

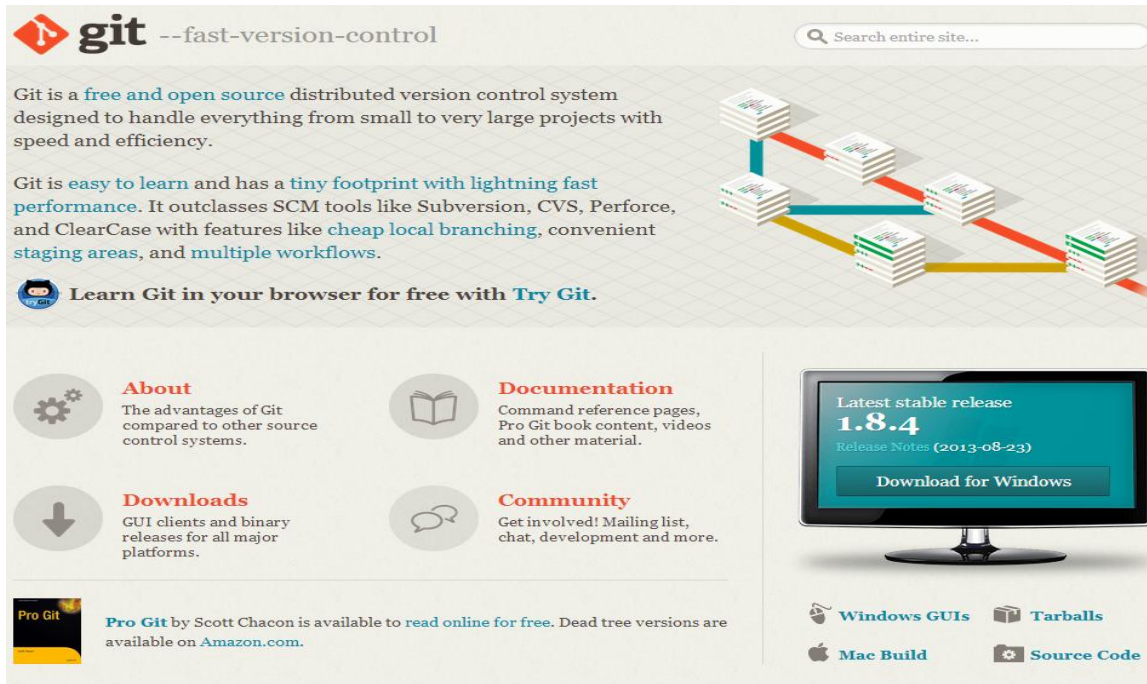
Basics of GIT

GIT is a distributed version control and file management system.

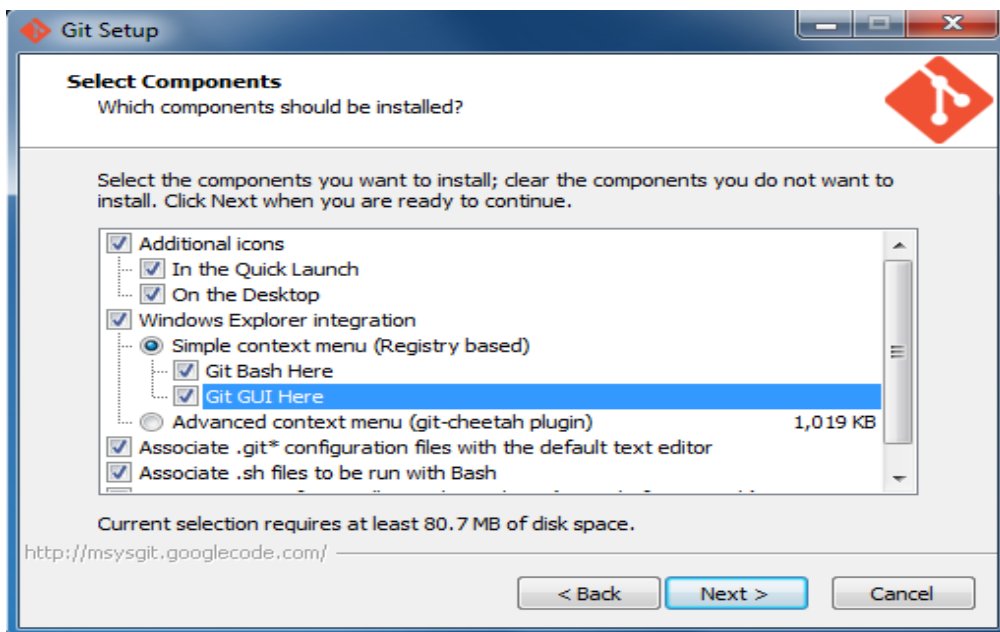
You can download the GIT from the following website:

<http://git-scm.com>

Once you open this website you will find this screen:



Run the downloaded EXE file and select the following components shown in the figure:

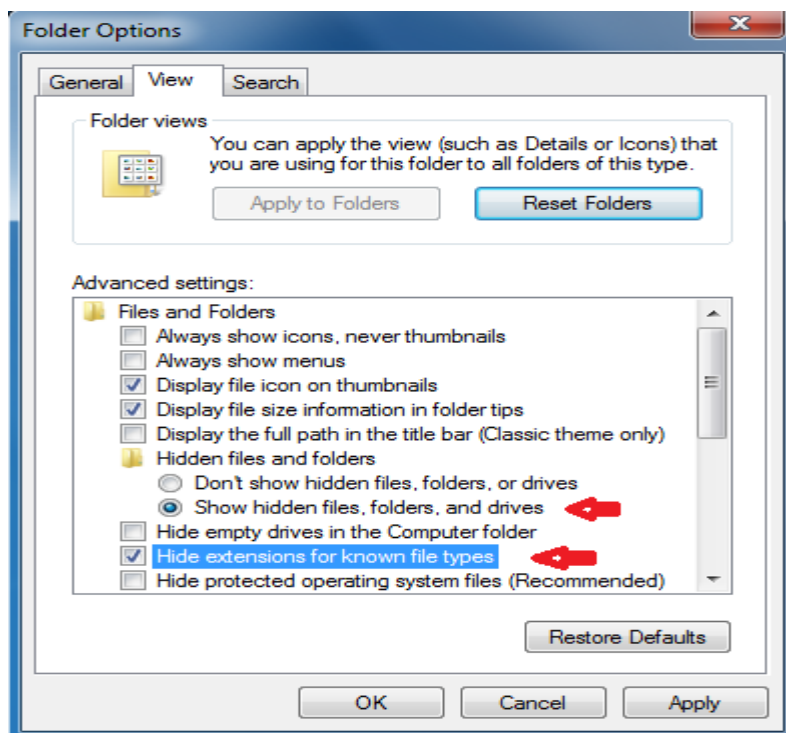


After finishing installation you will find the following icon on your desktop:

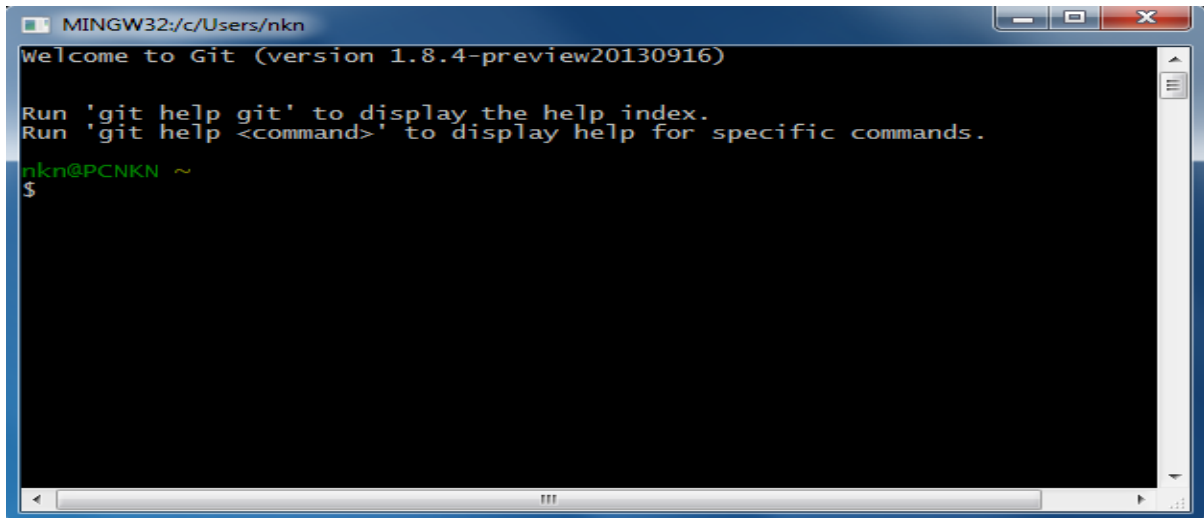


If you are doing GIT first time then before starting Git Shell you can change two settings to see all your hidden file and extensions:

Control Panel> Folder Options > View



Now open Git Shell window by double clicking the Git Shell icon on desktop then following screen will appear:



```
MINGW32:/c/Users/nkn
Welcome to Git (version 1.8.4-preview20130916)

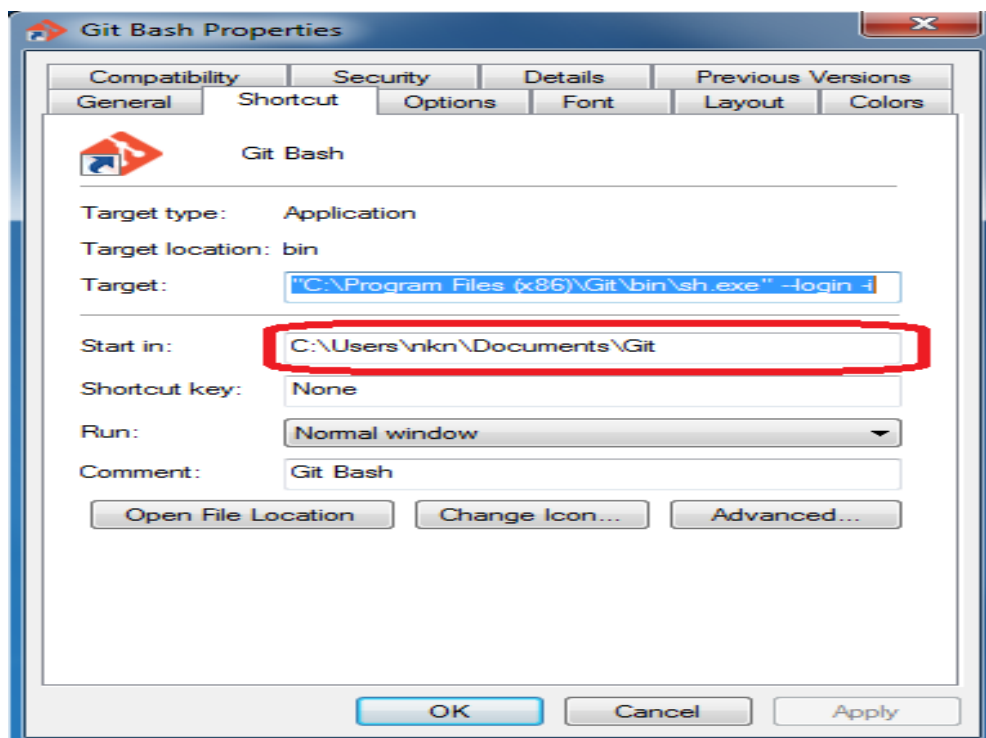
Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

nkn@PCNKN ~
$
```

Run the very first command “ls” to see the list of all files and folders in the current directory (here **nkn**):

\$ ls

You can also set the path of directory where you want to store Git projects by right clicking on Git Shell icon and change the path as shown in the figure:



Once you change the path you will find the changed path on the Bash Shell prompt. Check the files and folders in **Git** folder using “ls” command:

\$ ls

Perform two configuration settings about username and email which is important because they are always recorded with every “commit” operation of snapshot.

\$ git config --global user.name “nkn”

\$ git config --global user.email “nkn@aber.ac.uk”

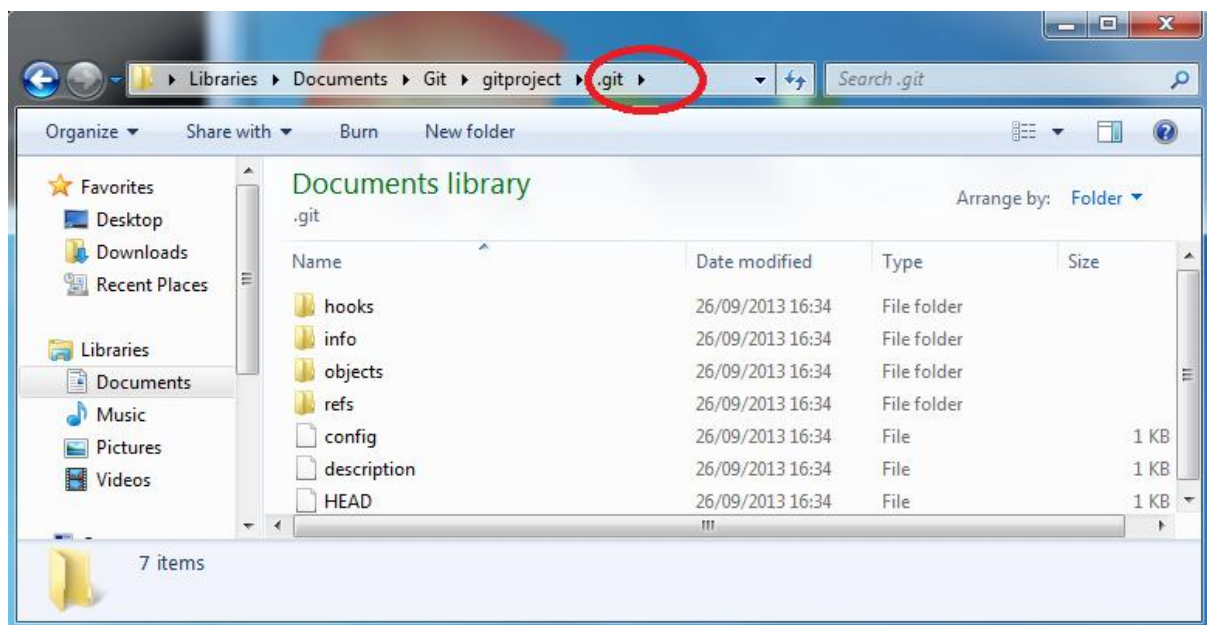
Now create a new folder “gitproject” inside the selected **Git** folder to keep the complete project in it. Also create two text files: “test1.txt” and “test2.txt” inside the “gitproject” to perform the version control operation and check with “ls” command:

\$ ls

Run the following command to make the “gitproject” folder as a git repository:

\$ git init gitproject

Got to the “gitproject” and click on the hidden folder “.git” to see all the required stuff necessary for version control as shown in the figure:



Change the directory to “gitproject” using command:

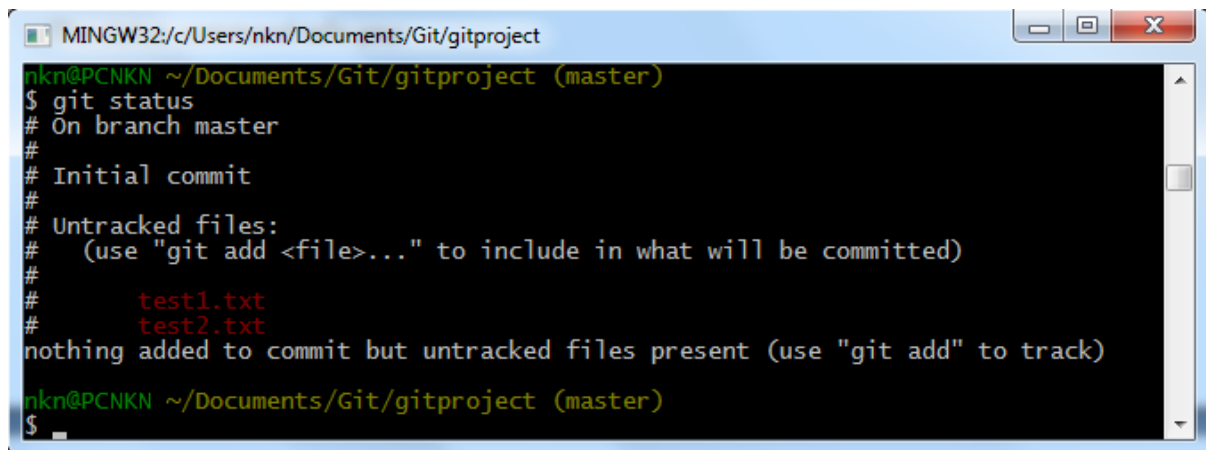
\$ cd gitproject

Once you change the director you will see the message “(master)” next to the “gitproject” directory that tells it is the main directory for our version control. See the two text files:

\$ ls

Run the following command:

\$ git status

A terminal window titled 'MINGW32:/c/Users/nkn/Documents/Git/gitproject' showing the output of the 'git status' command. The output indicates an initial commit on the master branch with two untracked files, 'test1.txt' and 'test2.txt', shown in red text. The prompt is 'nkn@PCNKN ~/Documents/Git/gitproject (master) \$'.

The status command tells you:

- What branch you're on
- What files have changed
- What files are not tracked
- Hints on what to do next

Here it is telling that both of our files are untracked so they appeared in red colour.

Now we can tell Git what file we want track or version control by using the “add” command

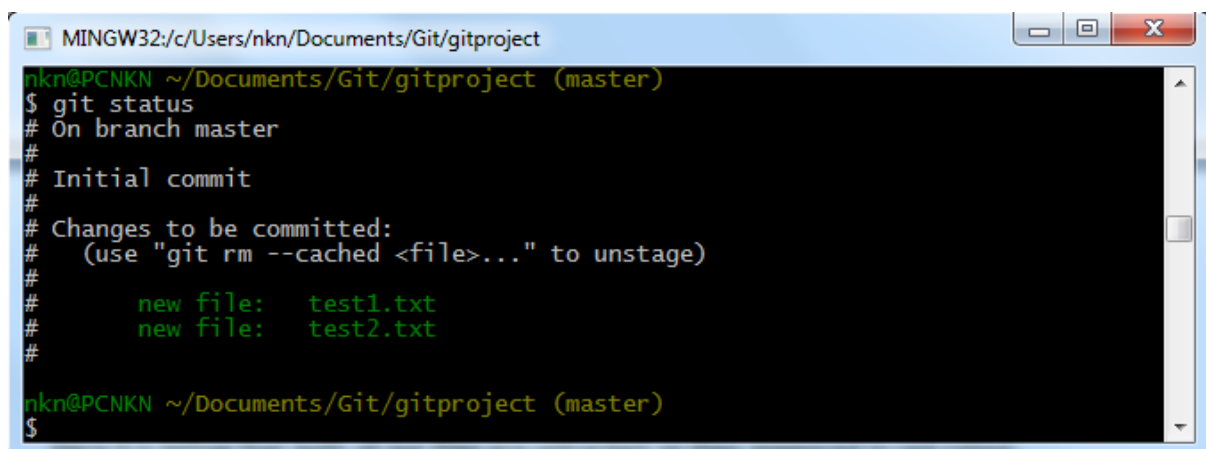
```
$ git add test1.txt
```

We can also mention all file using the command:

```
$ git add .
```

The “add” command adds the entire file to the Git folder-“gitproject” (staging area).

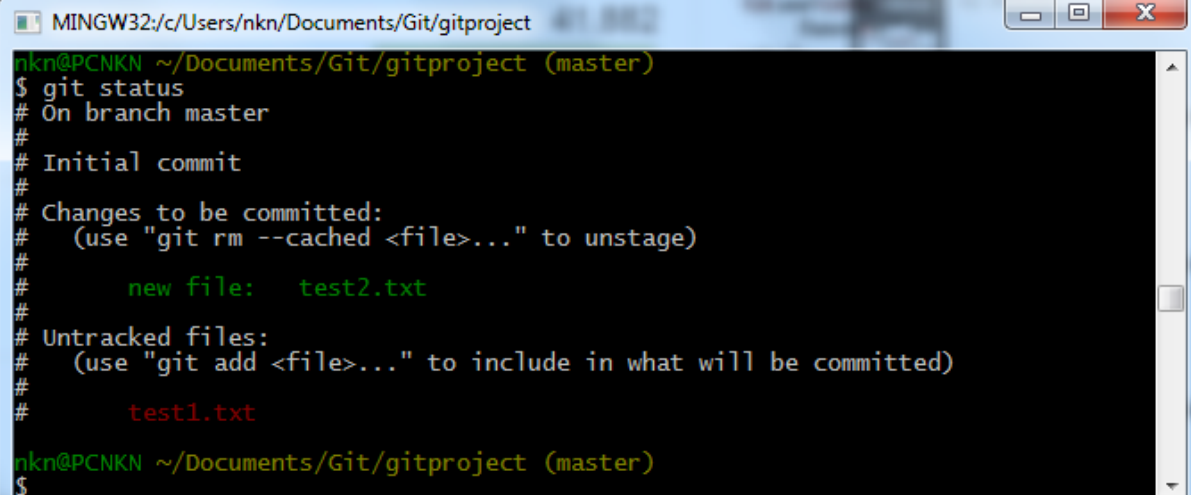
Again check the status of these files you will find that files colour is changed to green which reflects that these files are added to the Git folder-“gitproject” (staging area).

A terminal window titled 'MINGW32:/c/Users/nkn/Documents/Git/gitproject' showing the output of the 'git status' command after adding the files. The output indicates that 'test1.txt' and 'test2.txt' are now staged for commit, shown in green text. The prompt is 'nkn@PCNKN ~/Documents/Git/gitproject (master) \$'.

We haven't committed any file till now so we can remove these files from the caching (staging) area using the following command:

```
$ git rm --cached test1.txt
```

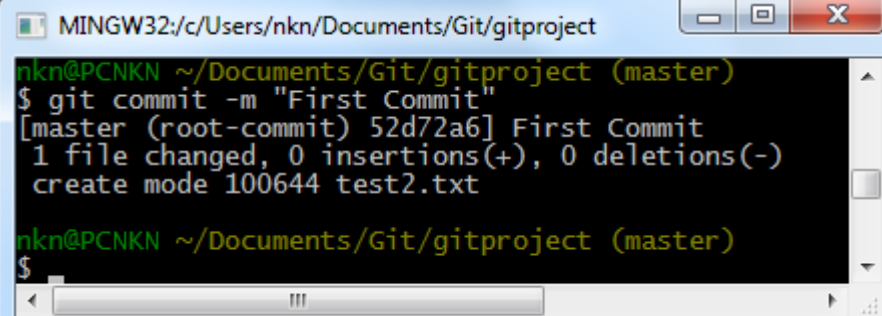
See the result on the screen you will find that "test1.txt" file became untracked again as shown in the figure.

A terminal window titled 'MINGW32/c/Users/nkn/Documents/Git/gitproject' showing the output of the 'git status' command. The output indicates an initial commit on the master branch with one new file, 'test2.txt', staged for commit. 'test1.txt' is listed as an untracked file.

```
mingw32/c/Users/nkn/Documents/Git/gitproject
nkn@PCNKN ~/Documents/Git/gitproject (master)
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   test2.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       test1.txt
nkn@PCNKN ~/Documents/Git/gitproject (master)
$
```

Make the first commit to record the snapshot to your project (working directory/ local repository).

```
$ git commit -m "First Commit"
```

A terminal window titled 'MINGW32/c/Users/nkn/Documents/Git/gitproject' showing the output of the 'git commit -m "First Commit"' command. The output shows a successful commit on the master branch with the message 'First Commit'. It indicates that 1 file changed, with 0 insertions and 0 deletions, and that 'test2.txt' was created with mode 100644.

```
mingw32/c/Users/nkn/Documents/Git/gitproject
nkn@PCNKN ~/Documents/Git/gitproject (master)
$ git commit -m "First Commit"
[master (root-commit) 52d72a6] First Commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 test2.txt
nkn@PCNKN ~/Documents/Git/gitproject (master)
$
```

Let's check status again:

```
$ git status
```

Which shows test1.txt is still untracked. Track this file using the following command:

```
$ git add test1.txt
```

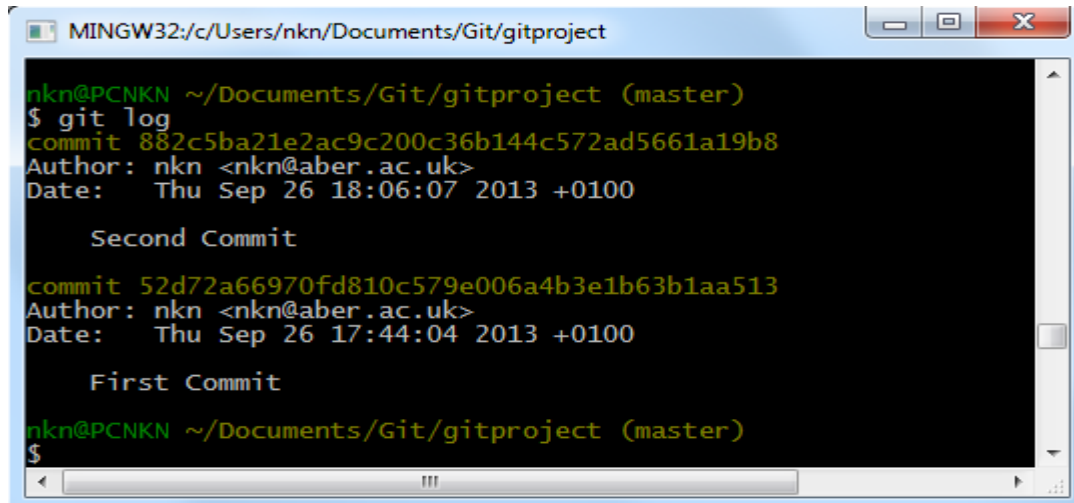
Perform commit again to record snapshot to local directory/repository. It is good practice to use "commit" after "add".

```
$ git commit -m "First Commit"
```

Note: Another alternate way for both commands is use "\$ git commit -a" command.

After you have created several commits, or if you have cloned a repository with an existing commit history, you'll probably want to look back to see what has happened. The most basic and powerful tool to do this is the git log command.

```
$ git log
```



```
MINGW32:/c/Users/nkn/Documents/Git/gitproject
nkn@PCNKN ~/Documents/Git/gitproject (master)
$ git log
commit 882c5ba21e2ac9c200c36b144c572ad5661a19b8
Author: nkn <nkn@aber.ac.uk>
Date: Thu Sep 26 18:06:07 2013 +0100

    Second Commit

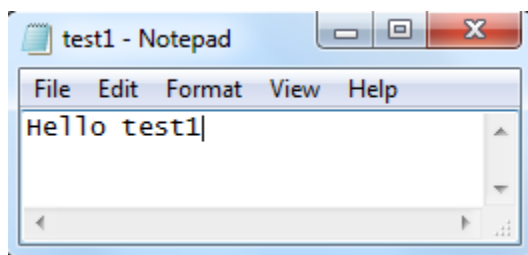
commit 52d72a66970fd810c579e006a4b3e1b63b1aa513
Author: nkn <nkn@aber.ac.uk>
Date: Thu Sep 26 17:44:04 2013 +0100

    First Commit

nkn@PCNKN ~/Documents/Git/gitproject (master)
$
```

It tells you the details about all commits such as authors, emails, dates, etc.

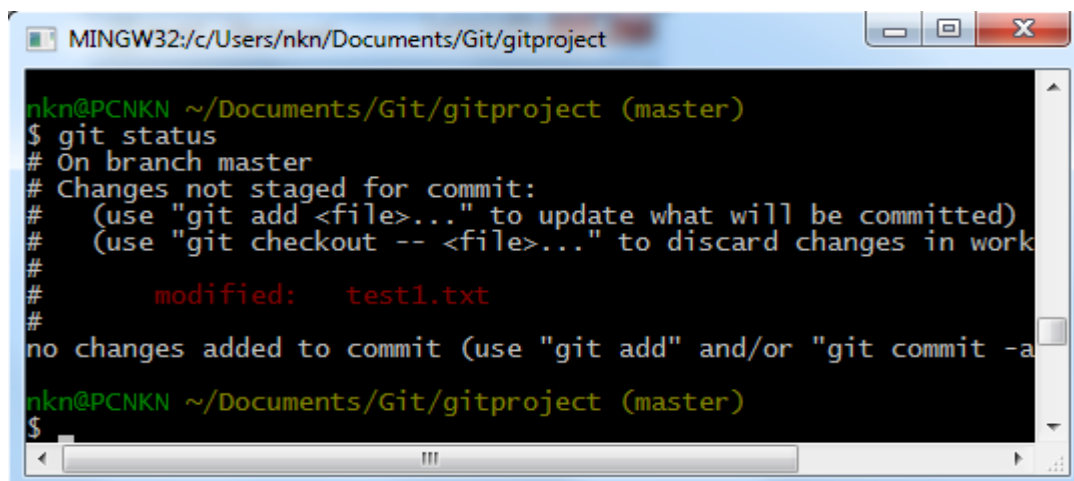
Now make the changes to the file "test1.txt" to see how version control should happen:



Check the status after modification:

```
$ git status
```

It will give you the message "modified: test1.txt" as shown in the figure:



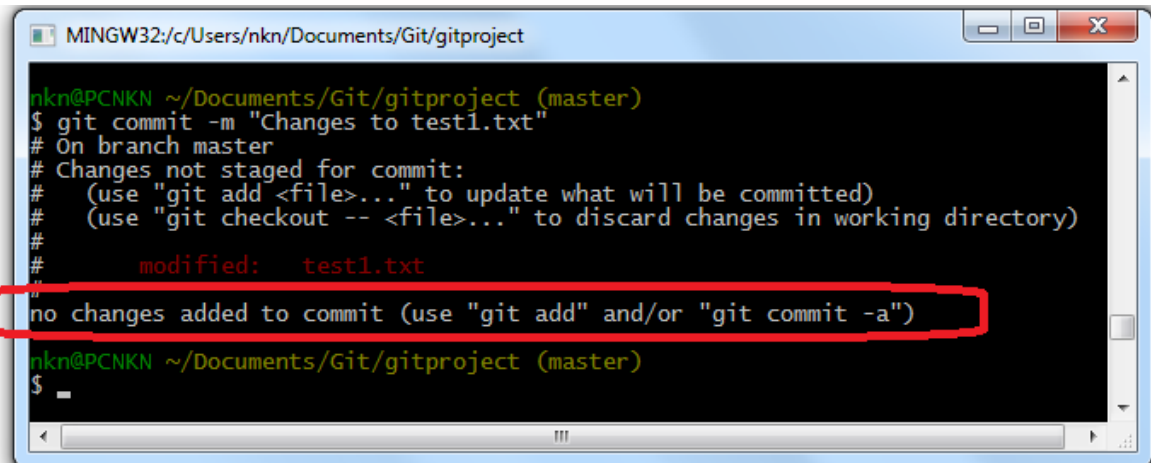
```
MINGW32:/c/Users/nkn/Documents/Git/gitproject
nkn@PCNKN ~/Documents/Git/gitproject (master)
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in work)
#
#       modified:   test1.txt
#
no changes added to commit (use "git add" and/or "git commit -a")

nkn@PCNKN ~/Documents/Git/gitproject (master)
$
```


Perform commit to record this change to the local directory/repository.

```
$ git commit -m "Changes to test1.txt"
```

This **commit** wouldn't execute because change should always go to the caching/staging area (using **add** command) before commit. Actually **commit** command always clears the caching/staging area.

A terminal window titled 'MINGW32:/c/Users/nkn/Documents/Git/gitproject' showing the output of a git commit command. The prompt is 'nkn@PCNKN ~/Documents/Git/gitproject (master)'. The command entered is '\$ git commit -m "Changes to test1.txt"'. The output shows: '# 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: test1.txt', '#', and then a red box highlights the message 'no changes added to commit (use "git add" and/or "git commit -a")'. The prompt returns to '\$ -'.

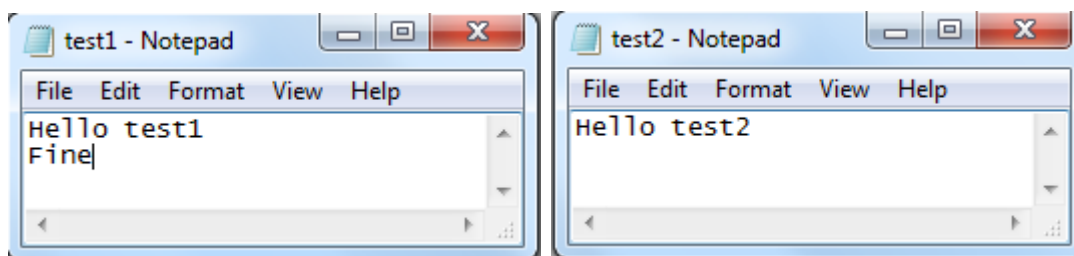
Write the following commands:

```
$ git add test1.txt
```

```
$ git commit -m "Changes to test1.txt"
```

```
$ git log
```

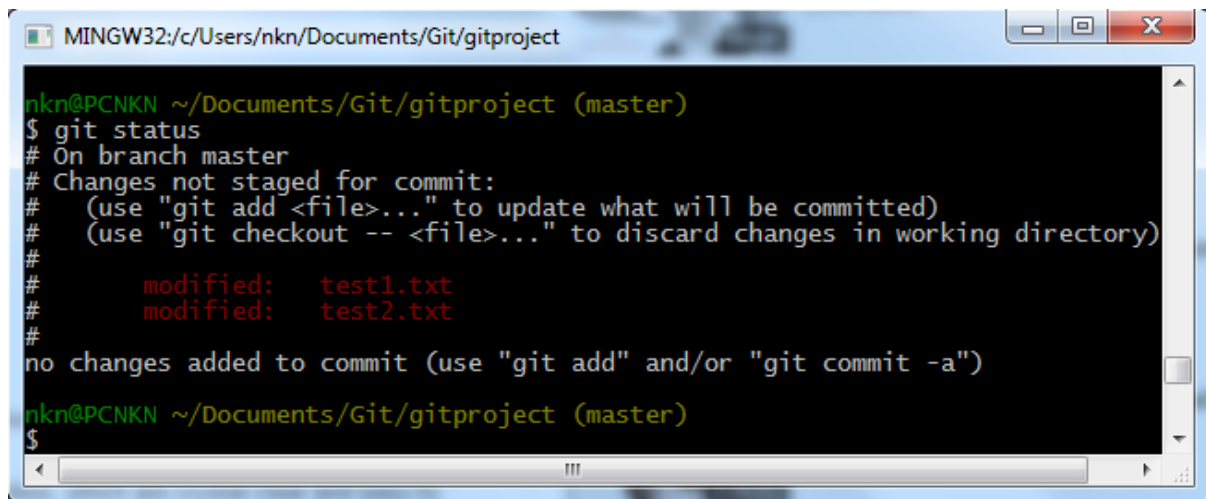
Now modify both files as shown in figures:



Run the status command:

```
$ git status
```

It will tell you that both files are modified as shown in figure:

A terminal window titled 'MINGW32:/c/Users/nkn/Documents/Git/gitproject' showing the output of the 'git status' command. The output indicates that the user is on the 'master' branch and that there are changes not staged for commit. These changes are listed as 'modified: test1.txt' and 'modified: test2.txt' in red text. The terminal also provides instructions on how to stage changes using 'git add' or discard them using 'git checkout --'.

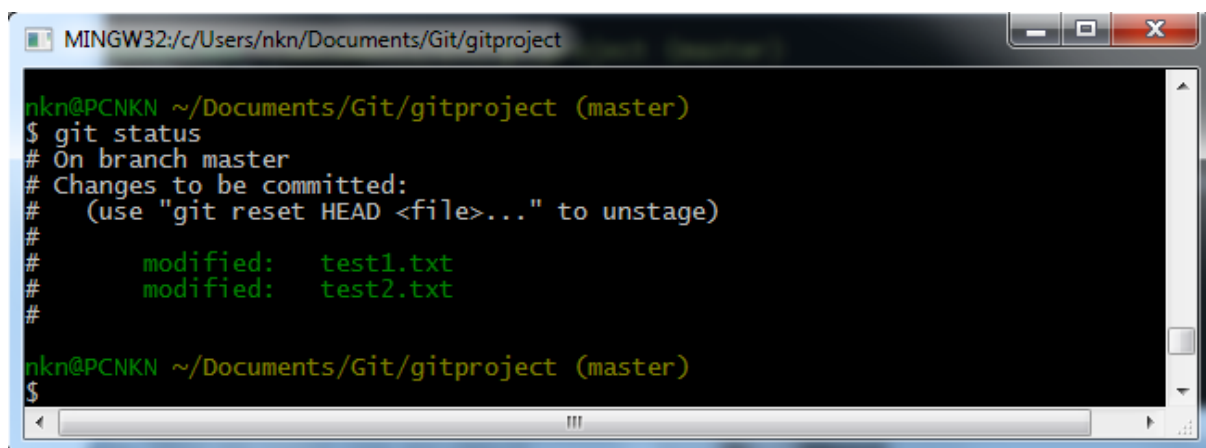
```
mingw32:/c/Users/nkn/Documents/Git/gitproject
nkn@PCNKN ~/Documents/Git/gitproject (master)
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   test1.txt
#       modified:   test2.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
nkn@PCNKN ~/Documents/Git/gitproject (master)
$
```

Now add these two files to staging area:

```
$ git add .
```

```
$ git status
```

It has been tracking these files now so their colour is changed to green.

A terminal window titled 'MINGW32:/c/Users/nkn/Documents/Git/gitproject' showing the output of the 'git status' command after staging the files. The output now shows 'Changes to be committed:' in green text, with 'modified: test1.txt' and 'modified: test2.txt' also in green. The terminal provides instructions on how to unstage changes using 'git reset HEAD'.

```
mingw32:/c/Users/nkn/Documents/Git/gitproject
nkn@PCNKN ~/Documents/Git/gitproject (master)
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   test1.txt
#       modified:   test2.txt
#
nkn@PCNKN ~/Documents/Git/gitproject (master)
$
```

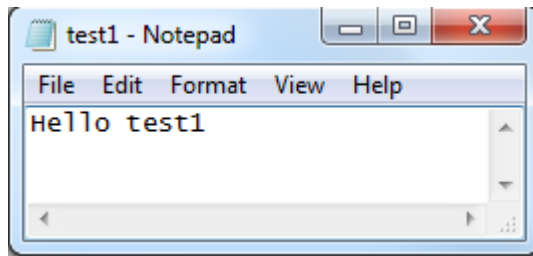
Finally run the commit command:

```
$ git commit -m "Changes to test1 and test2"
```

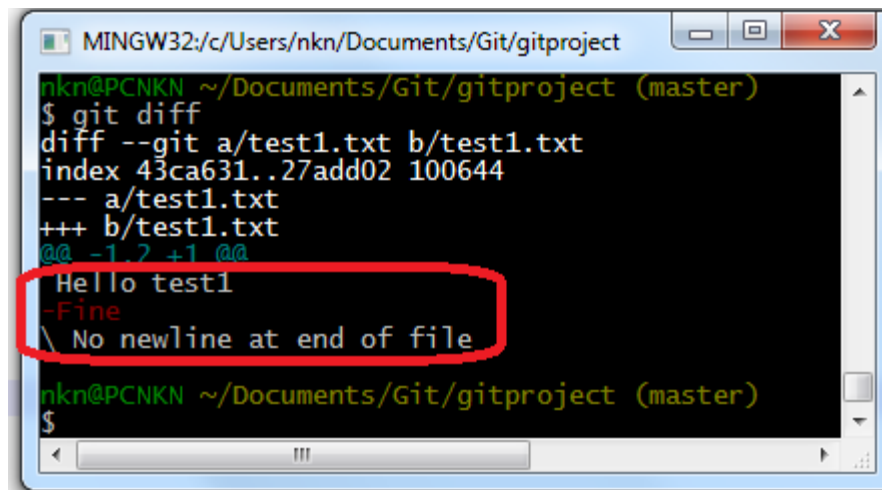
```
$ git log
```

It will show only those commits which happened after the recent commit. If the log list is too long then use **SHIFT+ZZ** shortcut to come back on \$ prompt.

Delete text "Fine" from the test1.txt file and run the following command:



\$ git diff



It will display what code or content you've changed in your project since the last commit that are not yet staged for the next commit snapshot.

\$ git add test1.txt

\$ git diff

This time it will show nothing because you staged the last changes.