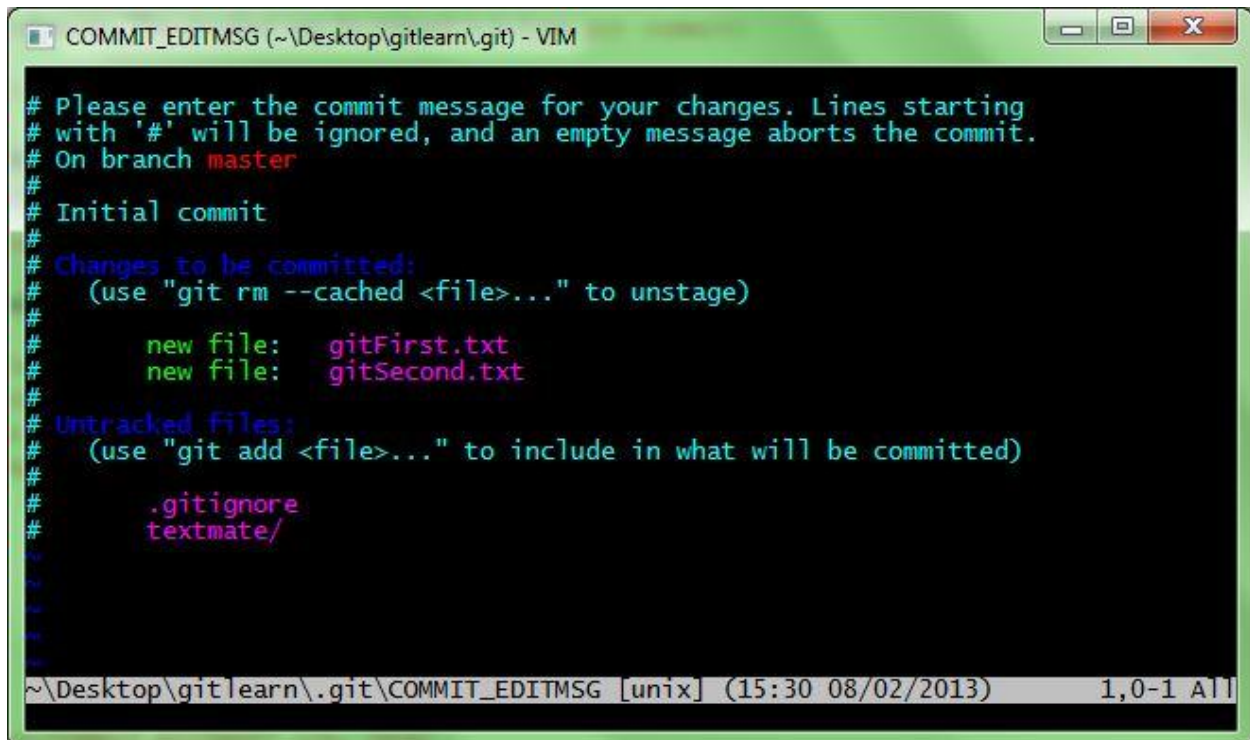


```
Git Bash
--- a/gitFirst.txt
+++ b/gitFirst.txt
@@ -1,1 @@
-First file for git.
\ No newline at end of file
+First file for git.
diff --git a/gitSecond.txt b/gitSecond.txt
index 334f38b..4ea4778 100644
--- a/gitSecond.txt
+++ b/gitSecond.txt
@@ -1,1,2 @@
-This is my second File in Git.
\ No newline at end of file
+This is my second File in Git.
+I am updating, this file.
\ No newline at end of file
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$
```

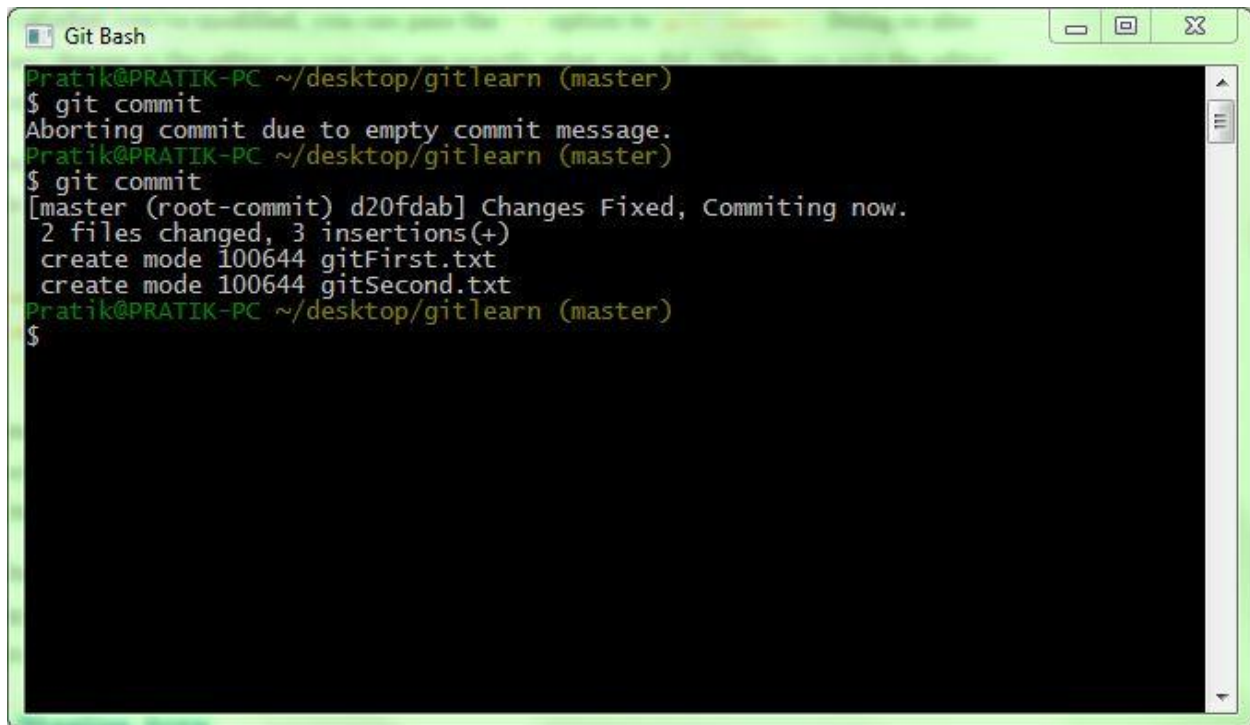
GIT COMMIT

Check if all the files are staged, then you can commit. If any file is in unstaged area that won't go in commit.

Just type *git commit* to commit. Git commit command gives you access to write some message before committing.



```
COMMIT_EDITMSG (~\Desktop\gitlearn\.git) - VIM
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   gitFirst.txt
#       new file:   gitSecond.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       .gitignore
#       textmate/
#
#
~\Desktop\gitlearn\.git\COMMIT_EDITMSG [unix] (15:30 08/02/2013) 1,0-1 All
```

A screenshot of a Git Bash terminal window. The window title is "Git Bash". The terminal shows the following commands and output:

```
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git commit
Aborting commit due to empty commit message.
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git commit
[master (root-commit) d20fdab] Changes Fixed, Committing now.
2 files changed, 3 insertions(+)
create mode 100644 gitFirst.txt
create mode 100644 gitSecond.txt
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$
```

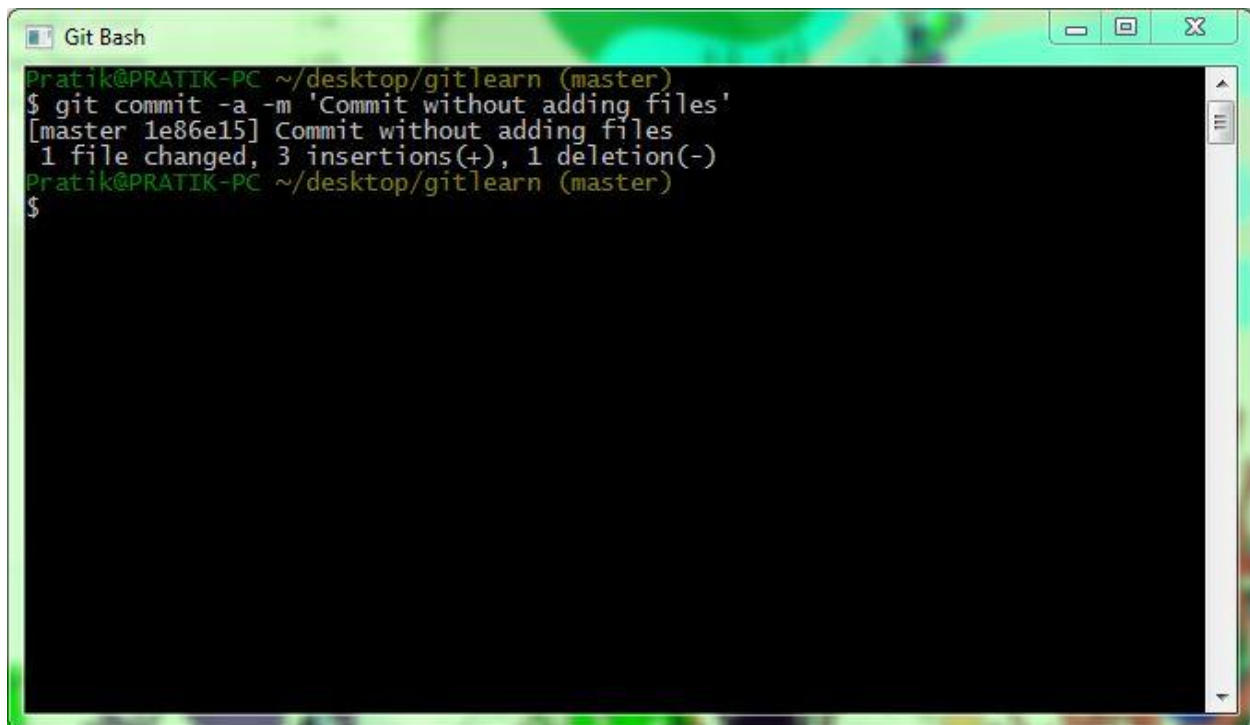
This will commit all your changes which are staged with SHA1 reference.

Without Message in editor

git commit -m "Committing First Changes"

If you modify some file, for example – gitSecond.txt and try to commit without adding it. Then it won't be possible. First you have to add then commit.

You can use commit with option –a which will commit and add all the unstaged files.

A screenshot of a Git Bash terminal window. The window title is "Git Bash". The terminal shows the following text:

```
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git commit -a -m 'Commit without adding files'
[master 1e86e15] Commit without adding files
1 file changed, 3 insertions(+), 1 deletion(-)
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$
```

We can see from our commit, which branch you committed to (master), what SHA-1 checksum the commit has (1e86e15), how many files were changed, and statistics about lines added and removed in the commit.

Everytime you are committing, you are taking snapshots of your staging area.

GIT REMOVE

To remove files which are tracked or in staging area, you need `git rm` command and then commit.

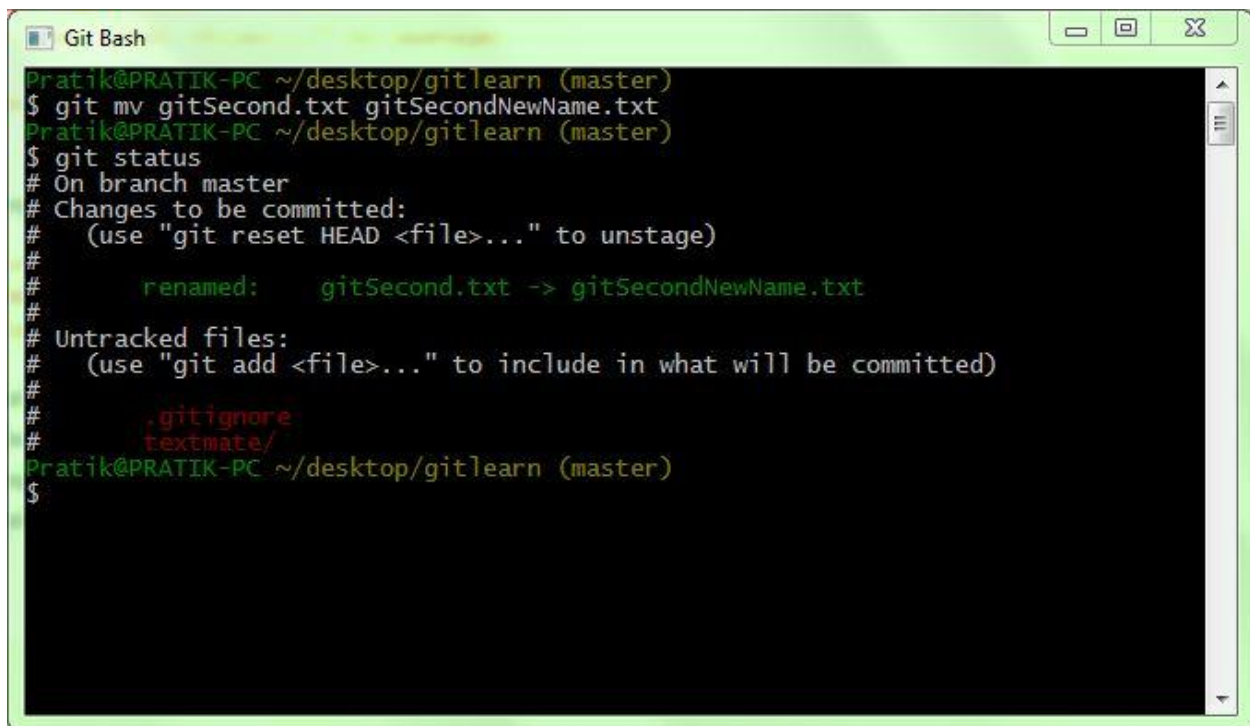
Removes all the Files that end with ~

```
$ git rm \*~
```

If you simply remove the file from your working directory using `rm` command, file will be in unstaged area("Changes not staged for commit").

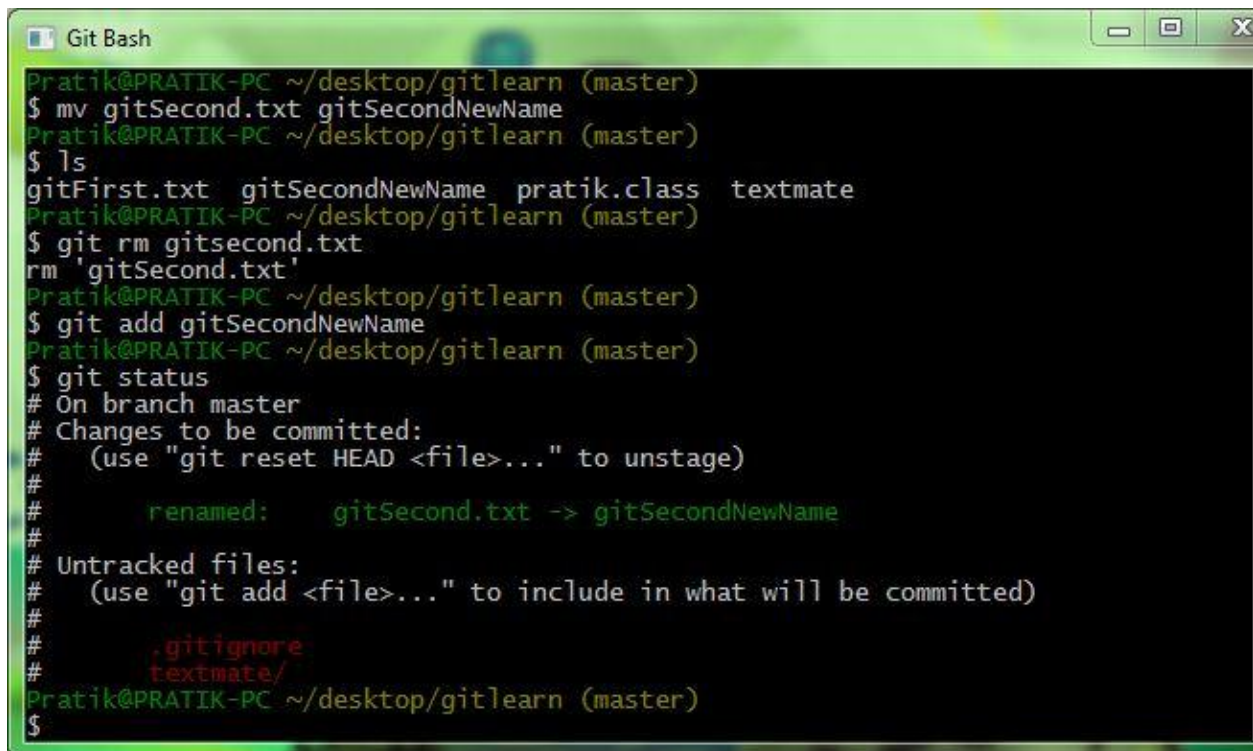
GIT MOVE

Git explicitly does not track file movement, and it does not store any metadata like if you rename the file. But git has mv command.



```
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git mv gitSecond.txt gitSecondNewName.txt
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       renamed:   gitSecond.txt -> gitSecondNewName.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       .gitignore
#       textmate/
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$
```

This git mv command is actually equivalent to three commands stated in below example.



```
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ mv gitSecond.txt gitSecondNewName
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ ls
gitFirst.txt  gitSecondNewName  pratik.class  textmate
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git rm gitsecond.txt
rm 'gitSecond.txt'
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git add gitSecondNewName
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       renamed:    gitSecond.txt -> gitSecondNewName
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       .gitignore
#       textmate/
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$
```


GIT LOG

Gives you details with checksum and messages in chronological order of commits.

git log

Option `-p` will show differences introduced in each commit and `-2` will restrict it to only last two entries.

git log -p -2

If we want to see some abbreviated stats for each commit. It also puts summary of information at the end.

git log --stat

Option `--pretty` changes the log output to formats other than the default. There are some predefined formats.

git log --pretty=format:"%h - %an, %ar : %s"

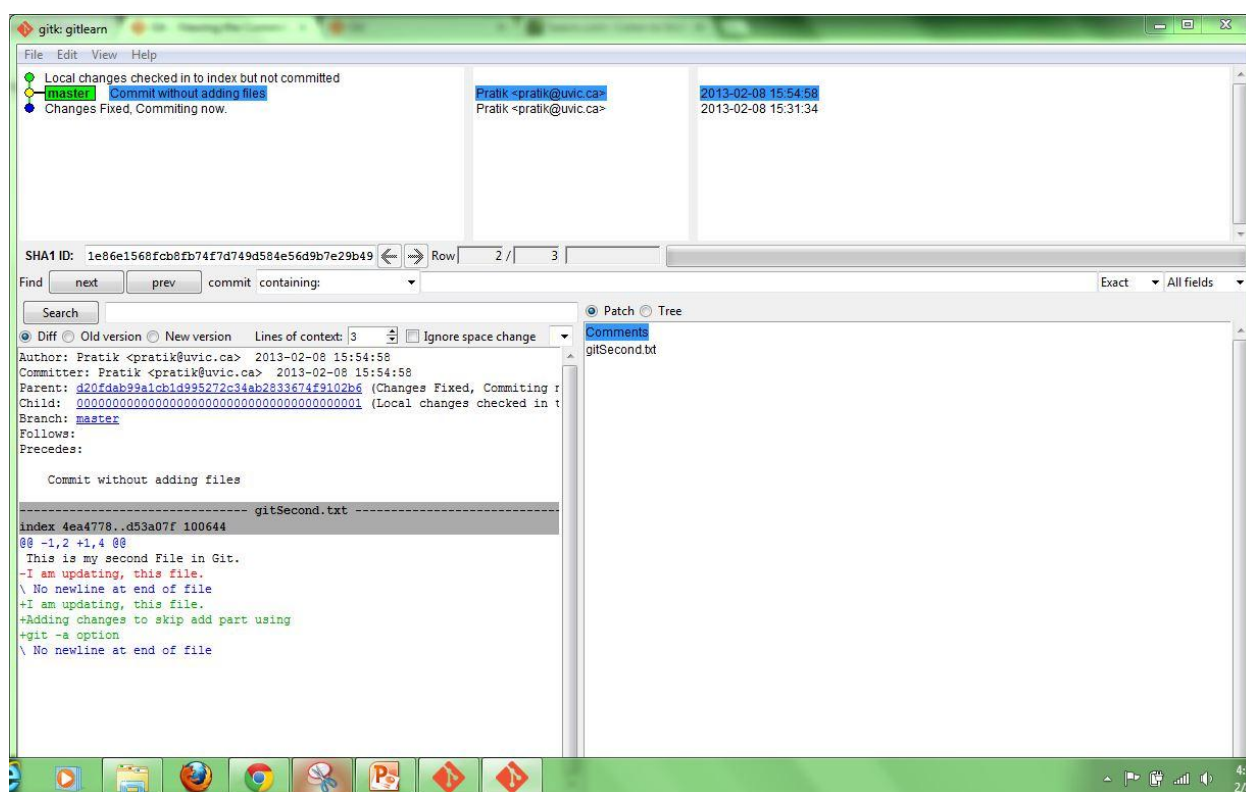
Where `%h` is abbr. commit hash, `%an` is author name, `%ar` is author date(relative) and `%s` is subject or message.

`%ad` can also be used for date format.

GIT GUI FOR HISTORY

Visualize tool with git to check commit history. Its basically a *git log* tool and accepts all options which git log provides.

gitk



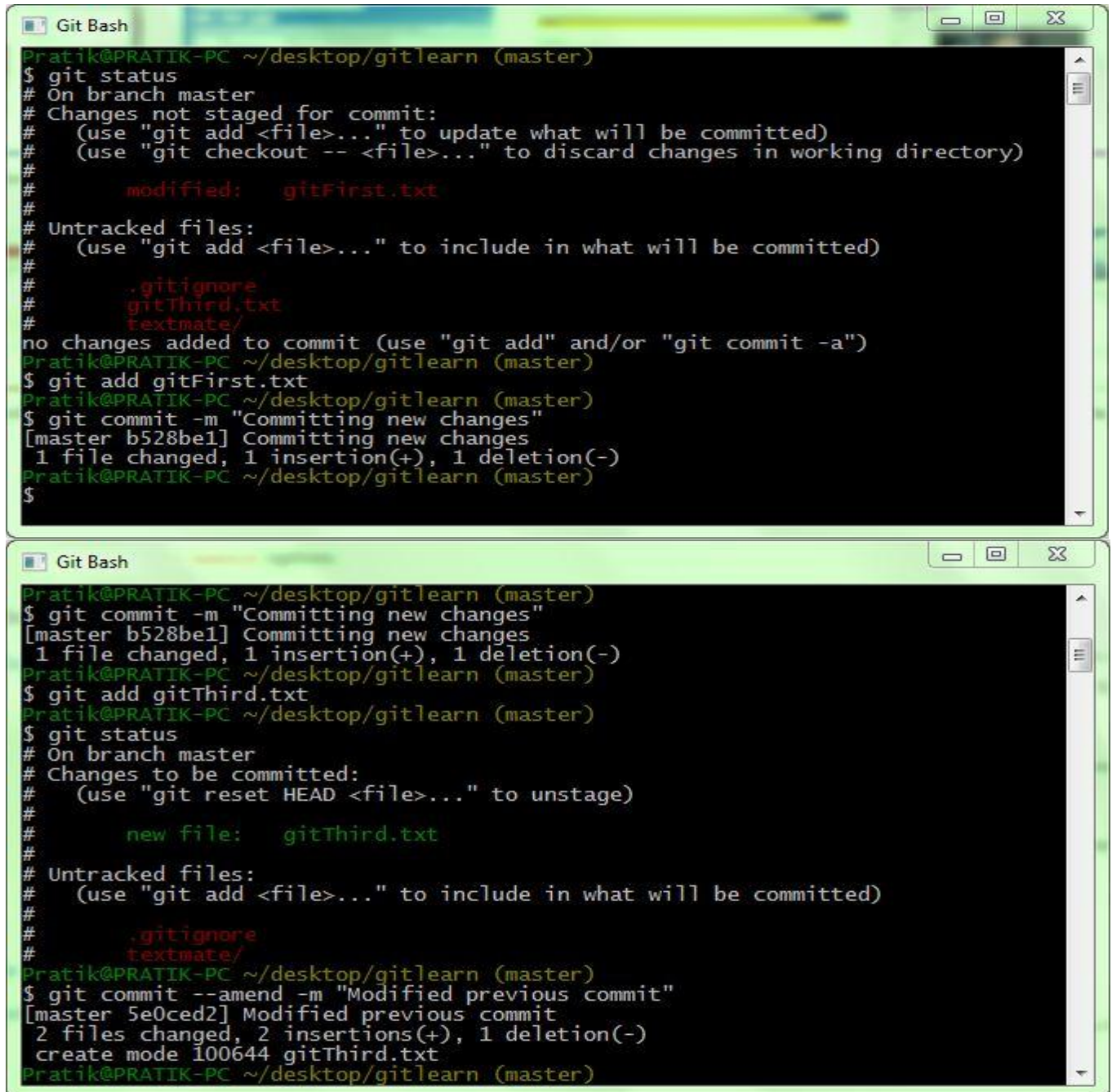
It comes with git gui tool also.

git gui

GIT UNDO

--amend option with git commit command gives you freedom to revert your changes and then commit with new message.

Here is an example when you forgot to add some files in a first commit, commit with amend having new message.

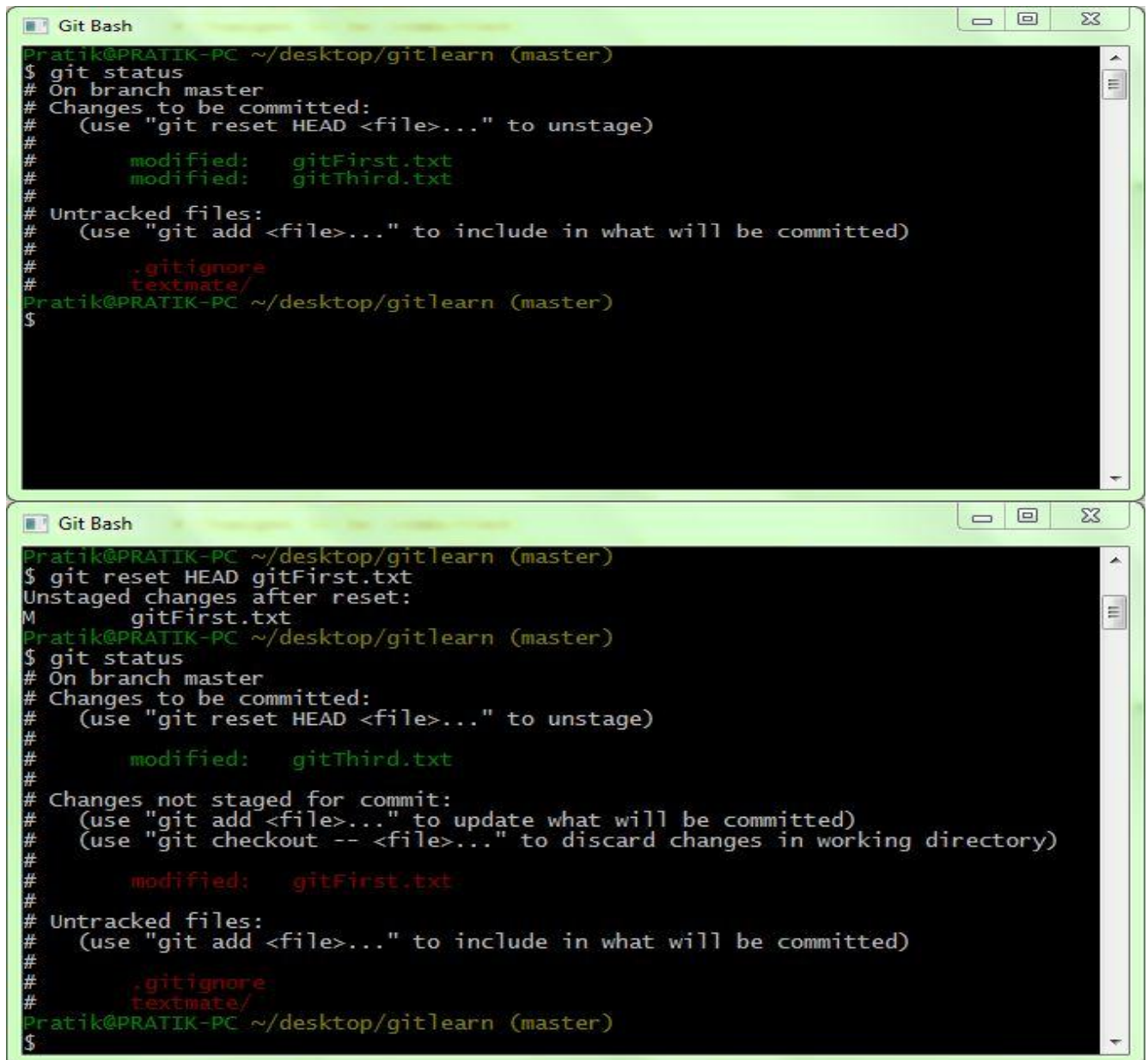


```
Git Bash
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   gitFirst.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       .gitignore
#       gitThird.txt
#       textmate/
no changes added to commit (use "git add" and/or "git commit -a")
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git add gitFirst.txt
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git commit -m "Committing new changes"
[master b528be1] Committing new changes
 1 file changed, 1 insertion(+), 1 deletion(-)
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$

Git Bash
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git commit -m "Committing new changes"
[master b528be1] Committing new changes
 1 file changed, 1 insertion(+), 1 deletion(-)
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git add gitThird.txt
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   gitThird.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       .gitignore
#       textmate/
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git commit --amend -m "Modified previous commit"
[master 5e0ced2] Modified previous commit
 2 files changed, 2 insertions(+), 1 deletion(-)
 create mode 100644 gitThird.txt
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
```

UNSTAGING STAGED FILE

If you modify two files and want to commit them as two separate changes, but accidentally both changes are staged now. You can unstage changes using *git reset head* command

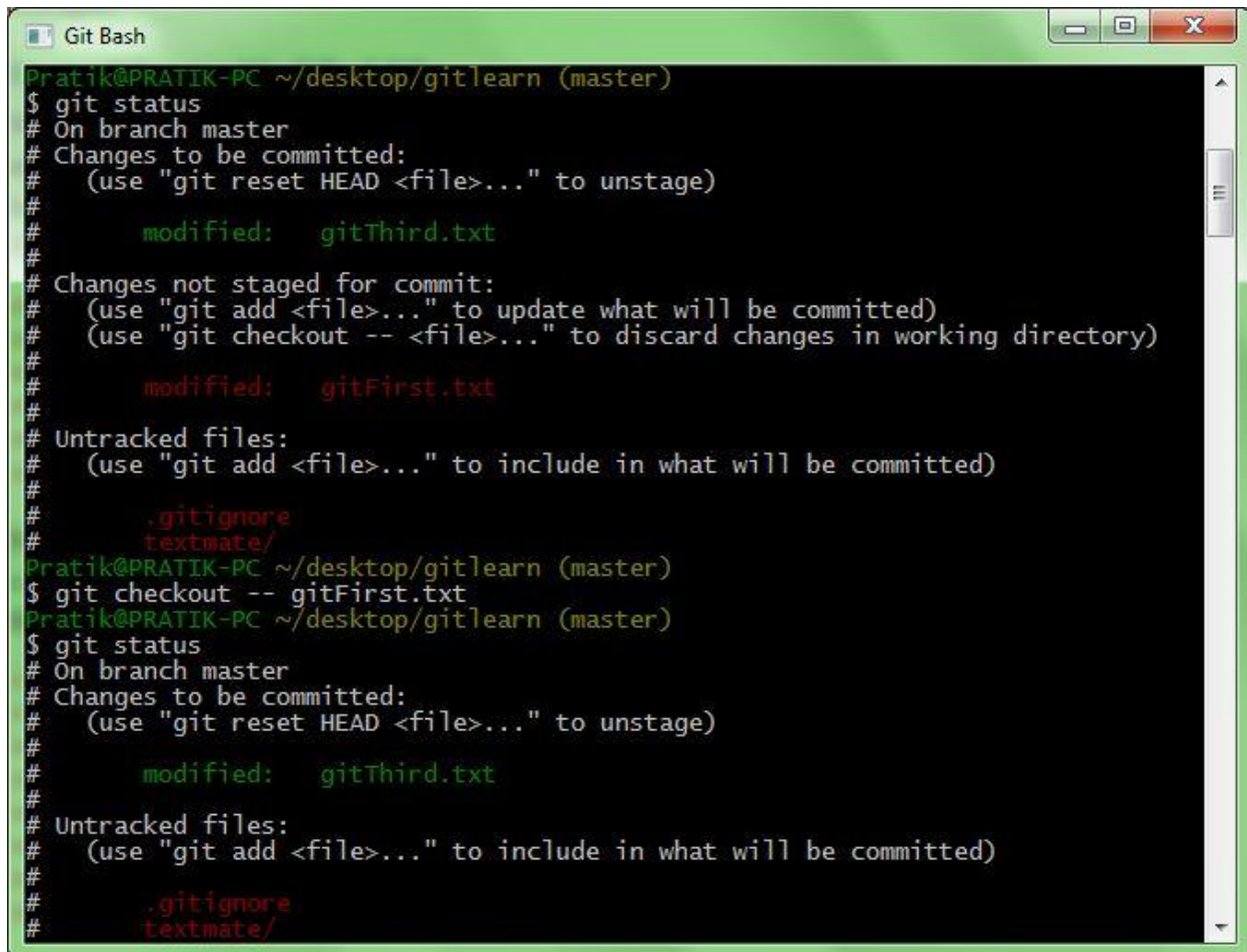


```
Git Bash
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   gitFirst.txt
#       modified:   gitThird.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       .gitignore
#       textmate/
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$

Git Bash
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git reset HEAD gitFirst.txt
Unstaged changes after reset:
M   gitFirst.txt
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   gitThird.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 directory)
#
#       modified:   gitFirst.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       .gitignore
#       textmate/
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$
```

UNMODIFYING MODIFIED FILE

Files which are not staged and are modified, we can unmodify it using *git checkout -- <file>*. For ex –unmodify a file which you have changed like gitFirst.txt.

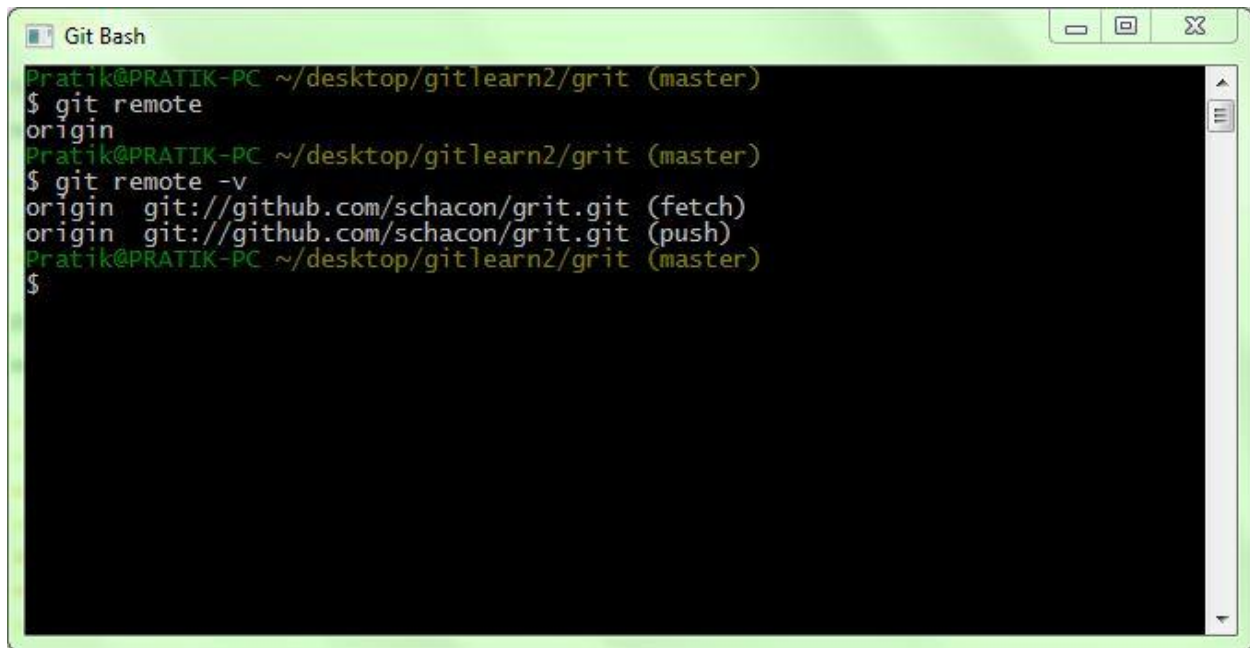
A screenshot of a Git Bash terminal window. The window title is "Git Bash". The terminal shows the following commands and output:

```
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   gitThird.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 directory)
#
#       modified:   gitFirst.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       .gitignore
#       textmate/
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git checkout -- gitFirst.txt
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   gitThird.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       .gitignore
#       textmate/
```

Remember any changes using this command, will be gone forever.

REMOTE REPOSITORIES

To check all remote handles, from a particular repository. We can use *git remote* command. Cloning a repository from any Git server gives a default name *origin* to that server.



```
Git Bash
Pratik@PRATIK-PC ~/desktop/gitlearn2/grit (master)
$ git remote
origin
Pratik@PRATIK-PC ~/desktop/gitlearn2/grit (master)
$ git remote -v
origin git://github.com/schacon/grit.git (fetch)
origin git://github.com/schacon/grit.git (push)
Pratik@PRATIK-PC ~/desktop/gitlearn2/grit (master)
$
```

-v option gives the full name of repository which git has stored with short name.

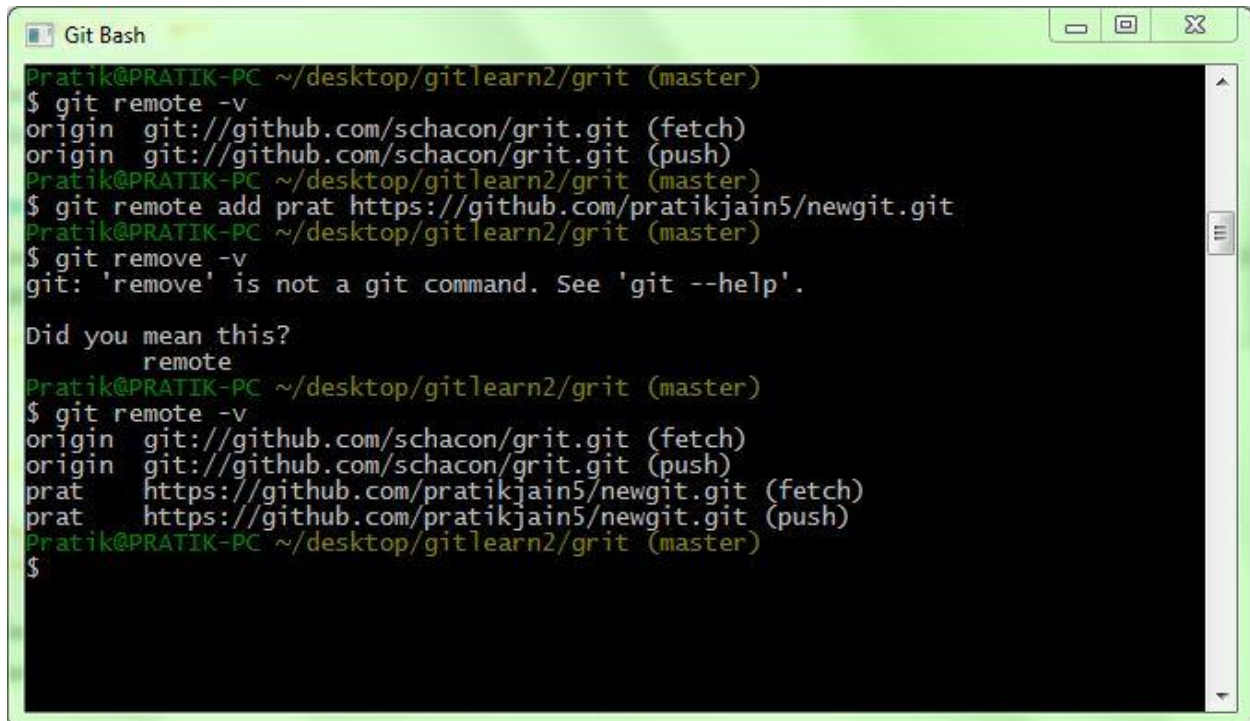
In git if we have multiple git remote repository. We can easily pull from any of repositories but push can be done with only ssh url origin. For ex-

```
$ cd grit
$ git remote -v
bakkdoor git://github.com/bakkdoor/grit.git
cho45 git://github.com/cho45/grit.git
defunkt git://github.com/defunkt/grit.git
koke git://github.com/koke/grit.git
origin git@github.com:mojombo/grit.git
```

ADDING REMOTE REPOSITORY

Create a new repository in Github, then add a remote repository using :

```
git remote add [shortname] [url]
```



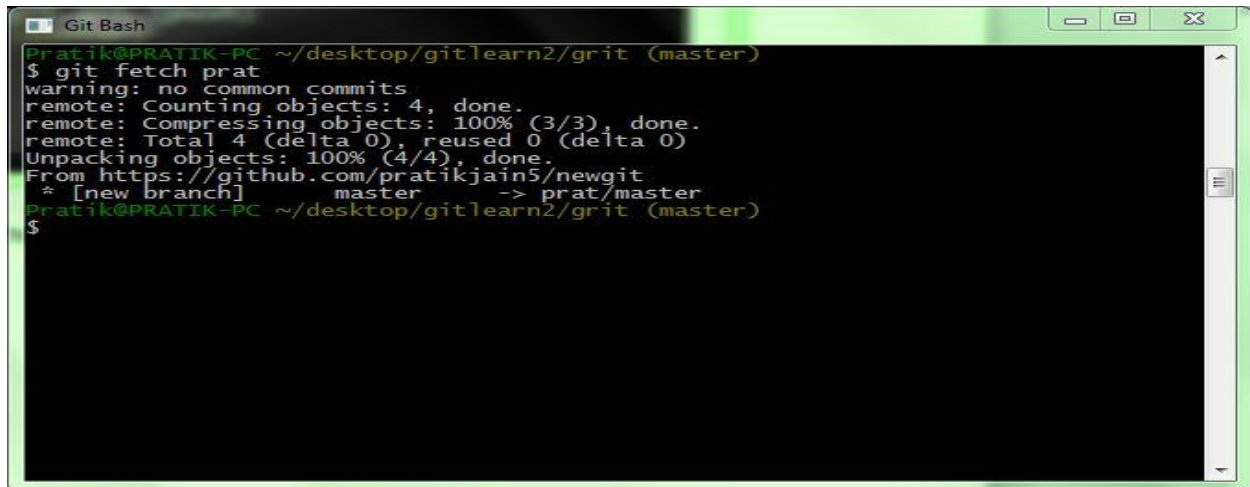
```
Pratik@PRATIK-PC ~/desktop/gitlearn2/grit (master)
$ git remote -v
origin git://github.com/schacon/grit.git (fetch)
origin git://github.com/schacon/grit.git (push)
Pratik@PRATIK-PC ~/desktop/gitlearn2/grit (master)
$ git remote add prat https://github.com/pratikjain5/newgit.git
Pratik@PRATIK-PC ~/desktop/gitlearn2/grit (master)
$ git remove -v
git: 'remove' is not a git command. See 'git --help'.

Did you mean this?
    remote
Pratik@PRATIK-PC ~/desktop/gitlearn2/grit (master)
$ git remote -v
origin git://github.com/schacon/grit.git (fetch)
origin git://github.com/schacon/grit.git (push)
prat   https://github.com/pratikjain5/newgit.git (fetch)
prat   https://github.com/pratikjain5/newgit.git (push)
Pratik@PRATIK-PC ~/desktop/gitlearn2/grit (master)
$
```

FETCH REMOTE REPOSITORY

We can fetch from short named remote repository prat using:

```
git fetch [shortname]
```



```
Pratik@PRATIK-PC ~/desktop/gitlearn2/grit (master)
$ git fetch prat
warning: no common commits
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
From https://github.com/pratikjain5/newgit
* [new branch]      master       -> prat/master
Pratik@PRATIK-PC ~/desktop/gitlearn2/grit (master)
$
```

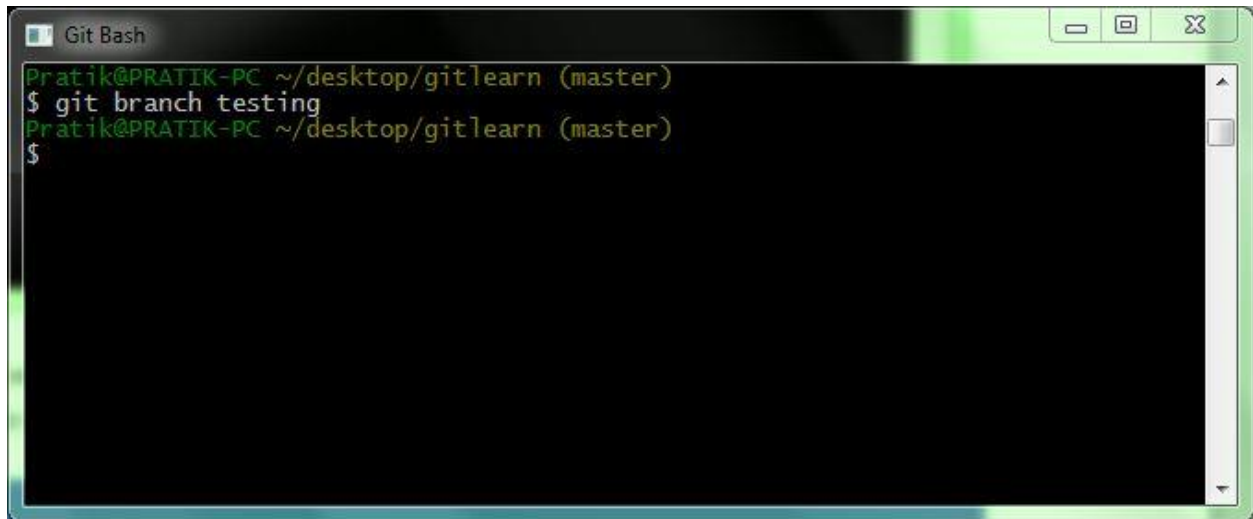
After fetching we will have all the references to all the branches from this remote repository. Fetch only pulls data to your local repository, any work you want to merge you have to do it manually.

git fetch origin will fetch any new data pushed to the server you cloned. Because origin is the default name of that server you cloned from.

GIT BRANCH

Create a git branch named testing.

git branch testing

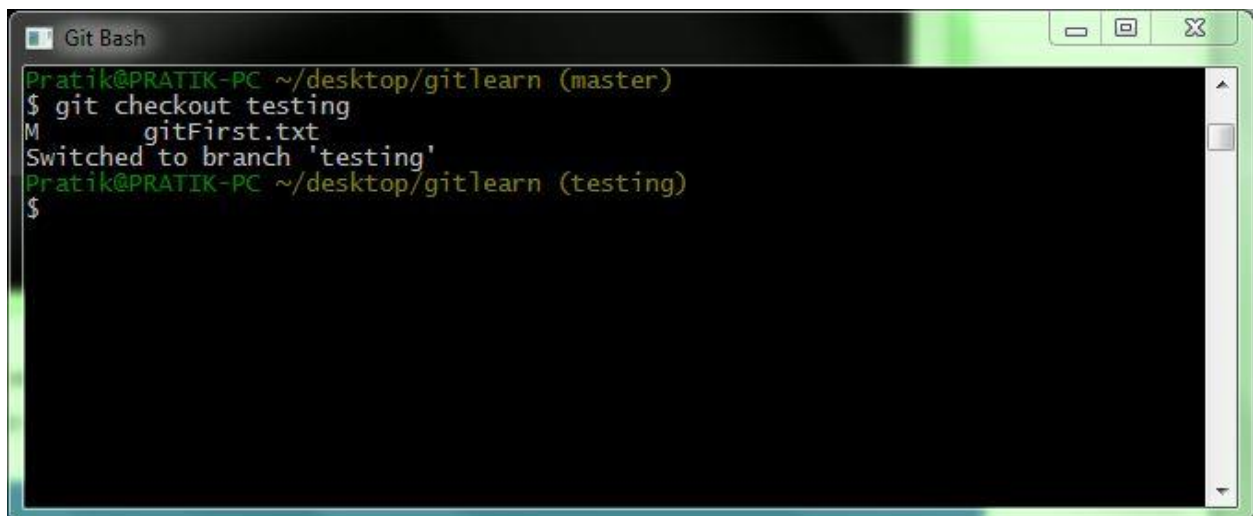


```
Git Bash
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git branch testing
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$
```

GIT CHECKOUT

Switch to branch named testing.

git checkout testing



```
Git Bash
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git checkout testing
M       gitFirst.txt
Switched to branch 'testing'
Pratik@PRATIK-PC ~/desktop/gitlearn (testing)
$
```

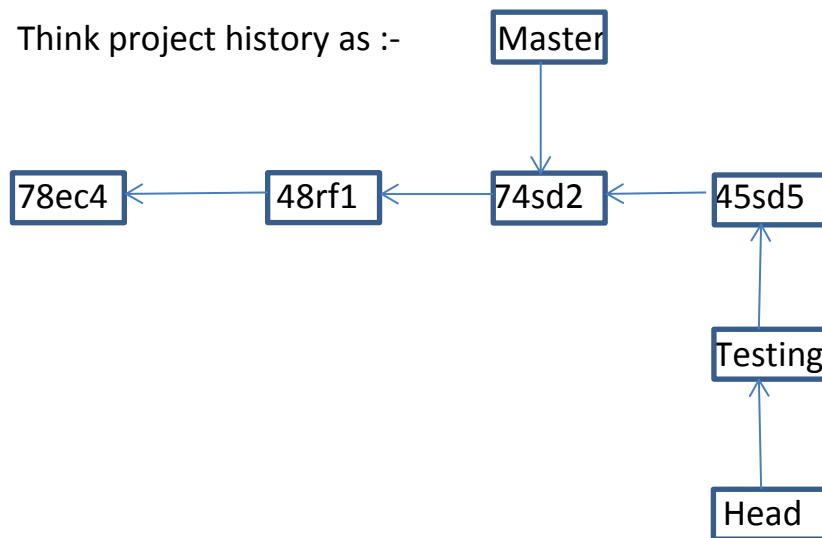
We can combine previous two commands create a git branch and then switching it to using `-b` option with checkout.

git checkout -b testing

Now head will point to testing.

Now you started working on your branch, and made few changes and commit.

Think project history as :-



DIFFERENCES BETWEEN GIT CLONE, GIT PULL, GIT FETCH

git pull will pull down from remote whatever trunk we are asking for, and it will instantly merge also by default into the local branch you are in when you make the request. Pull is a high-level request that runs 'fetch' then a 'merge' by default.

For ex-

```
$ git checkout localBranch
$ git pull origin master
$ git branch
Master
*localBranch
```

The above will merge the remote "master" branch into the local "localBranch".

git fetch it is similar to pull, only difference is it won't do merging by default.

For ex-

```
$ git checkout localBranch
$ git fetch origin remoteBranch
$ git branch
master
*localBranch
remoteBranch
```

git clone Git clone will clone a repo into a newly created directory. It's useful for when you're setting up your local repository.

```
$ cd newRepository
$ git clone git@github.com:whatever/something.git
$ git branch
*master
remoteBranch
```

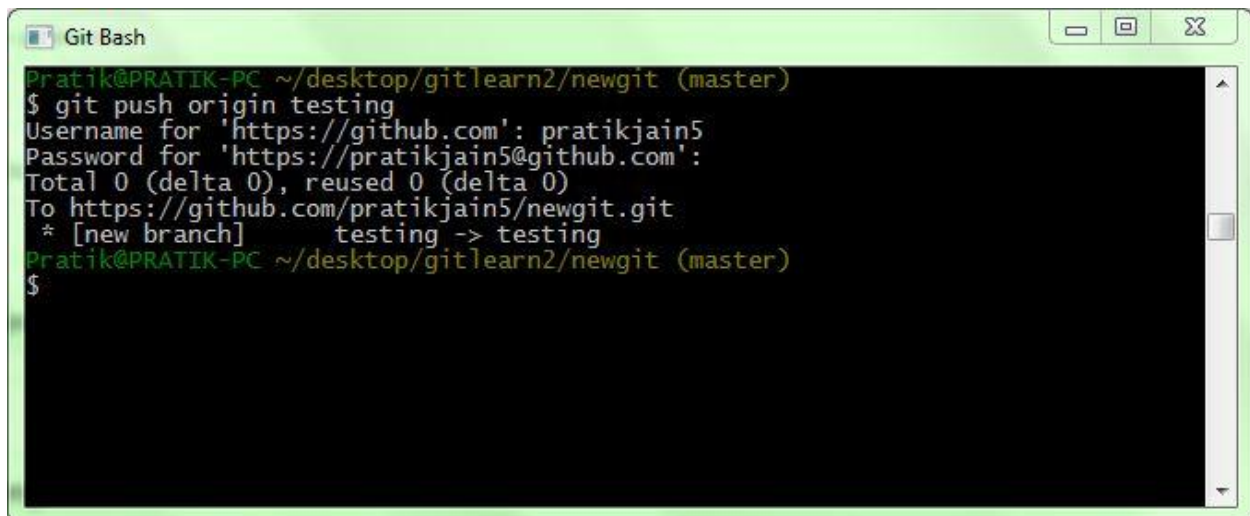
Git clone additionally creates a remote called 'origin' for the repo cloned from, sets up a local branch based on the remote's active branch (generally master), and creates remote-tracking branches for all the branches in the repo.

If you get stuck, run 'git branch -a' and it will show you exactly what's going on with your branches. You can see which are remotes and which are local.

GIT PUSH

To share your work to others you can push your branch to origin or to some other remote repository.

git push [remote name][branch name]



```
Git Bash
Pratik@PRATIK-PC ~/desktop/gitlearn2/newgit (master)
$ git push origin testing
Username for 'https://github.com': pratikjain5
Password for 'https://pratikjain5@github.com':
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/pratikjain5/newgit.git
 * [new branch]      testing -> testing
Pratik@PRATIK-PC ~/desktop/gitlearn2/newgit (master)
$
```

GIT TAGGING

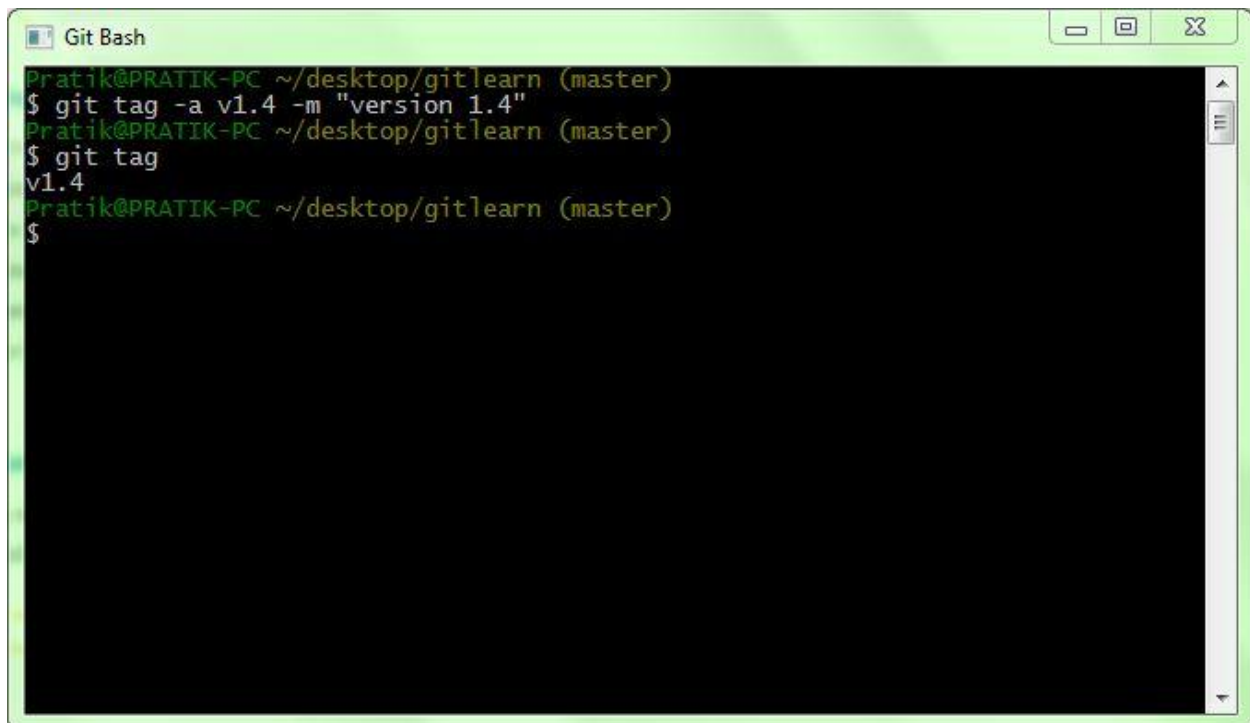
If at some point you feel this part is very important in history of your code or whatever you are working for. You can tag your work. Generally, people uses it for releasing versions like v1.0, v2.0

create a tag using:

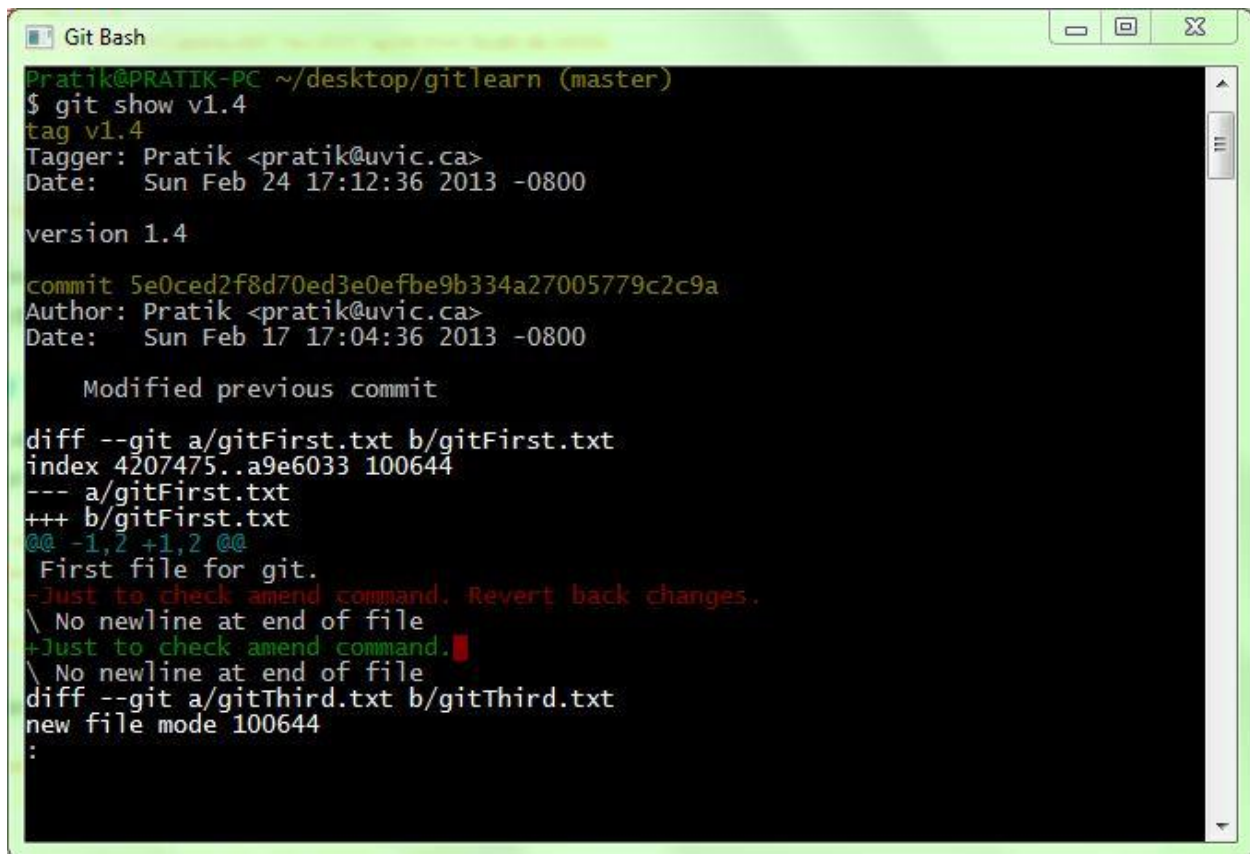
```
git tag -a [tag name] -m "Message"
```

-a option is for creating annotated tags.

To create lightweight or temporary tags do not specify -a or -m option.



```
Git Bash
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git tag -a v1.4 -m "version 1.4"
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git tag
v1.4
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$
```



```
Git Bash
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git show v1.4
tag v1.4
Tagger: Pratik <pratik@uvic.ca>
Date: Sun Feb 24 17:12:36 2013 -0800

version 1.4

commit 5e0ced2f8d70ed3e0efbe9b334a27005779c2c9a
Author: Pratik <pratik@uvic.ca>
Date: Sun Feb 17 17:04:36 2013 -0800

    Modified previous commit

diff --git a/gitFirst.txt b/gitFirst.txt
index 4207475..a9e6033 100644
--- a/gitFirst.txt
+++ b/gitFirst.txt
@@ -1,2 +1,2 @@
 First file for git.
-Just to check amend command. Revert back changes.
 \ No newline at end of file
+Just to check amend command.
 \ No newline at end of file
diff --git a/gitThird.txt b/gitThird.txt
new file mode 100644
:
```

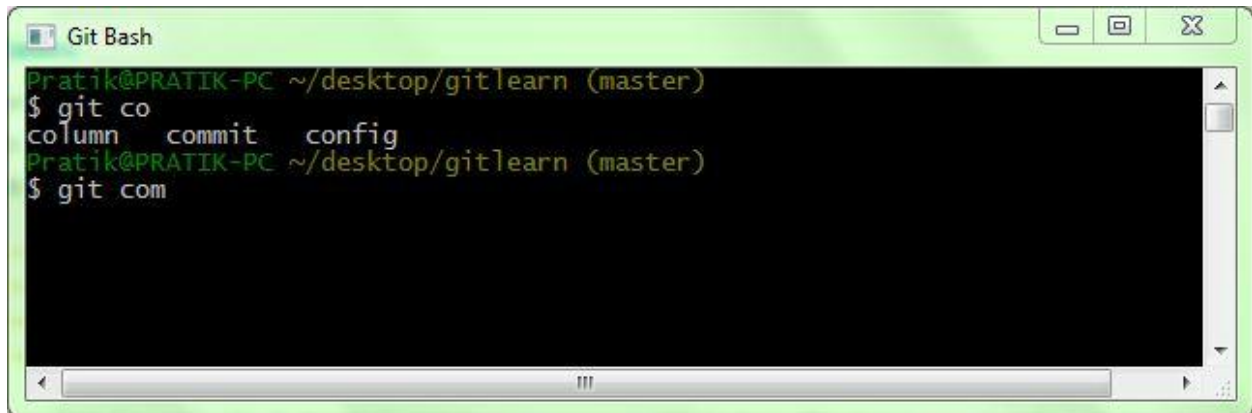
SHARING TAG

Git push does not transfer tags to remote servers we have to explicitly push git tags to remote servers using command:

git push origin v1.5

GIT MISCELLANEOUS

Git Tab for Auto-Completion



The screenshot shows a terminal window titled "Git Bash" with a light green border. The prompt is "Pratik@PRATIK-PC ~/desktop/gitlearn (master)". The user enters "\$ git co", and a list of suggestions appears: "column", "commit", and "config". The user then enters "\$ git com", and the terminal shows the first suggestion, "column", being selected.

Git Aliases

```
$ git config --global alias.co checkout
```

```
$ git config --global alias.br branch
```

```
$ git config --global alias.ci commit
```

```
$ git config --global alias.st status
```

Unsetting a Git Aliases

```
$ git config --global --unset alias.myAlias
```

REFERENCES

[SVN-Book](#)

[GIT- Book](#)

[GIT - Quick Reference](#)