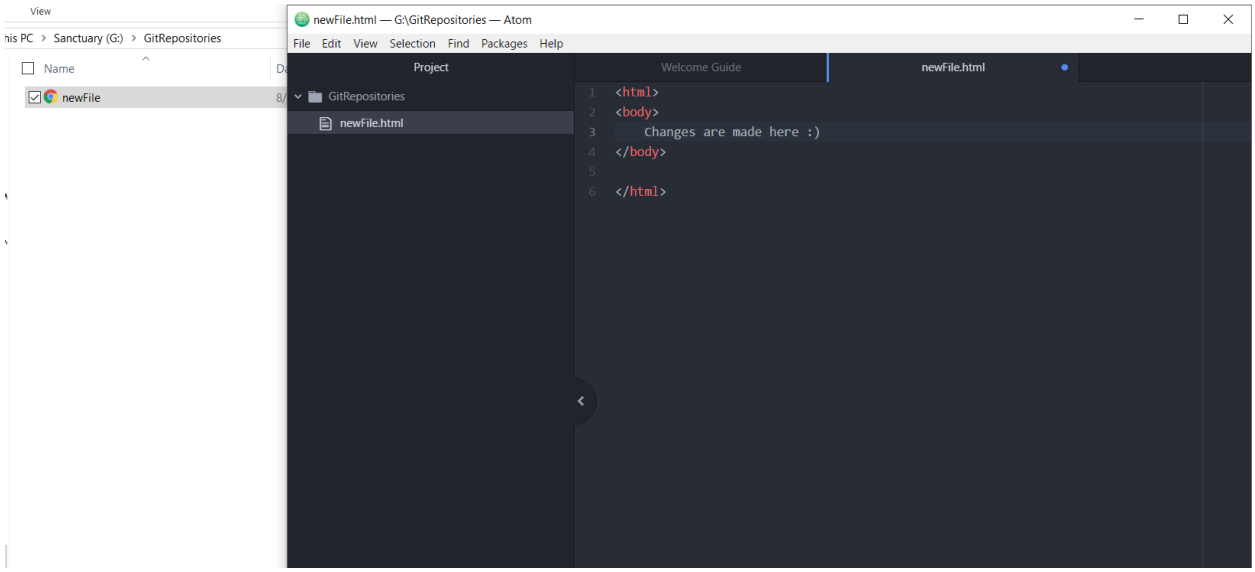
sudo apt-get install git.

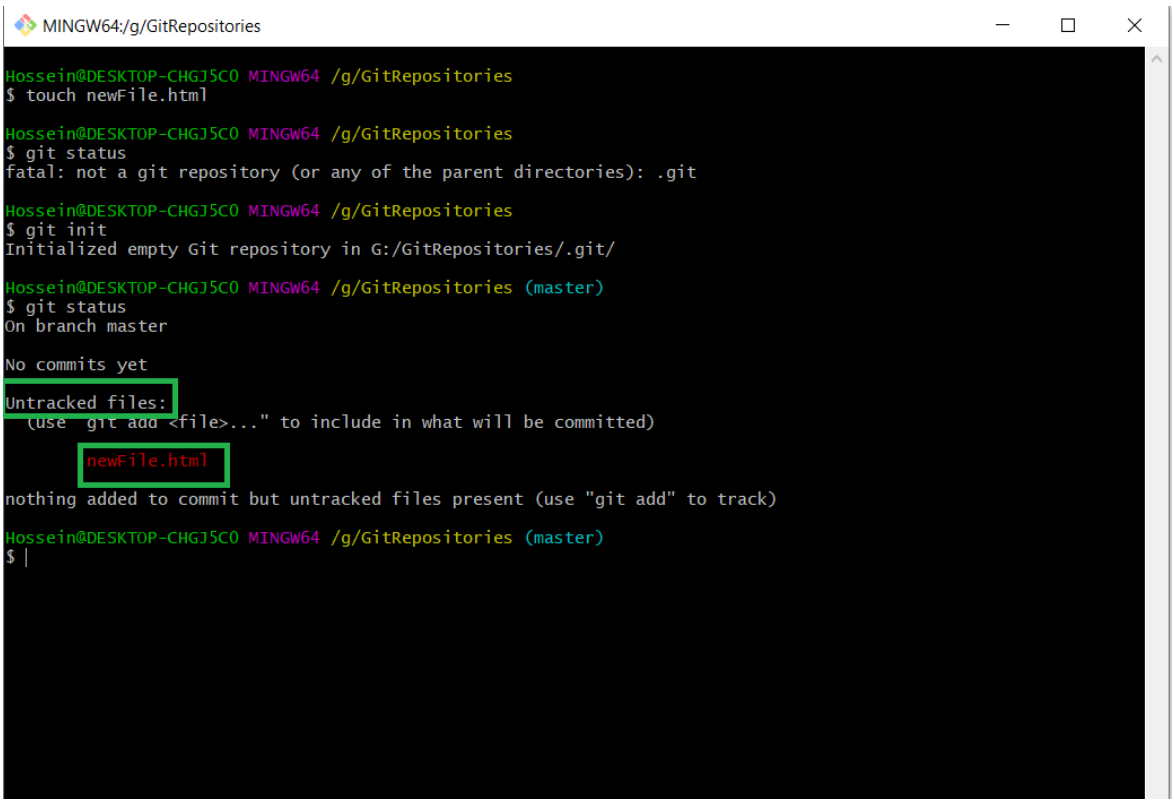
When We are in a directory and we want to start working with GIT, the first command is

$ git init which means from now on this directory is under control of git.

1) make a file -> change the file -> bring it to “stage mode” -> commit it.



$ git status



2) In order to make it staged, we say :

$ git add <file name>

$ git add . **-> adds all files that are not already added or changed somehow.**

$ git add –A **-> adds all files that are not already added or changed somehow.**

MINGW64:/g/GitRepositories

```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories (master)
$ git add newFile.html

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories (master)
$
```

3) Let's Tell git who we are and who is going to make changes to the file.

$ git config --global user.email "<emailAddress>"

OR

$ git config --global user.name "<Your name>"

```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories (master)
$ git config --global user.email "Hossein.dehghanipour1998@gmail.com"
```

4) when we make a change, git calls it an "Untracked File|change ".

$ git commit –m "*"

instead of the "*", we write some descriptions of what we have done (the changes) to the files.

```
$ git commit -m "Added Changes to the newFile.html"
 1 file changed, 6 insertions(+)
 create mode 100644 newFile.html
```

5) **$ git log** tells us what ever we have done to this git repository and who has made what changes.

```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories (master)
$ git log
commit a71810ed174106f8931b8b43cc082bd1cc826d91 (HEAD -> master)
Author: Hossein Dehghanipour <Hossein.dehghanipour1998@gmail.com>
Date:   Fri Aug 30 21:00:09 2019 +0430

    Made new Changes to the files

commit d01ff3a58f33d54d0db42929d684485e337a3176
Author: Hossein Dehghanipour <Hossein.dehghanipour1998@gmail.com>
Date:   Fri Aug 30 20:54:03 2019 +0430

    Added Changes to the newFile.html

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories (master)
$
```

Head is where we are **right now** which is usually the last repository we are working on.

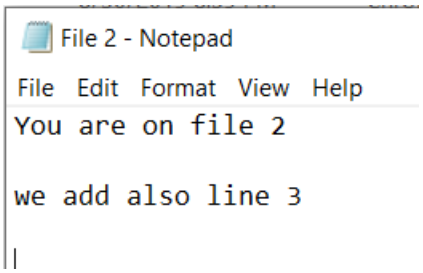
6) when we make some files at the “Staged” position but we are not sure if we have made the correct changes. so we use **$ git diff --staged** to see the changes made to the files.

```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        File 2.txt

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

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories (master)
$ git add -A

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   File 2.txt


Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories (master)
$ git diff --staged
diff --git a/File 2.txt b/File 2.txt
new file mode 100644
index 0000000..4207924
--- /dev/null
+++ b/File 2.txt
@@ -0,0 +1 @@
+You are on file 2

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories (master)
$
```

7) Assume that we have staged a file but we regret it now. there is a way to **Undo** our staging.
$ git reset “<FileName>”

```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories (master)
$ git reset "File 2.txt"

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        File 2.txt

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

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories (master)
$
```

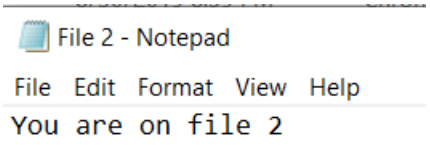
8) assume that we have made some changes to a file but we want to **Undo the Changes** and revert the changes made to the file.
$ git checkout -- <fileName>



-> New changes made.

that actually brings the file to it's **latest commit**.

```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories (master)
$ git checkout -- "File 2.txt"

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories (master)
$ git status
On branch master
nothing to commit, working tree clean

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories (master)
$
```



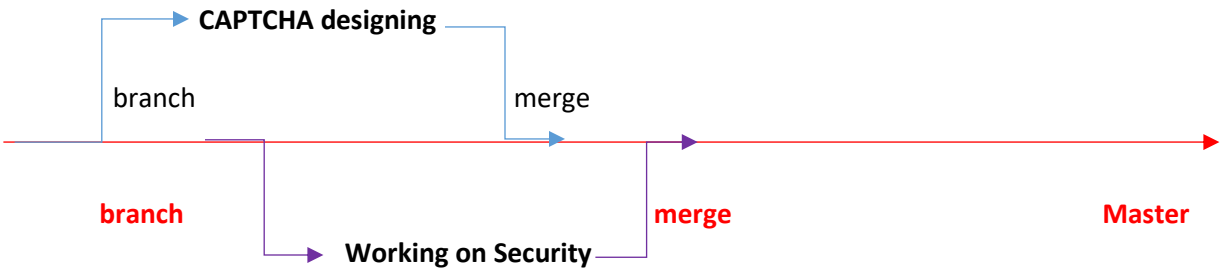
-> those new changes are discarded now.

9) we can see the changes made to the files by : **$ git diff HEAD** and also see our current head.

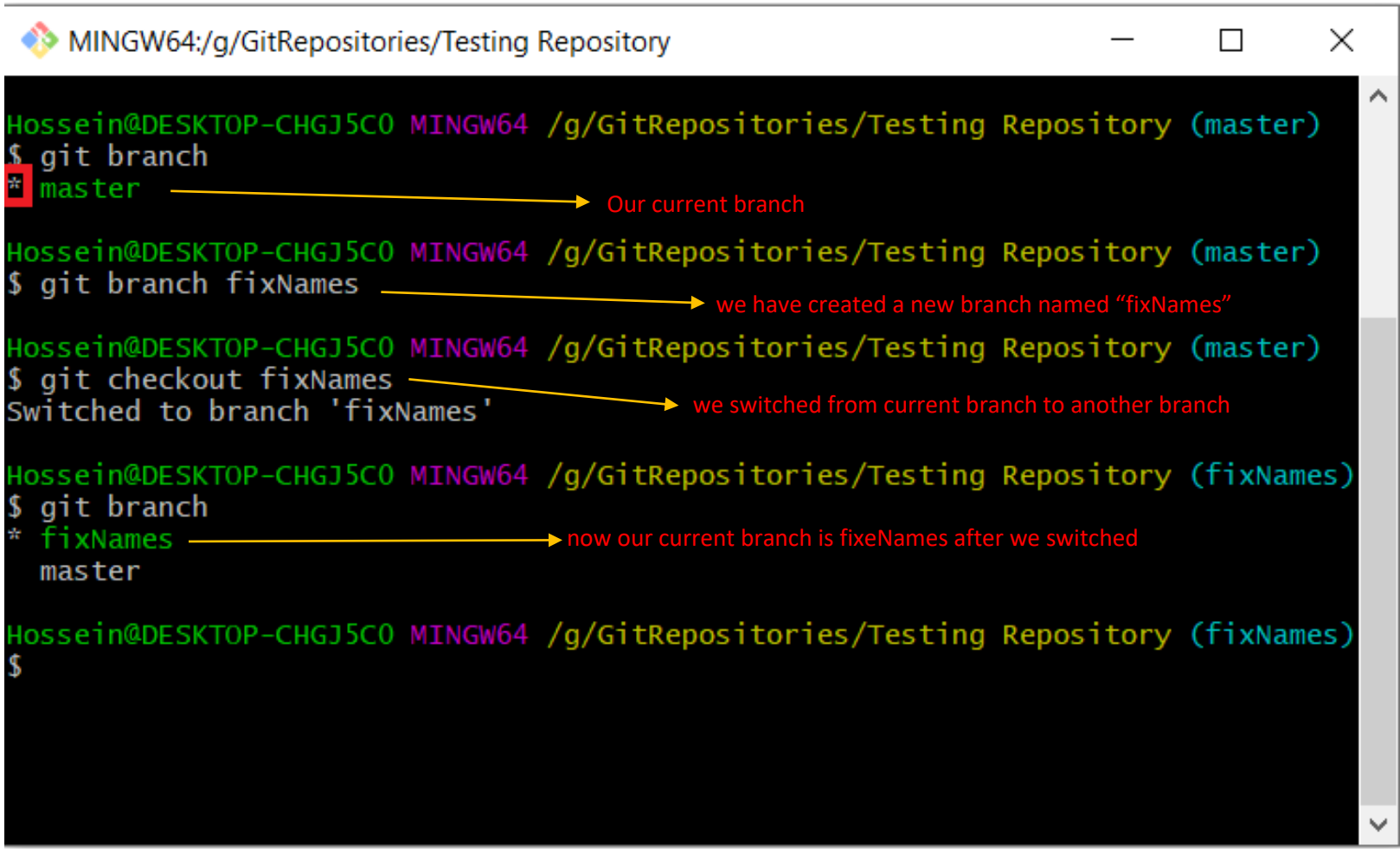
```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories (master)
$ git diff head
diff --git a/File 2.txt b/File 2.txt
index eb61757..bed351f 100644
--- a/File 2.txt
+++ b/File 2.txt
@@ -1,5 +1,3 @@
 You are on file 2

-we add also line 3
-
```

Branching & Merging :

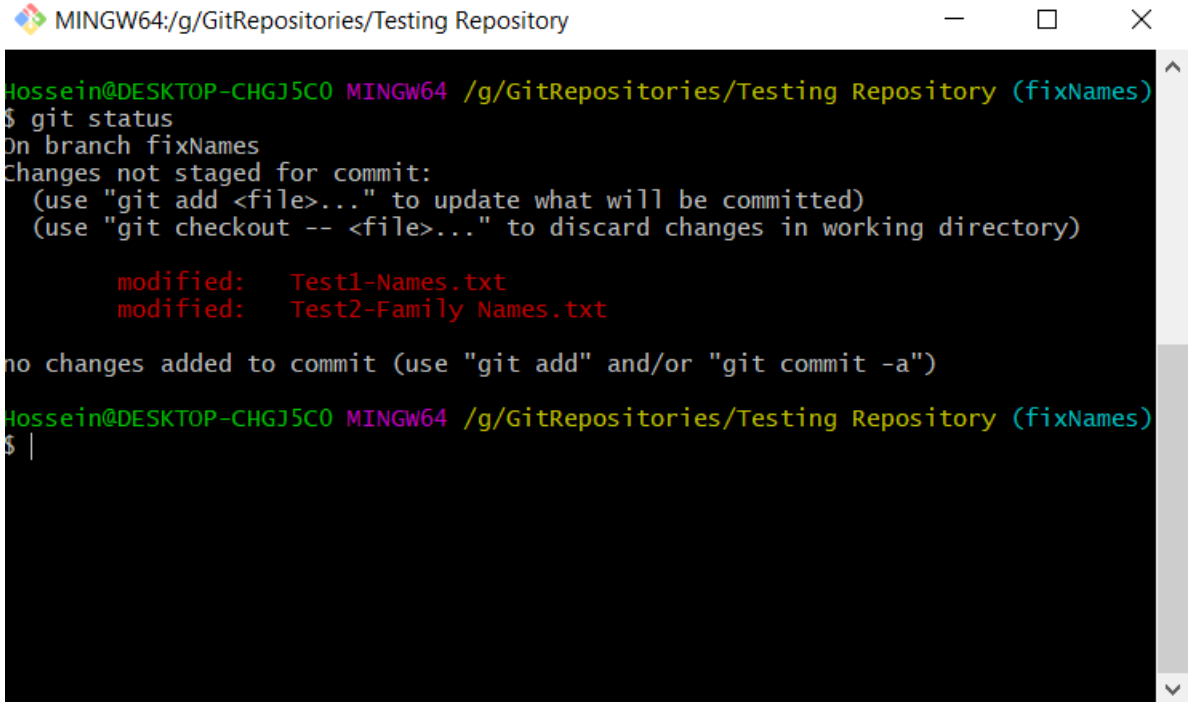
Assume that we are working on a project and each member of the team has taken responsibility for a different part of the project. One is taking control over “Security”, one is creating the “UI” and another guy is going to create the part of our application that asks for “CAPTCHA”. So if we assume that the red line is our Master branch (Main branch), guy #1 is going to work on the CAPTCHA by making his own branch and guy #2 is also going to work on the security simultaneously and they all Merge after they have done their job. The key point in here is that they are doing different things and there might not be an overlap in their tasks.

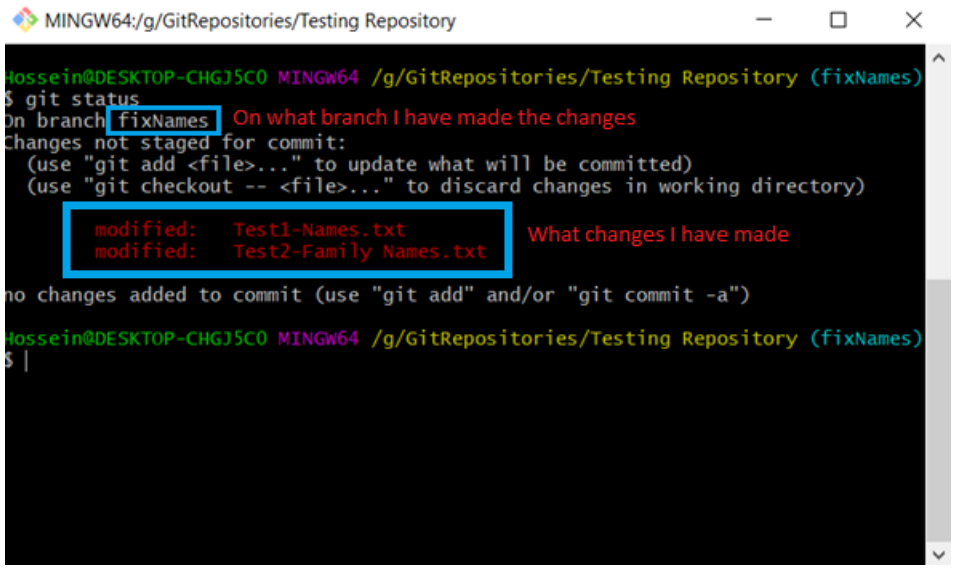


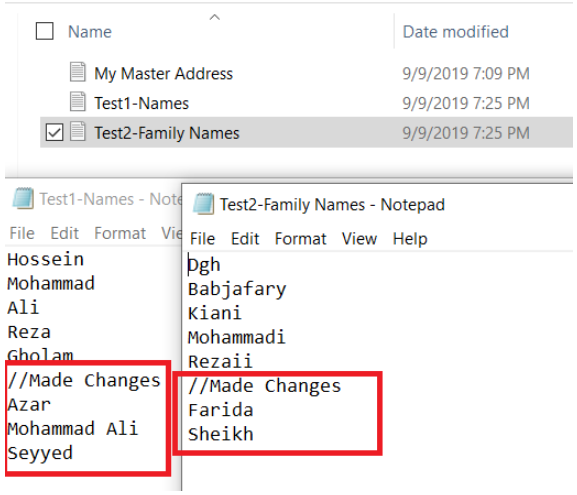
$ git branch -> **Shows our current branch that we are in**
$ git branch <branch name>
# you should have a commit on the master in order to create a new branch.



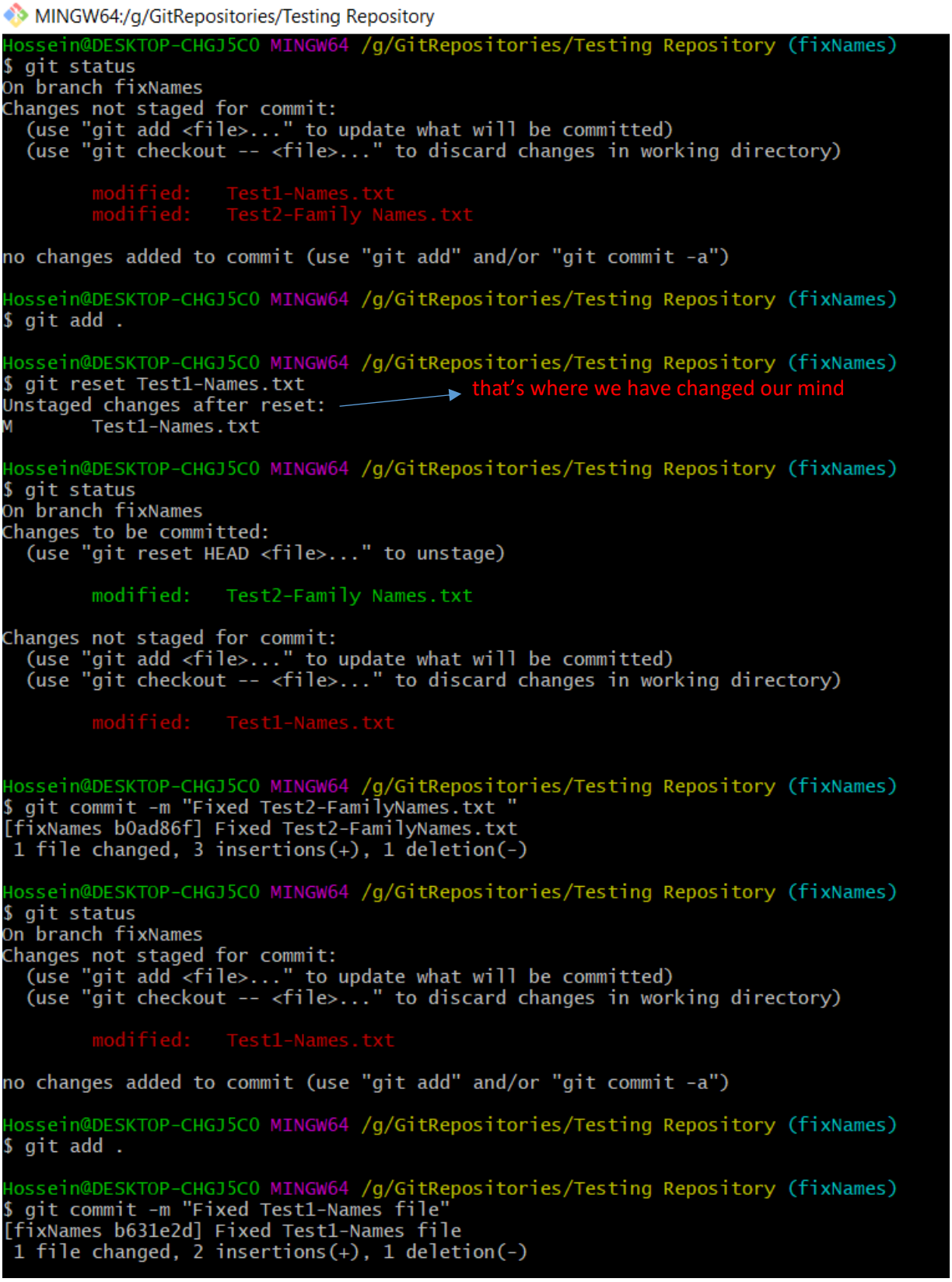
now if I add/change some files and then get the status that's what I will see :







some times we make changes to multiple files but they are not in the same category. We make separate commits for different categories. so we may have already added all files. What we must do is that we "reset" the unwanted files in the current commit and then we add them to another commit.



```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (fixNames)
$ git commit -m "Fixed Test1-Names file"
[fixNames b631e2d] Fixed Test1-Names file
 1 file changed, 2 insertions(+), 1 deletion(-)

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (fixNames)
$
```

Now assume that we switch to our master branch. as we know we have made some changes in our other branch “fixNames”. In order to have the changes on out master branch too, we have to merge them :

```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (fixNames)
$ git checkout master
Switched to branch 'master'

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$ git status
On branch master
nothing to commit, working tree clean

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$ git branch
  fixNames
* master

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$ git merge fixNames
Updating a01bf88..b631e2d
Fast-forward
 Test1-Names.txt        | 7 ++++++-
 Test2-Family Names.txt | 5 +++++
 2 files changed, 11 insertions(+), 1 deletion(-)

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$
```

```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$ git log
commit b631e2da23891ac9b6e219bcc36c428eec256f06 (HEAD -> master, fixNames)
Author: Hossein Dehghanipour <Hossein.dehghanipour1998@gmail.com>
Date:   Mon Sep 9 19:42:49 2019 +0430

    Fixed Test1-Names file

commit b0ad86f997f2ebb942198076b9cf223b6db2e293
Author: Hossein Dehghanipour <Hossein.dehghanipour1998@gmail.com>
Date:   Mon Sep 9 19:42:24 2019 +0430

    Fixed Test2-FamilyNames.txt

commit 540506b45a7fd31e129e0f722b8dbc6be95b53b4
Author: Hossein Dehghanipour <Hossein.dehghanipour1998@gmail.com>
Date:   Mon Sep 9 19:33:55 2019 +0430

    Made Changes to Names and FamilyNames

commit a01bf88913f2639780e3d2003c13b0fe4e8cc27e
Author: Hossein Dehghanipour <Hossein.dehghanipour1998@gmail.com>
Date:   Mon Sep 9 19:15:17 2019 +0430

    Added test files

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$
```

If we want to delete a file from git, we write **$ git rm <file name>** this command deletes the following file from both git and file system.

I can also use **LS** command in order to see the files in my branch.

```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$ ls
'My Master Address.txt'   Test1-Names.txt  'Test2-Family Names.txt'

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$ git rm Test1-Names.txt
rm 'Test1-Names.txt'

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$ ls
'My Master Address.txt'  'Test2-Family Names.txt'

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    Test1-Names.txt


Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$
```

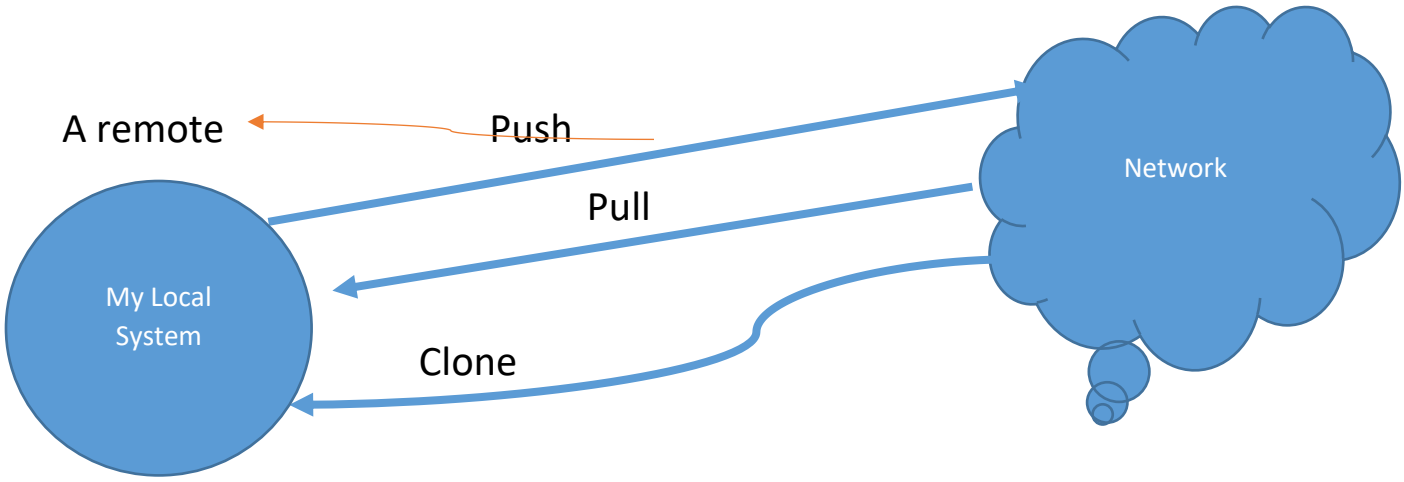
after we have done with our branch and we have merged it, we can delete the branch. so we use the command **$ git branch –d <branch name>.**

```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$ git branch
  fixNames
* master

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$ git branch -d fixNames
Deleted branch fixNames (was b631e2d).

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$ git branch
* master

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$
```

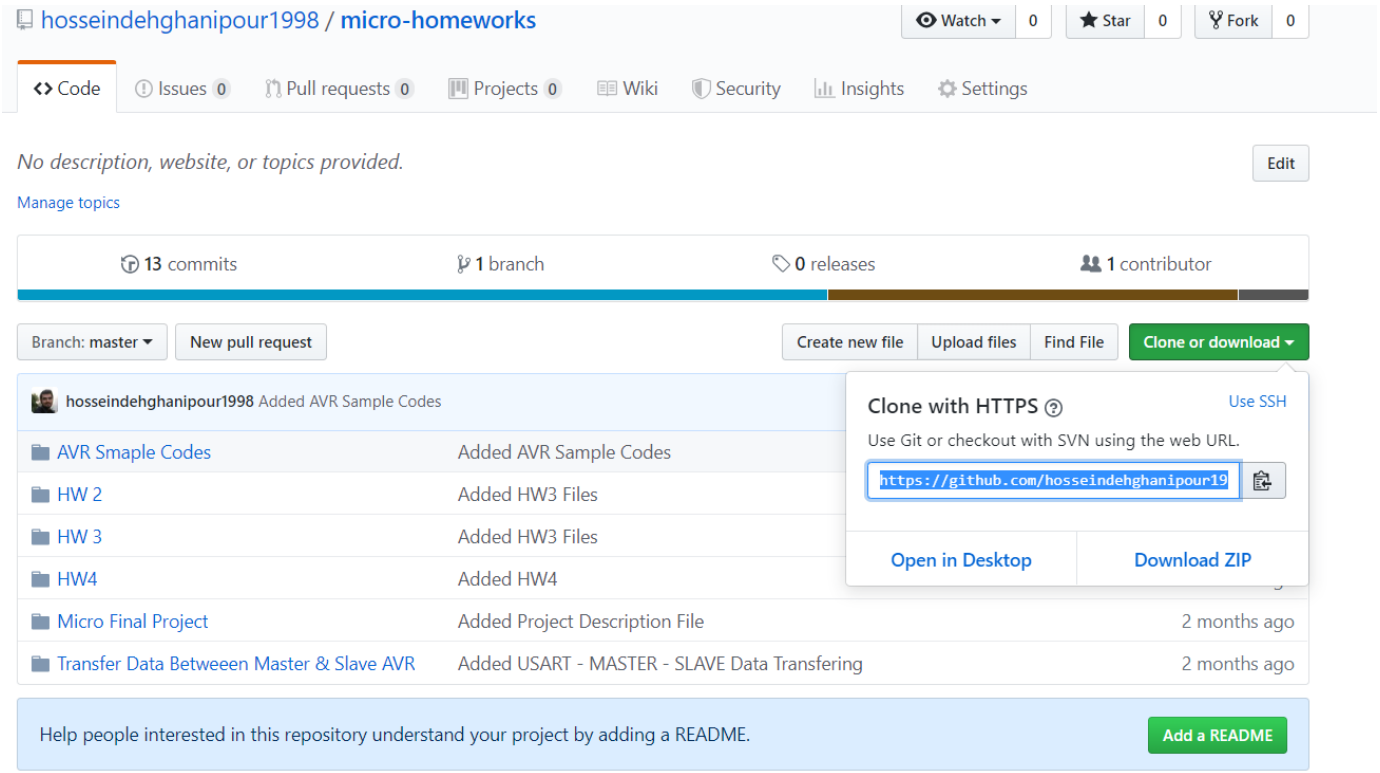
but it doesn’t show us that we have deleted a branch :

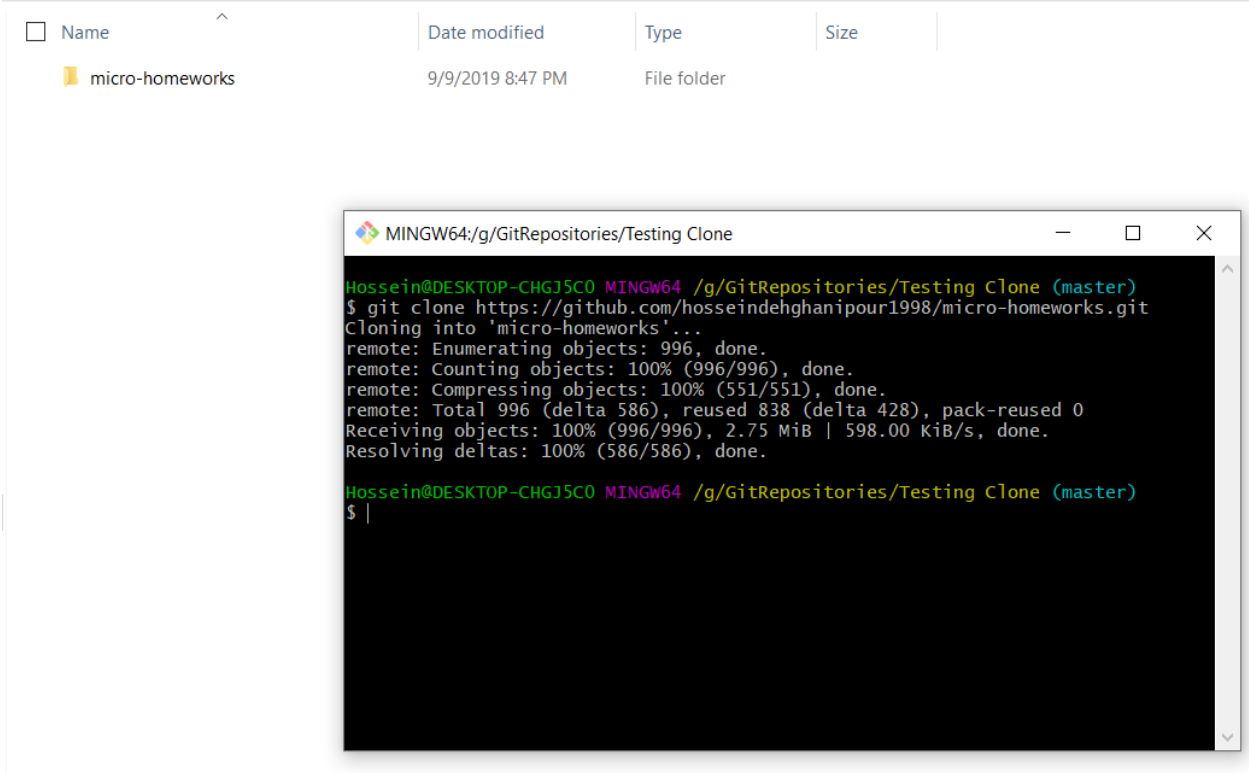
```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$ git branch
  fixNames
* master

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$ git branch -d fixNames
Deleted branch fixNames (was b631e2d).

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$ git branch
* master

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$ git log
commit ab7031b12cb59f36a4d9ef8cb79a2b83ba1c9286 (HEAD -> master)
Author: Hossein Dehghanipour <Hossein.dehghanipour1998@gmail.com>
Date:   Mon Sep 9 20:30:55 2019 +0430

    Deleted a file

commit b631e2da23891ac9b6e219bcc36c428eec256f06
Author: Hossein Dehghanipour <Hossein.dehghanipour1998@gmail.com>
Date:   Mon Sep 9 19:42:49 2019 +0430

    Fixed Test1-Names file

commit b0ad86f997f2ebb942198076b9cf223b6db2e293
Author: Hossein Dehghanipour <Hossein.dehghanipour1998@gmail.com>
Date:   Mon Sep 9 19:42:24 2019 +0430

    Fixed Test2-FamilyNames.txt

commit 540506b45a7fd31e129e0f722b8dbc6be95b53b4
Author: Hossein Dehghanipour <Hossein.dehghanipour1998@gmail.com>
Date:   Mon Sep 9 19:33:55 2019 +0430

    Made Changes to Names and FamilyNames

commit a01bf88913f2639780e3d2003c13b0fe4e8cc27e
Author: Hossein Dehghanipour <Hossein.dehghanipour1998@gmail.com>
Date:   Mon Sep 9 19:15:17 2019 +0430

    Added test files

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Repository (master)
$
```

Remotes : Origin



Clone : copies (downloads) all files from the system into your local pc.





$ git clone <the address copied from git >

$ git push origin master

when we clone a project from github, the remote set to the master of the project is usually “origin”.

$ git pull origin master

if we have a public repository, we can pull from it but pushing to a repository requires the owner’s permission.

we can add a remote to our directory. we can make remote to any directory from any other directory. If the directory doesn’t exist, we will face an error that tells us the path/URL we are aiming for doesn’t exist as a GIT repository.

our projects can have more than one remotes. We can push our project onto two different servers or pull from two different servers. for example, as a new commit is made we can push it to both gitlab and github simultaneously with two different remotes pointing to different servers.

$ git remote add <URL> -> creates a new remote to the porject

$ git remote –v -> shows us all the available remotes in the project.

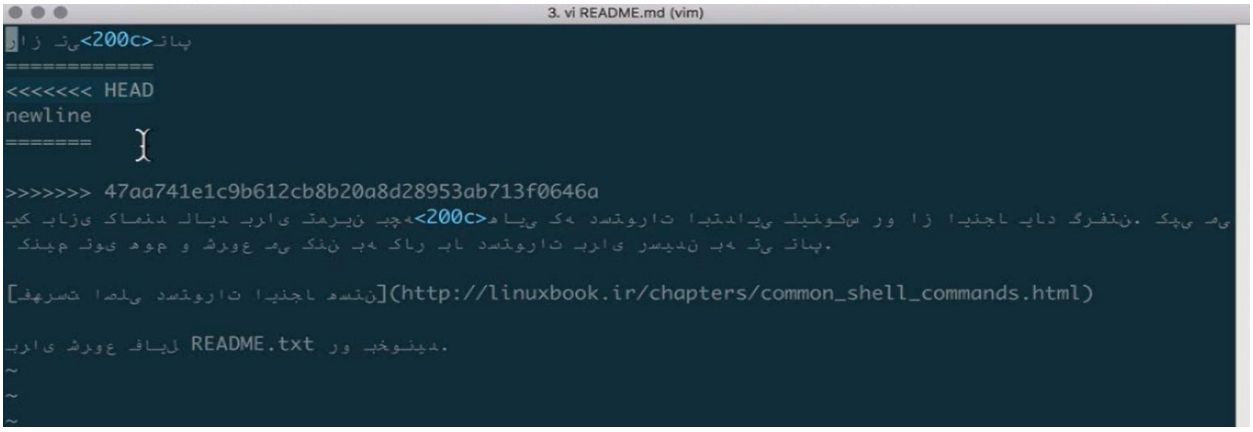
```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks
 (master)
$ git remote -v
origin  https://github.com/hosseindehghanipour1998/micro-homeworks.git (fetch)
origin  https://github.com/hosseindehghanipour1998/micro-homeworks.git (push)

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks
 (master)
$ git remote add cRemote https://github.com/hosseindehghanipour1998/principles-o
f-programming-2019.git

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks
 (master)
$ git remote -v
cRemote https://github.com/hosseindehghanipour1998/principles-of-programming-2019.git (fetch)
cRemote https://github.com/hosseindehghanipour1998/principles-of-programming-2019.git (push)
origin  https://github.com/hosseindehghanipour1998/micro-homeworks.git (fetch)
origin  https://github.com/hosseindehghanipour1998/micro-homeworks.git (push)

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$
```

In the illustration above we have created another remote to another repository of github named “ Principles-of-programming-2019 “ on which we can push and from which we can pull.

we can insert the copied URL into the terminal by pushing **“ ctrl + insert “**

some times we have a major problem. This problem will be caused when two different users are making changes on one same file. They both commit on one exactly the same branch. When one of them is going to push on the branch his own changes he will face an error saying that “ error: failed to push to URL ... “.In these situations you have to pull first in order to have his changes and then push

your own changes on the branch. The git would automatically merger your changes as you pull the project but something worse might happen which is both of the programmers have changed same lines of code which is the base of a major conflict in the file. git status will tell you that you both of you have made modified changes. You have to handle it manually by yourself. If you open the file you will see such a thing :



from the “<<<<<<HEAD” till “========” are the changes you have made and the line under the “=======” are the changes the other guy has made to this file.

and the line “ >>>>>>>>>>47.... “ is the name(hash) of the commit.

You can currently change and edit the file in order to solve the conflict and then make a new commit in order to push on the branch.

by **$ git log** you can see that you have resolved the conflict.

But if the both programmers change different parts of the code or file , the git will merge them together and doesn’t involve you to fix it manually.

by entering “git log” command you can see all the commits and their commiters. but what else you can see is that it shows you the hash of the commits. by command **$ git show <commit hash>** you can see the details of that commit .

```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks
 (master)
$ git log
commit 9d663d37c2adfd0b585c2a56597b1647434a0f92 (HEAD -> master, origin/master,
origin/HEAD)
Author: hossein dehghanipour <hossein.dehghanipour1998@gmail.com>
Date:   Mon Jul 8 13:35:39 2019 +0430

    Added AVR Sample Codes

commit 5bcd0d2e9363808aa71c061774161fbcb2176cf2
Author: hossein dehghanipour <hossein.dehghanipour1998@gmail.com>
Date:   Mon Jul 8 13:33:00 2019 +0430

    Added Project Description File

commit 52dd6f9ba595a193e89fc2edeeff92fe2f5ce7a7
Author: hossein dehghanipour <hossein.dehghanipour1998@gmail.com>
Date:   Mon Jul 8 13:30:13 2019 +0430

    Added Final Project
```

```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ git show 9d663d37c2adfd0b585c2a56597b1647434a0f92
commit 9d663d37c2adfd0b585c2a56597b1647434a0f92 (HEAD -> master, origin/master, origin/HEAD)
Author: hossein dehghanipour <hossein.dehghanipour1998@gmail.com>
Date:   Mon Jul 8 13:35:39 2019 +0430

    Added AVR Sample Codes

diff --git a/AVR Smaple Codes/ADC/AC/Debug/Exe/ac.a b/AVR Smaple Codes/ADC/AC/Debug/Exe/ac.a
new file mode 100644
index 0000000..dcff31f
--- /dev/null
+++ b/AVR Smaple Codes/ADC/AC/Debug/Exe/ac.a
@@ -0,0 +1,213 @@
+;/*****************************************************^M
+;This program was created by the^M
+;CodeWizardAVR V3.12 Advanced^M
+;Automatic Program Generator^M
+;<A9> Copyright 1998-2014 Pavel Haiduc, HP InfoTech s.r.l.^M
+;http://www.hpinfotech.com^M
+;^M
+;Project :^M
+;Version :^M
+;Date    : 6/2/2019^M
+;Author  :^M
+;Company :^M
+;Comments:^M
+;^M
+;^M
+;Chip type               : ATmega32^M
+;Program type            : Application^M
+;AVR Core Clock frequency: 8.000000 MHz^M
+;Memory model            : Small^M
+;External RAM size       : 0^M
+;Data Stack size         : 512^M
+;*****************************************************/^M
+;^M
+;#include <mega32.h>^M
```

Changes

After our project has reached a level that it's suitable for usage for the client, we public a version of our application. Or when we update or improve the app we upgrade it's version. As an example we call it "V 1.1.2". We call this operation tagging which means that we tag a label to our commit in order to remind ourselves that this commit is the first version of our program we have released and in the possible future when we want to look for our first released version we search for the tag instead of reading all changes and the codes we have written.

```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ git tag

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ git tag -a v2.0 -m "Our first version"

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ git tag
v2.0

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$
```

notice that **–a** stands for **annotation** and **–m** stands for **Message.**

assume that we are at the stage of v2.0 but we have forgotten to label our v1.8 . it's not late at all. all we have to do is to find the commit that makes out v1.8 and copy first 10-11 alphabets of it's hash. and then stick the wanted label to it.

```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ clear

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ git tag

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ git tag -a v2.0 -m "Our first version"

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ git tag
v2.0

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ git log
commit 9d663d37c2adfd0b585c2a56597b1647434a0f92 (HEAD -> master, tag: v2.0, origin/master,
Author: hossein dehghanipour <hossein.dehghanipour1998@gmail.com>
Date:   Mon Jul 8 13:35:39 2019 +0430

    Added AVR Sample Codes

commit 5bcd0d2e9363808aa71c061774161fbcb2176cf2
Author: hossein dehghanipour <hossein.dehghanipour1998@gmail.com>
Date:   Mon Jul 8 13:33:00 2019 +0430

    Added Project Description File

commit 52dd6f9ba595a193e89fc2edeeff92fe2f5ce7a7
Author: hossein dehghanipour <hossein.dehghanipour1998@gmail.com>
Date:   Mon Jul 8 13:30:13 2019 +0430

    Added Final Project
```

```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ git tag -a v1.8 52dd6f9ba595a -m "Version 1.8 released"

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ git tag
v1.8
v2.0

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$
```

$ git tag –L "v*" -> shows you all of the tags that start with "v"

```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ clear

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ git tag -a v1.8 52dd6f9ba595a -m "Version 1.8 released"

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ git tag
v1.8
v2.0

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ git tag -a B2.0 -m "Beta version added "

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ git tag
B2.0
v1.8
v2.0

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ git tag -l "v*
> '^C

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ git tag -l "v*"
v1.8
v2.0
```

by **$ git show <version number>** we can see through the details of a specific version.

```
Hossein@DESKTOP-CHGJ5CO MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ git show v1.8
tag v1.8
Tagger: Hossein Dehghanipour <Hossein.dehghanipour1998@gmail.com>
Date:   Tue Sep 10 17:07:11 2019 +0430

Version 1.8 released

commit 52dd6f9ba595a193e89fc2edeeff92fe2f5ce7a7 (tag: v1.8)
Author: hossein dehghanipour <hossein.dehghanipour1998@gmail.com>
Date:   Mon Jul 8 13:30:13 2019 +0430

    Added Final Project

diff --git a/Micro Final Project/Design_2.DSN b/Micro Final Project/Design_2.DSN
new file mode 100644
index 0000000..4bb7d4e
Binary files /dev/null and b/Micro Final Project/Design_2.DSN differ
diff --git a/Micro Final Project/Design_2.PWI b/Micro Final Project/Design_2.PWI
new file mode 100644
index 0000000..ea51346
Binary files /dev/null and b/Micro Final Project/Design_2.PWI differ
diff --git a/Micro Final Project/Last Loaded Design_2.DBK b/Micro Final Project/Last Loaded
new file mode 100644
index 0000000..16309e4
Binary files /dev/null and b/Micro Final Project/Last Loaded Design_2.DBK differ
diff --git a/Micro Final Project/Master/Debug/Exe/alcd.al b/Micro Final Project/Master/Debu
new file mode 100644
index 0000000..6bfec64
--- /dev/null
+++ b/Micro Final Project/Master/Debug/Exe/alcd.al
@@ -0,0 +1,325 @@
+;PCODE: $00000000 VOL: 0
+       #ifndef __SLEEP_DEFINED__
+;PCODE: $00000001 VOL: 0
+       #define __SLEEP_DEFINED__
+;PCODE: $00000002 VOL: 0
+       .EQU __se_bit=0x80
+;PCODE: $00000003 VOL: 0
+       .EQU __sm_mask=0x70
+;PCODE: $00000004 VOL: 0
+       .EQU __sm_powerdown=0x20
+;PCODE: $00000005 VOL: 0
+       .EQU __sm_powersave=0x30
+;PCODE: $00000006 VOL: 0
+       .EQU __sm_standby=0x60
+;PCODE: $00000007 VOL: 0
+       .EQU __sm_ext_standby=0x70
+;PCODE: $00000008 VOL: 0
+       .EQU __sm_adc_noise_red=0x10
+;PCODE: $00000009 VOL: 0
+       .SET power_ctrl_reg=mcucr
+;PCODE: $0000000A VOL: 0
+       #endif
+;PCODE: $0000000B VOL: 0
+;PCODE: $0000000C VOL: 0
+
```

now we have made tags but they are locally made. We have to also push the tags on our origin branch.

we say : `$ git push origin --tags` which pushes the tags on the main project on the server.

I can move to version v1.8 from where I am standing right now.

$ git checkout <version name> ex : $ git checkout v1.8

we are now standing on branch v1.8 but we can't make changes to this version and then commit it here. what we can do is make another branch on this version and make the changes on that branch then merge it with master.

The hash we see as commit is something beyond a simple string. That's a security procedure in order to avoid someone else making changes to the project in your name. So if it says that the commit is made by Hossein, That's definitely true. If someone wants to change the project and make it looks like you have done it, then he has to steal all of your keys and uses them against you which is a hard work and somehow impossible to do. The encryption that is used nowadays in computers is **PGP** which stands for **Pretty Good Privacy.**

GPG is the encryption for **Genau** Systems.(Google it)

In this system, There are a pair of keys for each user (we assume A & B). We keep A key to ourselves in order to encrypt what ever we write (we use it so make a digital Signature) and we publish key B for everyone to be able to see what changes we have made. So if someone can read some data with Key B it means that we had the Key A to encrypt it.

Ex : Assume there are 3 programmers. programmer A, B, C. Each one of these guys has two keys (A1, A2, B1, B2, C1, C2) each user encrypts his own data ba key #1 and publishes key #2 for others to use. If programmer B uses key A2 to read some data it means that is written by Programmer A only because only his key (A2) could decrypt the data.

Git works like this. Every user uses a key to encrypt and publishes the other key for others to decrypt the written data.

`$ gpg --list-keys` -> this command shows you all of your keys.

let's make some keys :

`$ gpg --gen-key`

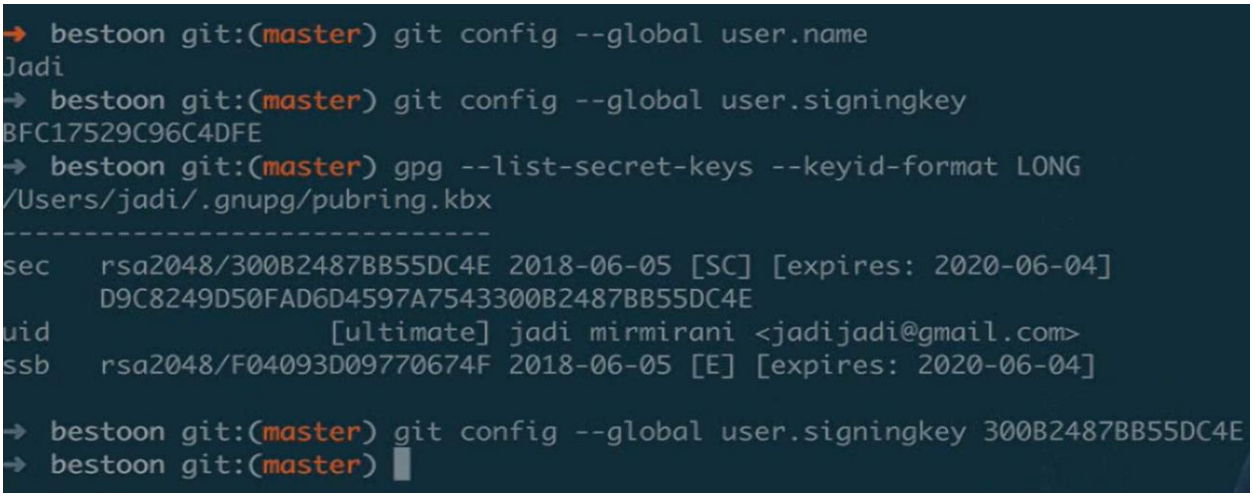
```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks (master)
$ gpg --gen-key
gpg (GnuPG) 1.4.22; Copyright (C) 2015 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: keyring `/c/Users/hosse/.gnupg/secring.gpg' created
Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 3
DSA keys may be between 1024 and 3072 bits long.
What keysize do you want? (2048) 1024
Requested keysize is 1024 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 1
Key expires at Wed, Sep 11, 2019  6:54:22 PM IDT
Is this correct? (y/N) y

You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: hossein
Email address: hossein.dehghanipour1998@gmail.com
Comment: Testing On making a Key
You selected this USER-ID:
    "hossein (Testing On making a Key) <hossein.dehghanipour1998@gmail.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
You need a Passphrase to protect your secret key.

We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
++++++++++++++++++++++++.++++++++++++++++++++++++++++++..+++++.++++++++++..++++++++++++++++++++.++++++++
gpg: key D2EA8FBC marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2019-09-11
pub   1024D/D2EA8FBC 2019-09-10 [expires: 2019-09-11]
      Key fingerprint = 4D1B 84C1 E6E0 DACF 67F8  E2D6 F510 BCB7 D2EA 8FBC
uid                  hossein (Testing On making a Key) <hossein.dehghanipour1998@gmail.com>

Note that this key cannot be used for encryption.  You may want to use
the command "--edit-key" to generate a subkey for this purpose.
```

the blue parts are the parts that get information from the user in order to create the key and the red part is the generated public key.

`$ git config --global user.signinkey` this command tells us that with what key we have signed in.

`$git --list-secret-keys --keyid-format LONG`

By the commands below we tell git to change our sign in key to what we give it (actually what we give must be a valid gpg key that we have already made) :



as we have already said we can label our commits but if we write **–s** instead of **–a** it means Sign (not annotate) and after this command git requires a password from us in order to make sure that we are the real us :) and by typing :

$ git show v2.1 then it shows who has released it and shows his digital signature key.

because some times some problems might be caused, either all members of the team should sign or none of the members should sign. Be aware that if you want to sign then you should sign all of the commits you make.

`$ git help <what?>` -> $ git help blame

`$ git blame <fileName> -L <Line Number >`

checks the specific line in that file in order to find who has done any changes on it.

`$git blame <File Name >`

checks the whole file and changes .

```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks
 (master)
$ git blame test\ Text.txt
dc03cf90 (Hossein 2019-09-10 19:51:57 +0430 1) Hello
dc03cf90 (Hossein 2019-09-10 19:51:57 +0430 2) My name
dc03cf90 (Hossein 2019-09-10 19:51:57 +0430 3) Is Hossein
```

```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/m
 (master)
$ git blame test\ Text.txt -L 3
dc03cf90 (Hossein 2019-09-10 19:51:57 +0430 3) Is Hossein
```

```
Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks
 (master)
$ git blame test\ Text.txt -L8,10
fatal: file test Text.txt has only 3 lines

Hossein@DESKTOP-CHGJ5C0 MINGW64 /g/GitRepositories/Testing Clone/micro-homeworks
 (master)
$ git blame test\ Text.txt -L2,3   Show me changes from line 2 to line 3
dc03cf90 (Hossein 2019-09-10 19:51:57 +0430 2) My name
dc03cf90 (Hossein 2019-09-10 19:51:57 +0430 3) Is Hossein
```

this allows us to blame the programmer who wrote the line of code which has a bug.

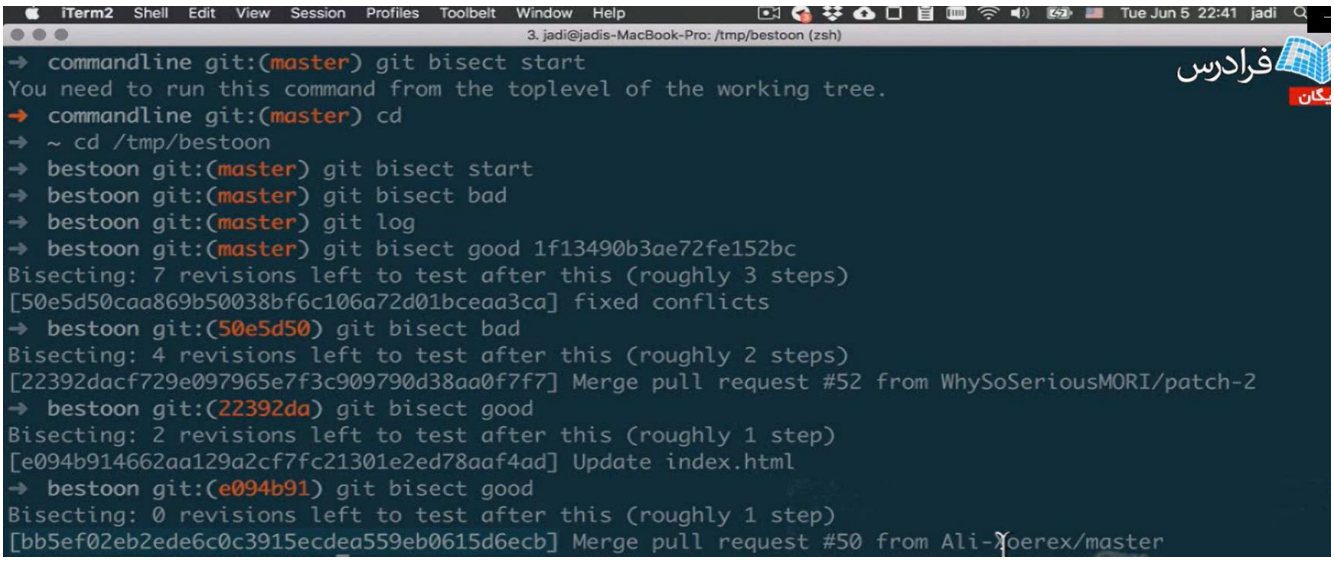
git has a tool which helps us find a bug due to our commits. you tell git to start your work in finding the bug :

$ git bisect start

ok. now we have started and we must be at the top directory level to call this command. we can say at each step whether the current commit is good or bad.

what git does is that is runs a binary search algorithm according to your saying about a commit to be good or bad and at each step git gives you the hash of a specific commit to check. If you **check** the code in that commit and see that the

code is fine you say `$ git insect good` otherwise you say `$git insect bad.`



Forking :

we can fork a project from some other guy. We make some changes. Then we make a “pull request” in order to have the changes on that other guy’s project. so that guy should approve whether he wants you to make changes to his project or not.

- git init

شروع کار با git

- git add fileName

افزودن فایل به git

- git add -A

اضافه کردن همه فایل‌ها به git

- git commit -m "description"

کامیت کردن تغییرات با توضیحات

- git status

نمایش وضعیت

- git reset fileName

خارج کردن فایل از حالت stage

- git diff HEAD

نمایش وضعیت فعلی نسبت به وضعیت آخرین کامیت

- git diff --staged

نمایش وضعیت فعلی نسبت به وضعیت stage

- git checkout --fileName

فایل را از آخرین کامیت استخراج کرده و جایگزین فایل فعلی می کند

- git branch

نمایش شاخه‌های موجود

- git branch branchName

ساخت شاخه جدید با نام تعیین شده

- git checkout branchName

سوئیچ کردن از شاخه فعلی به شاخه تعیین شده

- git merge branchName

شاخه مذکور را با شاخه فعلی ادغام می کند

- git rm fileName

حذف فایل از git و از فایل سیستم

- git branch -d branchName

حذف شاخه

- git push origin master

شاخه master را به origin ارسال می کند

- git pull origin master

شاخه master را از origin دریافت می کند

- git remote

نمایش remote

- git remote add origin url

افزودن remote با آدرس تعیین شده و نام origin

- git show commitID

نمایش جزئیات commit با شناسه تعیین شده

- git tag

نمایش تگ ها

- git tag -a tagName -m "description

افزودن تگ با نام و توضیحات تعیین شده

- git show tagName

نمایش جزئیات تگ

- git blame fileName -L lineNumber

مشاهده اینکه خط مذکور از فایل تعیین شده را چه کسی نوشته است

- git bisect

برای debug به کار می رود

- git config

برای تنظیمات ابزار به کار می رود. مانند مشخصات نویسده و همچنین تنظیمات مربوط به پراکسی برای کلاینت گیت

Thanks to Jadi for this amazing tutorial.

Jadi’s GitHub .