2 – Git Basics

Getting a repository

➔ One can get Git repository in two ways

1. One can convert local repository into a git repository.
2. One can clone a git repository from elsewhere.

Cloning an existing repository

➔ One can clone an existing repository using

```
$ git clone <url>
```

Initializing a repository in a existing directory

➔ If one has project directory not under version control then first he needs to go that directory on git bash as shown below

```
Jayant@JayantDELL MINGW64 /f/Java material/Video tutorials/Machine Learning
Courses/CS109/Git/Labs/Git Test/my_project
```

and type

```
$ git init
```

This will make my_project a git repository. This will also create new subdirectory name '.git' which contains all of ones necessary repository files – a git repository skeleton.

Staging and Committing

➔ Lets create two files in the repository README.txt and LICENSE.txt in my_project. Then we have

```
Jayant@JayantDELL MINGW64 /f/Java material/Video tutorials/Machine Learning
Courses/CS109/Git/Labs/Git Test/my_project (master)
$ ls
LICENSE.txt  README.txt
```

Getting status of your file

➔ Remember each file in the working directory will be in one of the two states : *tracked* or *untracked*.

➔ Tracked files are those who were there in the last snapshot. That is they are the files which Git knows about. They could be unmodified, modified or staged.

➔ Untracked files are anything else. Which are not in the last snapshot or in staging area.

➔ When one clone a repository all files there are tracked and unmodified because nothing has been edited or deleted yet. As one edit files Git sees them as modified.

➔ To know the status of a files/repository we have command 'git status'. Lets see what is the status of our repository.

```
$ git status
On branch master

No commits yet

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

```
        LICENSE.txt
        README.txt

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

➔ Above we can see both LICENSE.txt and README.txt are marked 'untracked files:'because they are newly added but not committed yet.

Tracking new files

➔ Lets add them to the staging area with the command

$ git add *.txt

(Note : One can add single file too as git add LICENSE.txt, but then for each file needs one command).

➔ Lets check the status once again after adding files to the staging area.

```
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   LICENSE.txt
        new file:   README.txt
```

➔ Above we can see both files are marked 'Changes to be committed:' which means they have been added to the staging area.

➔ Lets commit now as shown below

```
$ git commit -m "LICENSE.txt and README.txt have been added"
[master (root-commit) 411a52f] LICENSE.txt and README.txt have been added
 2 files changed, 2 insertions(+)
 create mode 100644 LICENSE.txt
 create mode 100644 README.txt
```

➔ Lets check git status again

```
$ git status
On branch master
nothing to commit, working tree clean
```

➔ Now lets add and commit one more file contributing.txt in the same way above. So now repository will have

```
$ ls
contributing.txt  LICENSE.txt  README.txt
```

Staging modified files

➔ Lets modify file contributing.txt by adding some content.When we changed previously tracked file then status will be

```
$ 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:   contributing.txt

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

➔ Above we can see it marks it as 'Changes not staged for commit:' because git can track this file and it has been modified.

➔ Lets stage contributing.txt and run status again

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   contributing.txt
```

➔ Here we can see it has been staged. Now if before committing user changes contributing.txt again then status will be

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   contributing.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:   contributing.txt
```

➔ Now here contributing.txt is listed as both staged and unstaged. This clearly says 'git add' command dont stage file but stage **content**. So it has staged the previous change but not the new one. Hence it is listed both staged and unstaged. If you commit now the latest changes wont be committed and for that file needs to be staged again. That is if one changes file after 'git add' then he needs to 'git add' again to commit all the changes.

Short Status

➔ Current status for repository 'my_project' is

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.txt
        new file:   README1.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:   contributing.txt

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

        README2.txt
```

➔ 'git status' output is quite comprehensive. To make it short git have command 'git status -s' or 'git status –short' which both yield the same status.

➔ Lets create two more file README1.txt and README2.txt in the repository. Lets stage README1.txt file. Also edit README.txt and stage it. After all this lets check the status now

```
$ git status -s
M  README.txt
A  README1.txt
 M contributing.txt
?? README2.txt
```

➔ Above we can see

a. Untracked file README1.txt is newly created and staged hence marked as 'A'.

b. Untracked file README2.txt is created but not staged hence marked as '??'.

c. Tracked file contributing.txt is modified but not staged hence marked 'M'(red).

d. Tracked file README.txt modifed and staged hence marked 'M'(green).

➔ Above we can see there are two colums in the output(the way A, M etc are oriented). Left side indicates

status of the staging area(green) and right side indicates status of the working directory(red).

Ignoring Files

➔ Sometimes there will be files that one dont want Git to automatically add or even show you as being untracked. In other words one wanted to avoid these files because he dont want to add them to index i.e. to commit to git.

➔ These are generally automatically generated files like log files or files generated by ones build system.

➔ For this, one can create file listing patterns to match them in .gitignore file. Below is an example of .gitignore file

```
$ cat .gitignore
*.[oa]
*~
```

The first line tells git to ignore any files ending in '.o' or '.a' extension, i.e. object and archive files. The second line tells to ignore all files whose name ends with ~, which is used by many text editors such as emacs to mark temporary files.

➔ One may also include log, tmp and pid directory, automatically generated documentation and so on.

➔ Setting up a .gitignore file for your new repository before you get going is a good idea so you dont accidentally commit files which you dont want in your git repository.

➔ Rules for patterns one can use in .gitignore file

- Blank lines or lines starting with # are ignored.
- Standard glob patterns work, and will be applied recursively throughout the entire working tree.
- You can start patterns with a forward slash (/) to avoid recursivity.
- You can end patterns with a forward slash (/) to specify a directory.
- You can negate a pattern by starting it with an exclamation point (!).

➔ Glob patterns are like simplified regular expressions that shells use. Example

- Asterik (*) matches zero or more characters.
- [abc] matches any character inside bracket(a, b or c in this case).
- a question marke (?) matches single characters.
- Bracket enclosing pattern separated by hyphen ([0-9]) matches any character between them(in this case 0 to 9).
- One can also use two asteriks to match nested directories like a/**/z would match a/z, a/b/z, a/b/c/d/z.

➔ Example

```
# ignore all .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the TODO file in the current directory, not subdir/TODO
/TODO

# ignore all files in the build/ directory
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```

➔ Git maintains list of .gitignore file examples to use for different situations at https://github.com/github/gitignore

➔ One can have different .gitignore files for main directory and sub directories. In such case of nested .gitignore files rules applies to the directories where they are located.

Viewing your staged and unstaged changes

➔ Command 'git status' gives one info about which files are created or changed or staged, but it don't give ones idea about what changes exactly made in the files. For this we have command 'git diff'.

➔ Current status of 'my_project' repository is

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.txt
        new file:   README1.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:   contributing.txt

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

        README2.txt
```

➔ Now lets check what changes have been made to the files using command 'git diff'.

```
$ git diff
diff --git a/contributing.txt b/contributing.txt
index 9fda799..abdf340 100644
--- a/contributing.txt
+++ b/contributing.txt
@@ -1 +1,3 @@
 contributing change1
+contributing change2
+contributing change3
```

➔ The above command compares ones working directory and staging area. The result will tell you about changes which have been made but not yet staged.

➔ If one want to know what difference between what has been staged and the last commit then he can use 'git diff --staged' or(git diff --cached) as shown below.

```
$ git diff --staged
diff --git a/README.txt b/README.txt
index f717364..5378c00 100644
--- a/README.txt
+++ b/README.txt
@@ -1 +1 @@
-Chapter 2 Git Basic from book Pro Git 2nd edition.
\ No newline at end of file
+Chapter 1-3 from book Pro Git 2nd edition.
diff --git a/README1.txt b/README1.txt
new file mode 100644
index 0000000..9c0f9f8
--- /dev/null
+++ b/README1.txt
@@ -0,0 +1 @@
+readme file 1
\ No newline at end of file
```

➔ That is 'git diff' shows changes which are unstaged. And 'git diff --staged' shows changes which are staged.

Skipping the staging area

➔ One can commit all the tracked files at one go by adding '-a' during git commit. In this way one dont need to use 'add' to add files one by one to staging area first. Lets commit our new changes using it.

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.txt
        new file:   README1.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:   README1.txt
        modified:   contributing.txt

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

        README2.txt


Jayant@JayantDELL MINGW64 /f/Java material/Video tutorials/Machine Learning
Courses/CS109/Git/Labs/git Test/my_project (master)
$ git commit -a -m "added changes to README, README1 and contributing.txt file"
[master 9943dc9] added changes to README, README1 and contributing.txt file
 3 files changed, 4 insertions(+), 1 deletion(-)
 create mode 100644 README1.txt

Jayant@JayantDELL MINGW64 /f/Java material/Video tutorials/Machine Learning
Courses/CS109/Git/Labs/git Test/my_project (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README2.txt

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

Removing files

➔ Removing a file from git means removing it from ones tracked files.

➔ Just removing it from ones working directory will shows it under “Changes not staged for commit:”. Lets remove README1.txt just from working directory and check status.

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    README1.txt

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

        README2.txt

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

➔ Hence we have 'git rm' command to remove file completely. Lets check how it works

```
$ git rm README1.txt
rm 'README1.txt'

Jayant@JayantDELL MINGW64 /f/Java material/Video tutorials/Machine Learning
Courses/CS109/Git/Labs/git Test/my_project (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    README1.txt

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

```
        README2.txt
```

➔ The next time one commits the file will be gone forever and will no longer will tracked.

➔ Lets commit README2.txt file now

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    README1.txt

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

        README2.txt


Jayant@JayantDELL MINGW64 /f/Java material/Video tutorials/Machine Learning
Courses/CS109/Git/Labs/git Test/my_project (master)
$ git add README2.txt

Jayant@JayantDELL MINGW64 /f/Java material/Video tutorials/Machine Learning
Courses/CS109/Git/Labs/git Test/my_project (master)
$ git commit -m "committing README2.txt file"
[master 154e8ed] committing README2.txt file
 2 files changed, 1 insertion(+), 1 deletion(-)
 delete mode 100644 README1.txt
 create mode 100644 README2.txt
```

➔ Above we can see README1.txt file has been deleted and README2.txt file added to the new snapshot.

➔ Sometime we want to delete file from git but keep it in working directory. For this we have command “git rm –cached <file name>”. Lets remove file README2.txt with this.

```
$ git rm --cached README2.txt
rm 'README2.txt'


Jayant@JayantDELL MINGW64 /f/Java material/Video tutorials/Machine Learning
Courses/CS109/Git/Labs/git Test/my_project (master)
$ git commit -m "just a plain commit"
[master 9bef986] just a plain commit
 1 file changed, 1 deletion(-)
 delete mode 100644 README2.txt

Jayant@JayantDELL MINGW64 /f/Java material/Video tutorials/Machine Learning
Courses/CS109/Git/Labs/git Test/my_project (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README2.txt

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

➔ We can see README2.txt file has been deleted during last commit and is shown as untracked currently.

➔ With 'git rm' command one can pass files, directories and file-glob patterns like

```
git rm log/\*.log
```

The above command removes all the file with extension .log present in the log/ directory.

git rm *~

Above command removes all files whose name end with a '~'.

Moving files

➔ One can rename a file using 'git mv' command.

```
$ git mv contributing.txt contributing_new.txt

Jayant@JayantDELL MINGW64 /f/Java material/Video tutorials/Machine Learning
Courses/CS109/Git/Labs/git Test/my_project (master)
$ git status
On branch master
```

```
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        renamed:    contributing.txt -> contributing_new.txt

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

        README2.txt
```

Viewing the commit history

➔ If one have committed several times or if one has copied a repository with commit history then he can check it through command 'git log'. Lets check it for our repository my_project.

```
$ git log
commit 9bef986304590eb99f185d1f3341013a81c66bff (HEAD -> master)
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sat Dec 16 12:51:03 2017 +0530

    just a plain commit

commit 154e8edce6ca7a115f890ca2bf0fba50d709608e
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sat Dec 16 12:43:16 2017 +0530

    committing README2.txt file

commit 9943dc9e6bfc342af4c1fdc1e7320bf0a590d018
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sat Dec 16 12:17:04 2017 +0530

    added changes to README, README1 and contributing.txt file

commit e3144ebccee6f27de126d94ff296a8bf16c178e9
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sun Dec 10 12:38:12 2017 +0530

    changes1 and changes2 added

commit 580ec214db580b0ef98c1701758da77cd50d28f0
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sun Dec 10 12:00:43 2017 +0530

    adding contributing.txt

commit 411a52f06af4998907eabccb660671a4354fd81b
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sun Dec 10 11:48:52 2017 +0530

    LICENSE.txt and README.txt have been added
```

➔ 'git log' lists the commits in reverse chronological order.

➔ This command lists commit with SHA-1 checksum, the author's name, email, date written and commit message.

➔ Variety of options are available with 'git log' like -p or --patch which shows the difference introduced in each output.

➔ Also one can limit the number of entries using -2 or -3 etc

```
$ git log -p -3
commit 9bef986304590eb99f185d1f3341013a81c66bff (HEAD -> master)
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sat Dec 16 12:51:03 2017 +0530

    just a plain commit

diff --git a/README2.txt b/README2.txt
deleted file mode 100644
index 1a32bb3..0000000
--- a/README2.txt
+++ /dev/null
@@ -1 +0,0 @@
-readme file 2
\ No newline at end of file

commit 154e8edce6ca7a115f890ca2bf0fba50d709608e
```

```
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sat Dec 16 12:43:16 2017 +0530

    committing README2.txt file

diff --git a/README1.txt b/README1.txt
deleted file mode 100644
index 65e2881..0000000
--- a/README1.txt
+++ /dev/null
@@ -1 +0,0 @@
-readme file 1 ##
\ No newline at end of file
diff --git a/README2.txt b/README2.txt
new file mode 100644
index 0000000..1a32bb3
--- /dev/null
+++ b/README2.txt
@@ -0,0 +1 @@
+readme file 2
\ No newline at end of file

commit 9943dc9e6bfc342af4c1fdc1e7320bf0a590d018
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sat Dec 16 12:17:04 2017 +0530

    added changes to README, README1 and contributing.txt file

diff --git a/README.txt b/README.txt
index f717364..5378c00 100644
--- a/README.txt
+++ b/README.txt
+++ b/README.txt
@@ -1 +1 @@
-Chapter 2 Git Basic from book Pro Git 2nd edition.
\ No newline at end of file
+Chapter 1-3 from book Pro Git 2nd edition.
diff --git a/README1.txt b/README1.txt
new file mode 100644
index 0000000..65e2881
--- /dev/null
+++ b/README1.txt
@@ -0,0 +1 @@
+readme file 1 ##
\ No newline at end of file
diff --git a/contributing.txt b/contributing.txt
index 9fda799..abdf340 100644
--- a/contributing.txt
+++ b/contributing.txt
@@ -1 +1,3 @@
 contributing change1
+contributing change2
+contributing change3
```

➔ To see some abbreviated state for commit

```
$ git log -2 --stat
commit 9bef986304590eb99f185d1f3341013a81c66bff (HEAD -> master)
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sat Dec 16 12:51:03 2017 +0530

    just a plain commit

 README2.txt | 1 -
 1 file changed, 1 deletion(-)

commit 154e8edce6ca7a115f890ca2bf0fba50d709608e
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sat Dec 16 12:43:16 2017 +0530

    committing README2.txt file

 README1.txt | 1 -
 README2.txt | 1 +
 2 files changed, 1 insertion(+), 1 deletion(-)
```

➔ Above we can see --stat command shows list of modified files, how many files were changed and how many lines in those files are added or removed. It also put summary of info at the end.

➔ The command "--pretty" shows log in a format other than default. A few pre-built options are available to use. Example is shown below

```
$ git log --pretty=oneline
9bef986304590eb99f185d1f3341013a81c66bff (HEAD -> master) just a plain commit
```

```
154e8edce6ca7a115f890ca2bf0fba50d709608e committing README2.txt file
9943dc9e6bfc342af4c1fdc1e7320bf0a590d018 added changes to README, README1 and contributing.txt file
e3144ebccee6f27de126d94ff296a8bf16c178e9 changes1 and changes2 added
580ec214db580b0ef98c1701758da77cd50d28f0 adding contributing.txt
411a52f06af4998907eabccb660671a4354fd81b LICENSE.txt and README.txt have been added
```

➔ Like oneline there are short, full, fuller options to get log in different formats.

➔ The command 'format' let one decide format of his log. It should be used as

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

➔ Similarly there are many options for git log(see more in text book). Most common have been listed below

Common options to `git log`

Option	Description
`-p`	Show the patch introduced with each commit.
`--stat`	Show statistics for files modified in each commit.
`--shortstat`	Display only the changed/insertions/deletions line from the --stat command.
`--name-only`	Show the list of files modified after the commit information.
`--name-status`	Show the list of files affected with added/modified/deleted information as well.
`--abbrev-commit`	Show only the first few characters of the SHA-1 checksum instead of all 40.
`--relative-date`	Display the date in a relative format (for example, “2 weeks ago”) instead of using the full date format.
`--graph`	Display an ASCII graph of the branch and merge history beside the log output.
`--pretty`	Show commits in an alternate format. Options include oneline, short, full, fuller, and format (where you specify your own format).
`--oneline`	Shorthand for `--pretty=oneline --abbrev-commit` used together.

Undoing things

➔ During early commits it happens that one has forgot to commit all files or commit message may got messed up. For this one should first stage those changes and commit again using --ammend option.

$ git commit --amend

➔ With this one will end up with single commit, the second commit replaces the result of the first.

➔ Lets add README2.txt to the repository again and check log.

```
$ git log
commit 45c4a370f4e79e58b3b377ddf33d9dc64c10ec20 (HEAD -> master)
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sun Dec 17 12:26:52 2017 +0530

    Adding README2.txt again

commit e3e4068702fc39b4a1ff8c8d3156f3e52dd884f8
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sun Dec 17 12:26:01 2017 +0530

    adding README2.txt again

commit 9bef986304590eb99f185d1f3341013a81c66bff
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sat Dec 16 12:51:03 2017 +0530

    just a plain commit

commit 154e8edce6ca7a115f890ca2bf0fba50d709608e
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sat Dec 16 12:43:16 2017 +0530
```

```
    committing README2.txt file

commit 9943dc9e6bfc342af4c1fdc1e7320bf0a590d018
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sat Dec 16 12:17:04 2017 +0530

    added changes to README, README1 and contributing.txt file

commit e3144ebccee6f27de126d94ff296a8bf16c178e9
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sun Dec 10 12:38:12 2017 +0530

    changes1 and changes2 added

commit 580ec214db580b0ef98c1701758da77cd50d28f0
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sun Dec 10 12:00:43 2017 +0530

    adding contributing.txt

commit 411a52f06af4998907eabccb660671a4354fd81b
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sun Dec 10 11:48:52 2017 +0530

    LICENSE.txt and README.txt have been added
```

(Note : Above we can see README2.txt committed twice).

➔ Now lets commit again with amend and change the message with following command.

```
$ git commit --amend -m "second amend"
[master 7f7499b] second amend
 Date: Sun Dec 17 12:26:52 2017 +0530
 1 file changed, 3 insertions(+)
 create mode 100644 README2.txt

Jayant@JayantDELL MINGW64 /f/Java material/Video tutorials/Machine Learning
Courses/CS109/Git/Labs/git Test/my_project (master)
$ git log
commit 7f7499bf7154d060c42d38c04adcd449d5ece5e1 (HEAD -> master)
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sun Dec 17 12:26:52 2017 +0530

    second amend

commit e3e4068702fc39b4a1ff8c8d3156f3e52dd884f8
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sun Dec 17 12:26:01 2017 +0530

    adding README2.txt again

commit 9bef986304590eb99f185d1f3341013a81c66bff
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sat Dec 16 12:51:03 2017 +0530

    just a plain commit

commit 154e8edce6ca7a115f890ca2bf0fba50d709608e
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sat Dec 16 12:43:16 2017 +0530

    committing README2.txt file

commit 9943dc9e6bfc342af4c1fdc1e7320bf0a590d018
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sat Dec 16 12:17:04 2017 +0530

    added changes to README, README1 and contributing.txt file

commit e3144ebccee6f27de126d94ff296a8bf16c178e9
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sun Dec 10 12:38:12 2017 +0530

    changes1 and changes2 added

commit 580ec214db580b0ef98c1701758da77cd50d28f0
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sun Dec 10 12:00:43 2017 +0530

    adding contributing.txt

commit 411a52f06af4998907eabccb660671a4354fd81b
Author: Jayant <jayantjoshi0209@gmail.com>
Date:   Sun Dec 10 11:48:52 2017 +0530

    LICENSE.txt and README.txt have been added
```

Unstaging a staged file

➔ Lets we have two files File1.txt and File2.txt. Both committed to my_project directory.

➔ Right now we want to only commit File1.txt. But we have used command

```
$ git add *
```

So now the status will be

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   File1.txt
        new file:   File2.txt
```

➔ Now to unstage File2.txt we should use command

```
$ git reset HEAD File2.txt
```

➔ Now lets check status again

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   File1.txt

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

        File2.txt
```

➔ Command '$ git status –hard' not only unstage but removes all staged file from local drive too. Hence one should use it carefully.

➔ There are many options to use with reset.

Unmodifying a modified file

➔ Lets one have made change to README2.txt file. To revert them he should use command

git checkout -- README.txt

Lets see the example below. Lets modify README2.txt and see the status

```
$ 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:   README2.txt
```

➔ Lets revert them using command mentioned above then we have

```
Jayant@JayantDELL MINGW64 /f/Java material/Video tutorials/Machine Learning
Courses/CS109/Git/Labs/git Test/my_project (master)
$ git checkout -- README2.txt

Jayant@JayantDELL MINGW64 /f/Java material/Video tutorials/Machine Learning
Courses/CS109/Git/Labs/git Test/my_project (master)
$ git status
On branch master
nothing to commit, working tree clean
```

➔ Here we can see nothing there to commit and if we open the file README2.txt we will find that changes are completely gone. Hence 'checkout' is a dangerous command, any changes made to file are completely gone.

Working With Remotes

➔ When some one clones a repository then by default it is called as origin in git.

➔ When one adds or modified a file in cloned repository then after committing he can PUSH this work to the remote(origin) using command

```
$ git push origin master
```

➔ If something has been added/modified in the remote directory then he first needs to pull/merge the latest update and then push his modified work to the remote. For pulling we have command

```
$ git pull <url>
```

➔ One can check the all remote directories details using command

```
$ git remote -v
```

➔ One can add a remote directory using command(in case its not cloned)

```
$ git remote add <rem dire name> “<url>”
```

➔ By using above command one can add many remote directories where he wants to push same work.

➔ One can push same work to the multiple repositories provided all repositories have same commit history. For example one can create Repository1 and file1.txt locally. Then he creates two repositories Repository2 and Repository3 on github(internet). He can add both of them as remote as shown below

```
$ git remote add repo2 “https://github.com/JayJosh2k09/Repository2.git”
$ git remote add repo3 “https://github.com/JayJosh2k09/Repository3.git”
```

➔ and then one can push his work to both of them using command

```
$ git push repo2 master
$ git push repo3 master
```