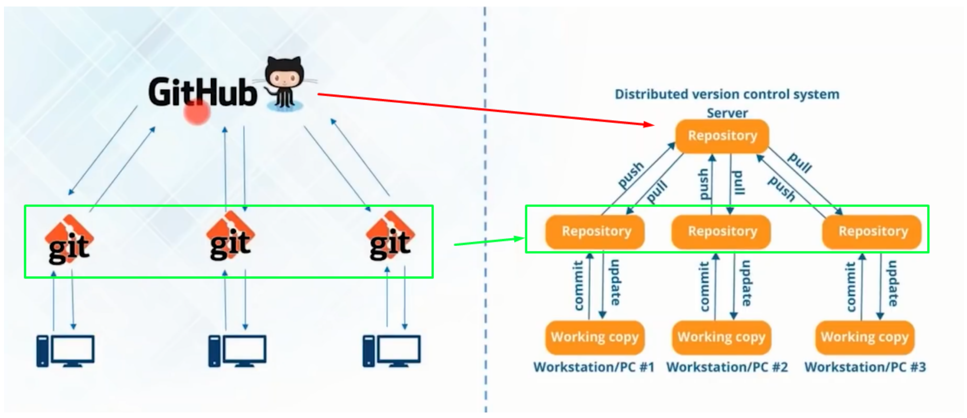
Git-Github Collaboration

▼ What is Git?

- Git is a distributed version control system
- That means, it keep tracks of your file locally. There is no need to a centralized network
- But svn for example a centralized version control system
- Github on the other hand is a remote repository. It acts like a cloud based server where you can manage your repositories or share your code/files with others



▼ How to use git with git commands?

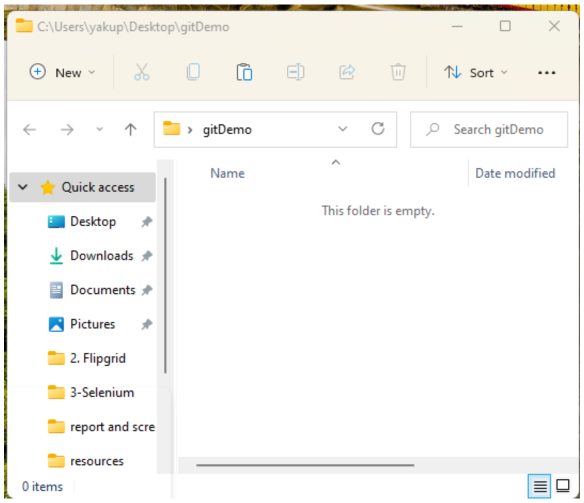
▼ installation

- Install git to your computer

▼ Creating Working Directory / Workspace

- Working directory or workspace is the folder where we keep track of changes using git
- First, we need to create a workspace

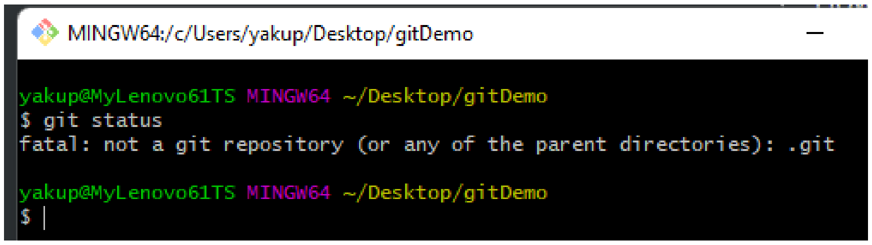
- Create an empty folder on Desktop which is going to be our workspace/directory



this is our workspace

▼ `git init`: Initialize the directory

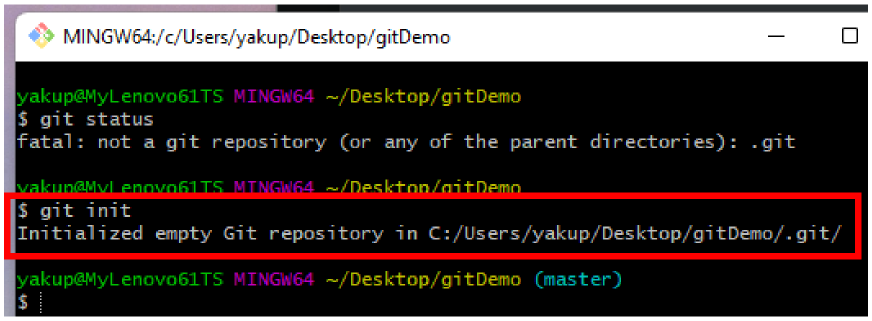
- Open the workspace/directory and right click-select "git bash here"
- If you directly type "git status", it'll give you "fatal: not a git repository" error



```
MINGW64:/c/Users/yakup/Desktop/gitDemo

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo
$ git status
fatal: not a git repository (or any of the parent directories): .git

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo
$ |
```

- The reason is that, we haven't initialized our workspace/directory as a "git directory/repository"
- git does not automatically takes a workspace/directory as a git repository.
- So we must tell git that this workspace/directory is gonna be our git repository/directory where we track the changes
- To initialize our workspace/directory as a git repository/directory, use "git init" command

- ***The "git init" command creates a new empty "git repository" or initializes an existing directory as a "git directory". Once the directory is a git directory you can run any git commands.***



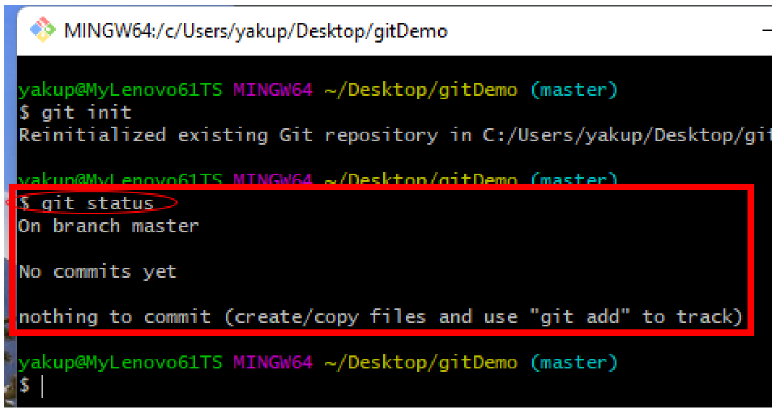
```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo
$ git status
fatal: not a git repository (or any of the parent directories): .git

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo
$ git init
Initialized empty Git repository in C:/Users/yakup/Desktop/gitDemo/.git/

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$
```

- After running "git init" command, a hidden folder ".git" is created in our workspace in which git metadata are saved
- We can re-run "git init" command on an already initialized git repository and it will change nothing

---

▼ `git status` : current status of the repo

- "git status" command returns us the current state of the directory
- *It lists which files are staged, unstaged, and untracked.*
  - We don't have any file in the working directory yet



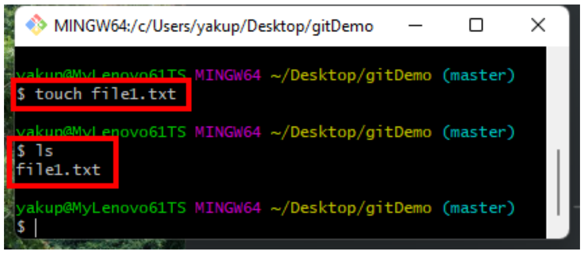
```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git init
Reinitialized existing Git repository in C:/Users/yakup/Desktop/gi

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

current state of the directory

▼ `git status` : "untracked files"

▼ touch <folder name>: creates a new file

- In order to create a file in the directory, we can use "touch <folder name>" command
- We can run "ls" command to see what we have in the directory

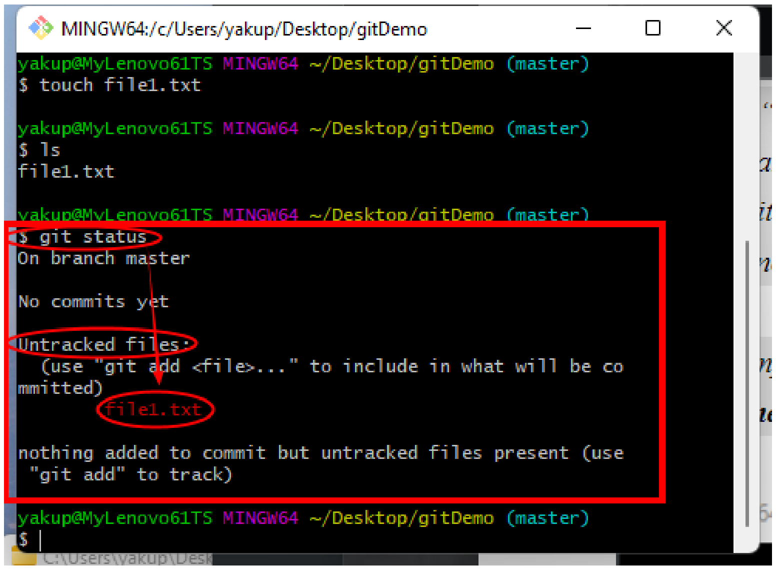


```
MINGW64:/c/Users/yakup/Desktop/gitDemo

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ touch file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ ls
file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

as we see here file1.txt is created after "touch" command

- if we re-run "git status" after we created file1.txt, it'll show that we have untracked files within the workspace

  "git status" command *lists which files are staged, unstaged, and **untracked**.*



```
MINGW64:/c/Users/yakup/Desktop/gitDemo
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ touch file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ ls
file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be co
mmitted)
        file1.txt

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

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

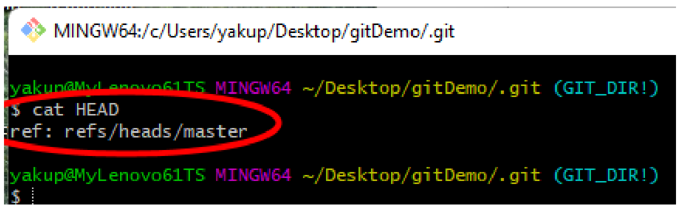
we have untracked files here

- *Untracked files are files that are present in the working directory but have not been added to the repo’s tracking index or staging area.*

- *In short, git does not track any changes to files till we explicitly add them to staging area using the **git add** command.*

▼ `cat HEAD` : to show HEAD

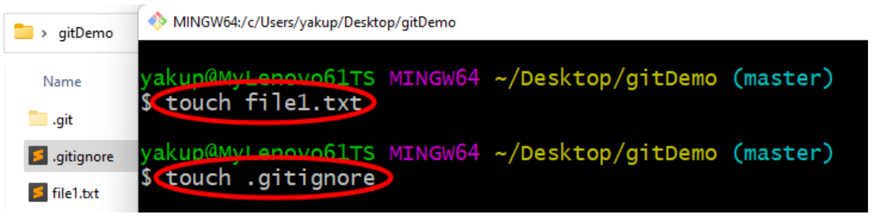
- *Inside **.git** folder we have a file named HEAD which **points out the last commit or tip in the current checkout branch***
- Inside the .git folder, open git bash here and type "cat HEAD" command



```
MINGW64:/c/Users/yakup/Desktop/gitDemo/.git

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo/.git (GIT_DIR!)
$ cat HEAD
ref: refs/heads/master

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo/.git (GIT_DIR!)
$
```

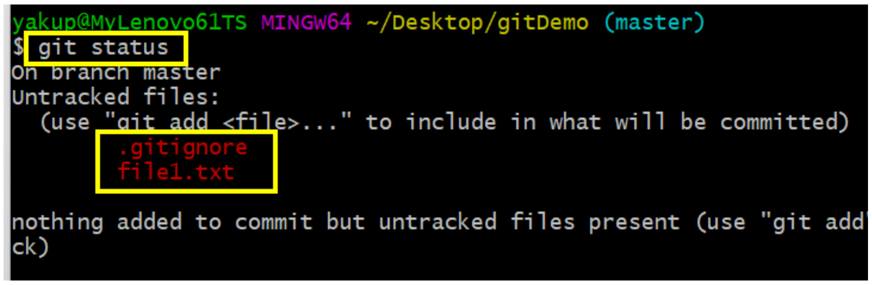
▼ `.gitignore` file : not to track files

- git will not keep track of those files we add to .gitignore file



```
MINGW64:/c/Users/yakup/Desktop/gitDemo

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ touch file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ touch .gitignore
```

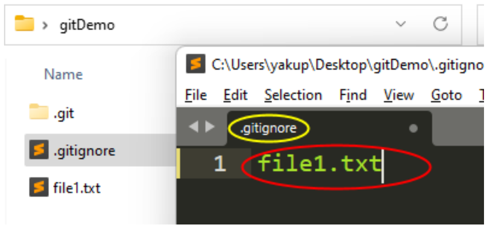
create file1.txt and .gitignore files



```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        file1.txt

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

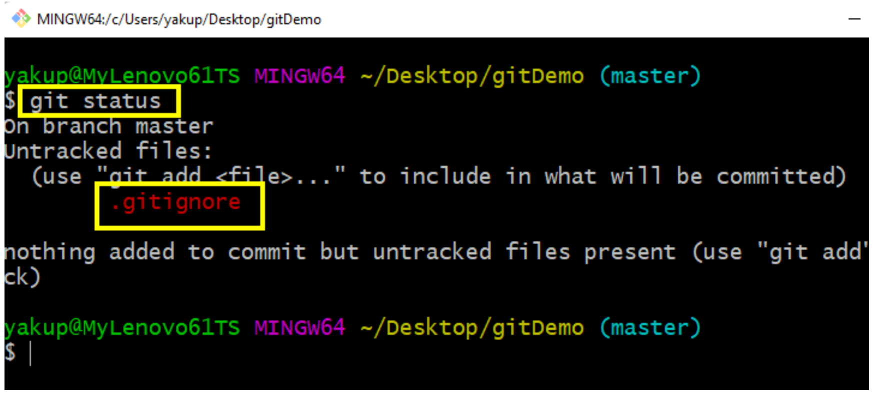
run git status command to see what we have. It says "file1.txt" is untracked

---


gitDemo
Name
.git
.gitignore
file1.txt
C:\Users\yakup\Desktop\gitDemo\.gitigno
File Edit Selection Find View Goto
.gitignore

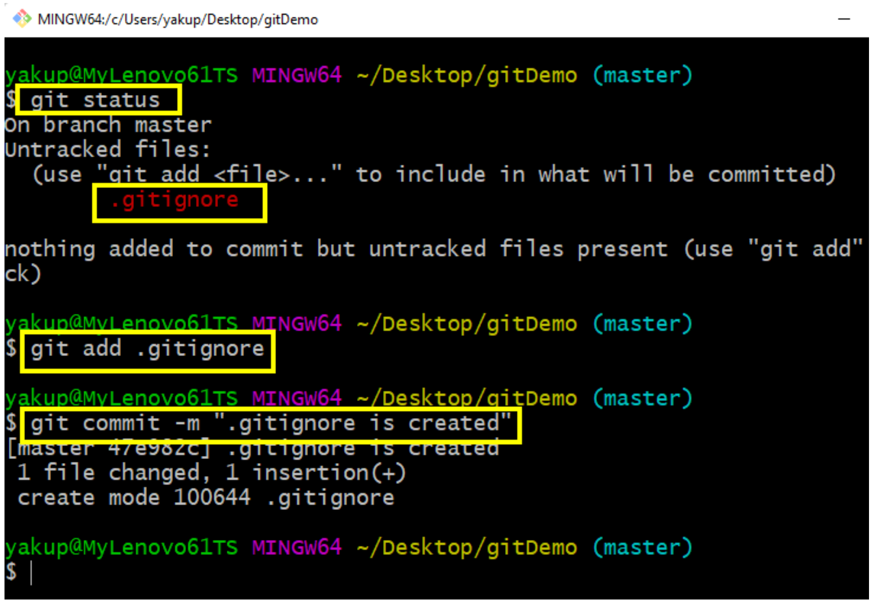
```
file1.txt
```

open .gitignore and write "file1.txt"



```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

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

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

after updating .gitignore, file1.txt is no longer tracked by git



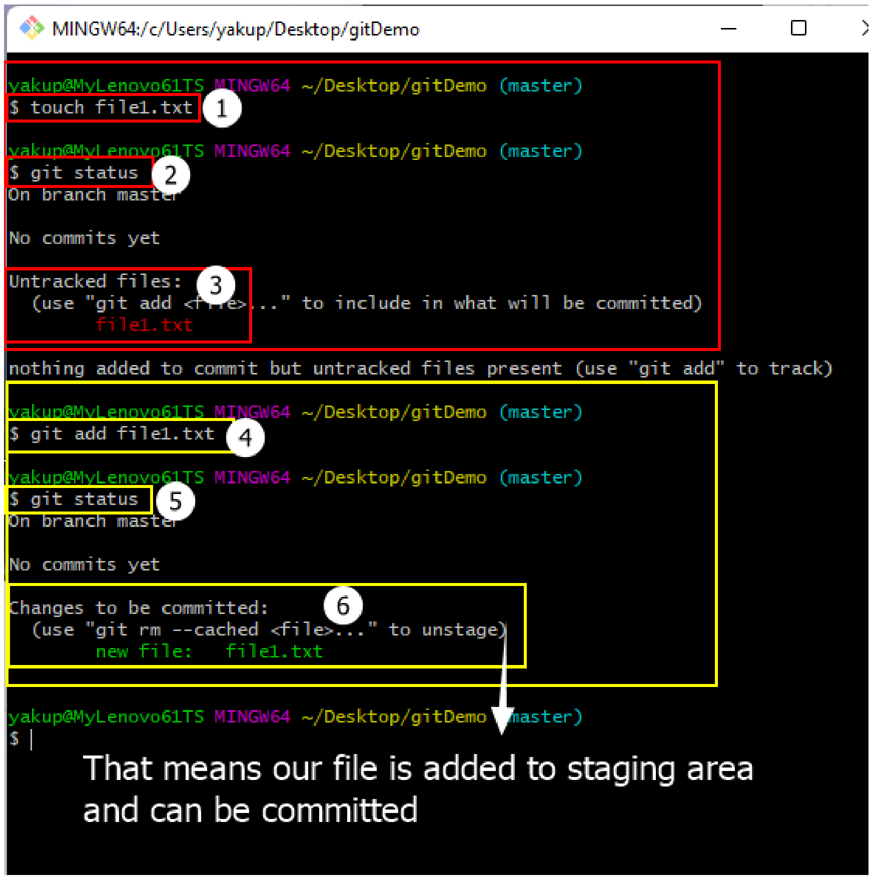
```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

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

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git add .gitignore

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git commit -m ".gitignore is created"
[master 47e982c] .gitignore is created
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

finally, add .gitignore to staging area and commit it

▼ `git add <file_name>` : to add staging area

- "git add" command is used to add untracked files *from the working directory to the Staging area*.

▼ *screenshot*



MINGW64:/c/Users/yakup/Desktop/gitDemo

```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ touch file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt

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

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git add file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

1
2
3
4
5
6
That means our file is added to staging area and can be committed

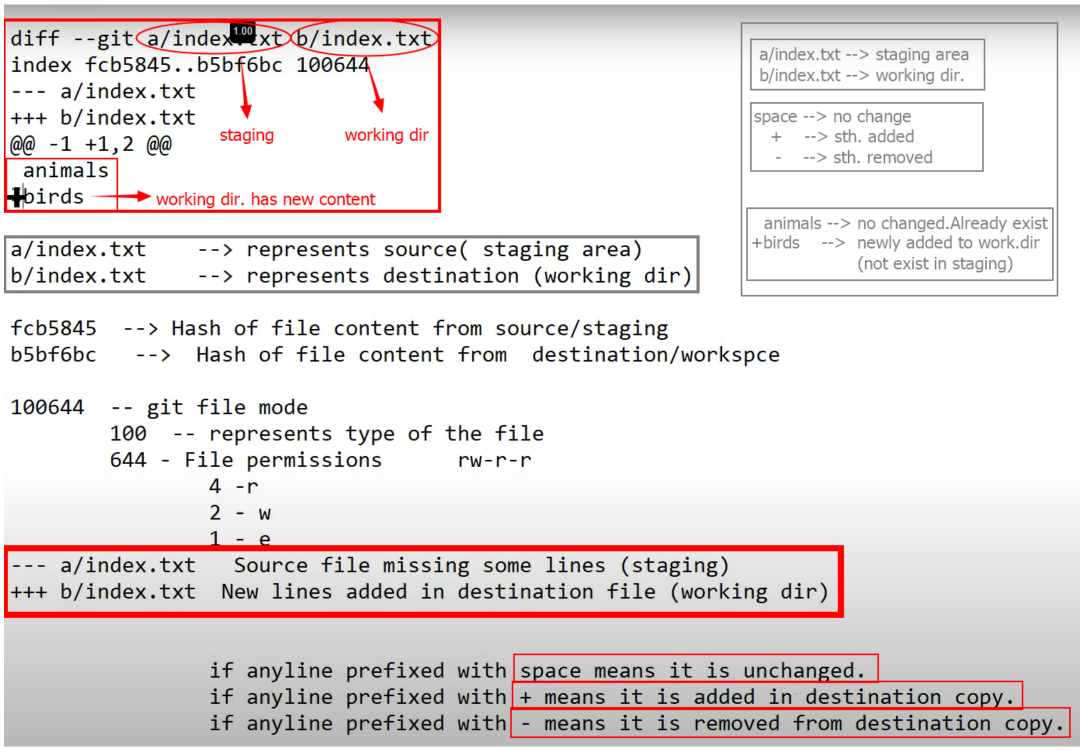
---

▼ `git diff` : to see the differences (between working dir. and staging area)

- `git diff HEAD <file_name> :` differences between working dir. and last commit
- `git diff --staged HEAD <file_name>` : differences between staging area and last commit
- `git diff <commit_id> <file_name>` : differences between a specific commit and working dir.
- `git diff --staged <commit_id> <file_name>` : differences between a specific commit and staging area

- `git diff <commit_id1> <commit_id2> <file_name>` : differences between two commits
- `git diff <branch_name1> <branch_name2> <file_name>` : differences between two branches
- `git diff <local_branch> <remote_branch> <file_name>` : differences between local and remote repo
- make some changes on the tracked file
- Open the "file1.txt" and make some changes on it and save
- Open git bash and type "git status", it'll show that there are some changes
- Then type "git diff" command to see what's been changed
- Finally type "git add file1.txt" to add the file staging area again??

▼ *screenshot*



```
diff --git a/index.txt b/index.txt
index fcb5845..b5bf6bc 100644
--- a/index.txt
+++ b/index.txt
@@ -1 +1,2 @@
 animals
+birds
```

```
a/index.txt    --> represents source( staging area)
b/index.txt    --> represents destination (working dir)
```

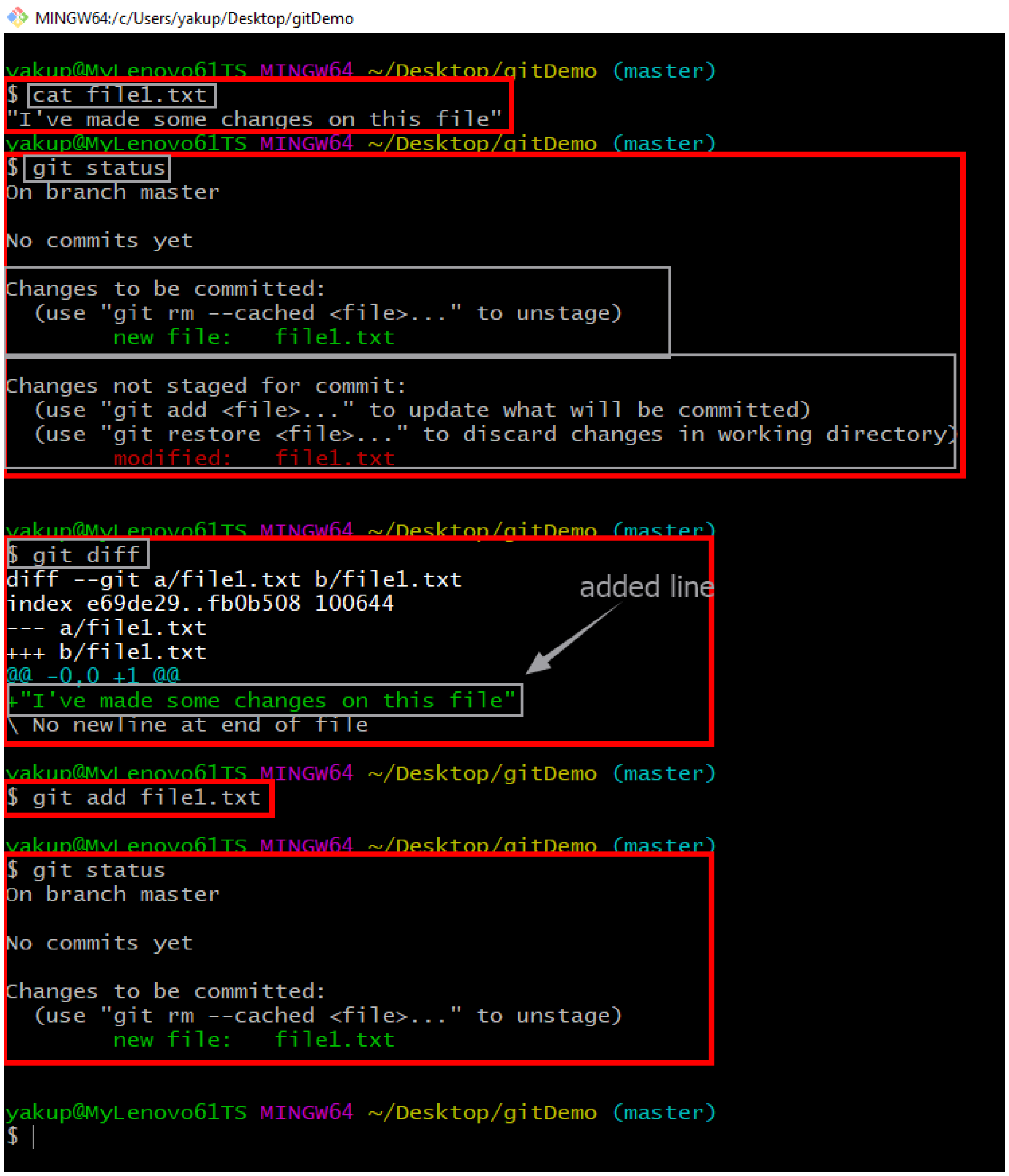
a/index.txt --> staging area
b/index.txt --> working dir.

space --> no change
+ --> sth. added
- --> sth. removed

animals --> no changed.Already exist
+birds --> newly added to work.dir (not exist in staging)

```
fcb5845   --> Hash of file content from source/staging
b5bf6bc    -->  Hash of file content from  destination/workspce

100644  -- git file mode
        100  -- represents type of the file
        644 - File permissions      rw-r-r
                4 -r
                2 - w
                1 - e
--- a/index.txt   Source file missing some lines (staging)
+++ b/index.txt  New lines added in destination file (working dir)


                if anyline prefixed with space means it is unchanged.
                if anyline prefixed with + means it is added in destination copy.
                if anyline prefixed with - means it is removed from destination copy.
```

▼ *screenshot*



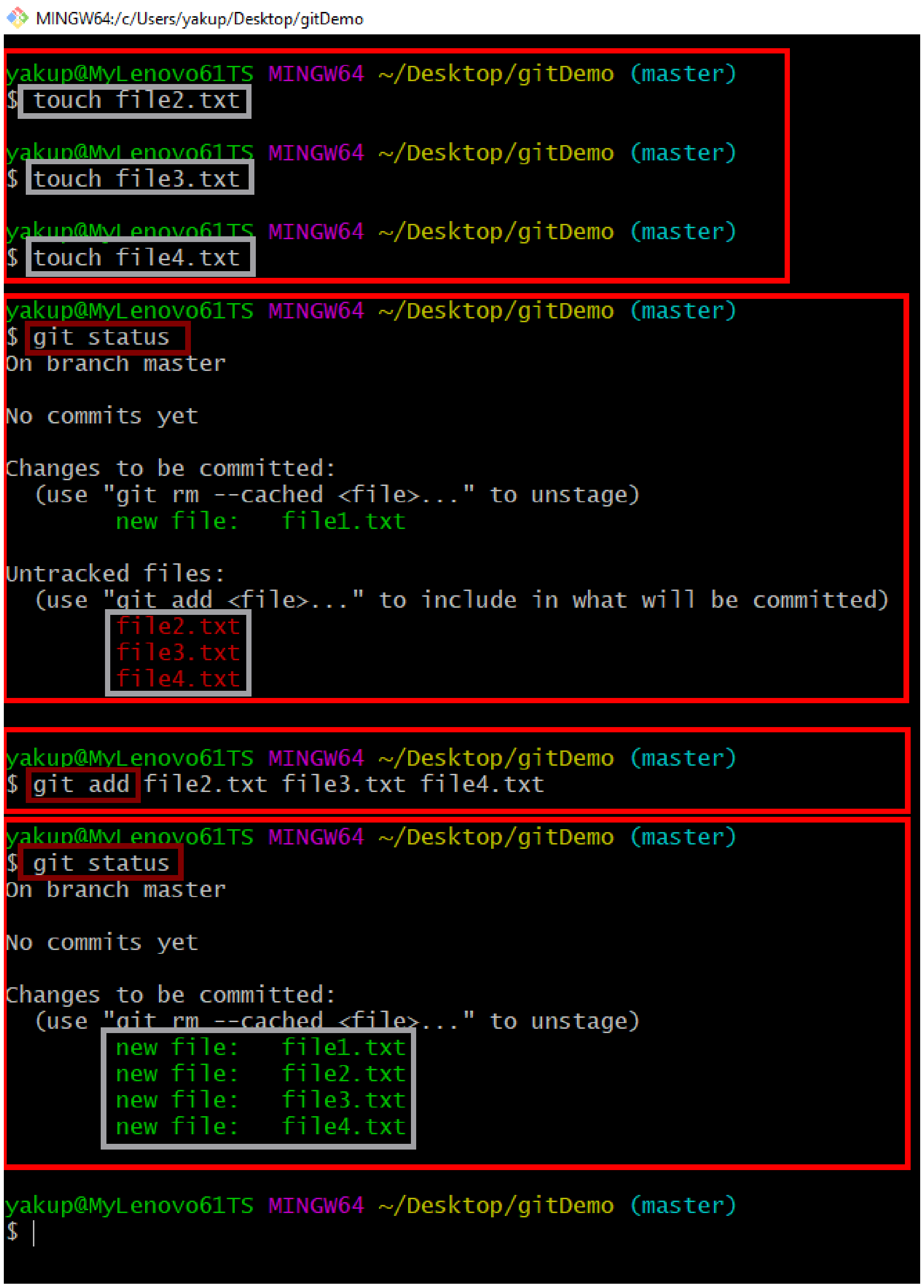
MINGW64:/c/Users/yakup/Desktop/gitDemo

```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ cat file1.txt
"I've made some changes on this file"
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt


yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git diff
diff --git a/file1.txt b/file1.txt
index e69de29..fb0b508 100644
--- a/file1.txt
+++ b/file1.txt
@@ -0,0 +1 @@
+"I've made some changes on this file"
\ No newline at end of file

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git add file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt


yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

▼ `git add .` : adding multiple files to staging area (only from current directory)

- Create multiple files in the directory
- Type git add and provide file names with a space between them

▼ *screenshot*



```
MINGW64:/c/Users/yakup/Desktop/gitDemo

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ touch file2.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ touch file3.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ touch file4.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file2.txt
        file3.txt
        file4.txt


yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git add file2.txt file3.txt file4.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt
        new file:   file2.txt
        new file:   file3.txt
        new file:   file4.txt


yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

▼ `git add -A` : adding multiple files to staging area (including higher directories)

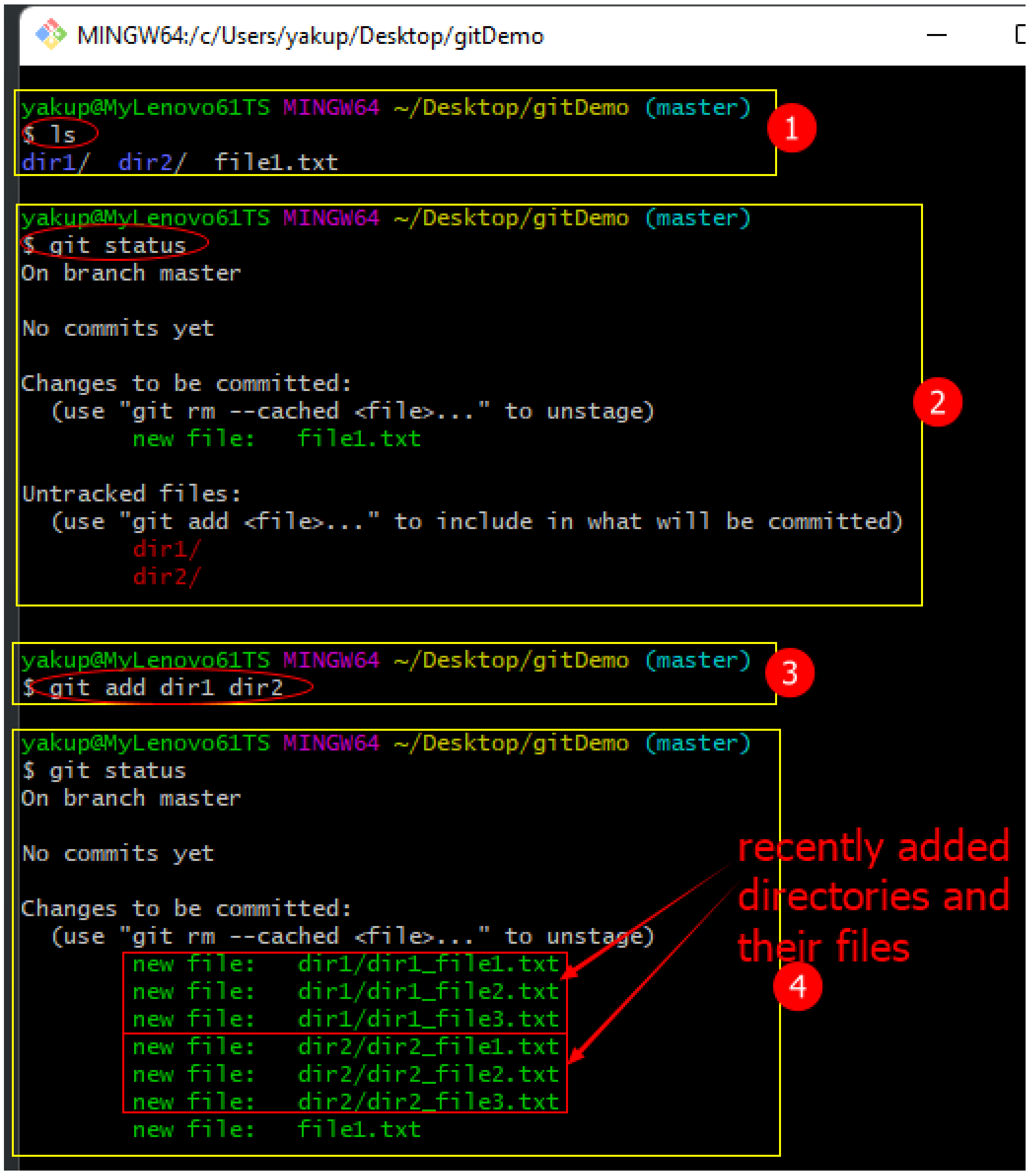
- Doing almost the same thing as "git add ." command, additionally includes files/changes from higher directories

---

▼ `git add <directory_name>`: to add directories

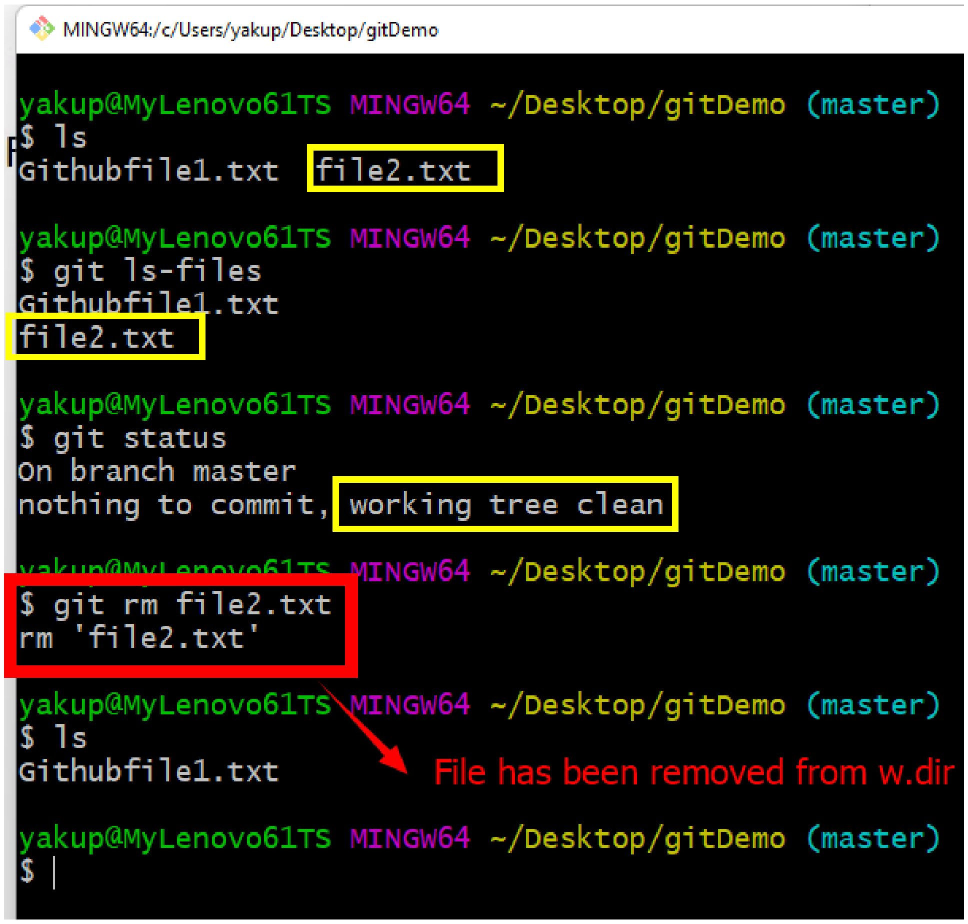
- Using the same git add command, we can add new directories to staging area
- Create tow new directories (dir1 and dir2) with some new files in each, within the working directory
- Use "git add dir1 dir2" command to add these new directories and their files in each

▼ *screenshot*


MINGW64:/c/Users/yakup/Desktop/gitDemo
1
2
3
recently added directories and their files
4

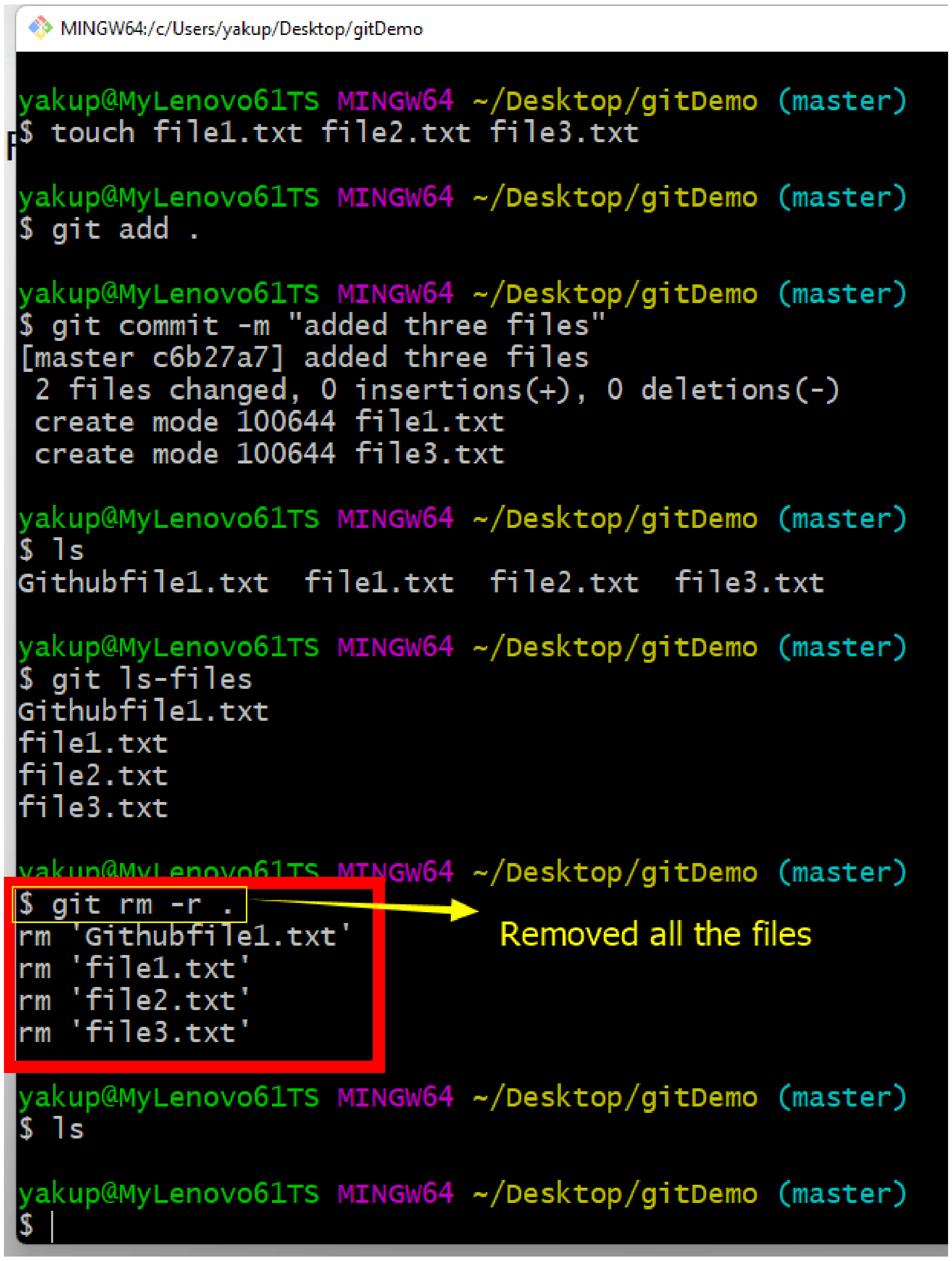
```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ ls
dir1/  dir2/  file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        dir1/
        dir2/

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git add dir1 dir2

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   dir1/dir1_file1.txt
        new file:   dir1/dir1_file2.txt
        new file:   dir1/dir1_file3.txt
        new file:   dir2/dir2_file1.txt
        new file:   dir2/dir2_file2.txt
        new file:   dir2/dir2_file3.txt
        new file:   file1.txt
```

- `git ls-files` : shows files in staging area
- `git rm <file_name>` : removes a file from both working dir. and staging area
  - in order to use this command, file must be added to staging and be committed


MINGW64:/c/Users/yakup/Desktop/gitDemo

```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ ls
Githubfile1.txt  file2.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git ls-files
Githubfile1.txt
file2.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master
nothing to commit, working tree clean

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git rm file2.txt
rm 'file2.txt'

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ ls
Githubfile1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

- `git rm -r .` : removes all files from both working dir. and staging area
  - Files must be in staging area and committed before being removed


MINGW64:/c/Users/yakup/Desktop/gitDemo

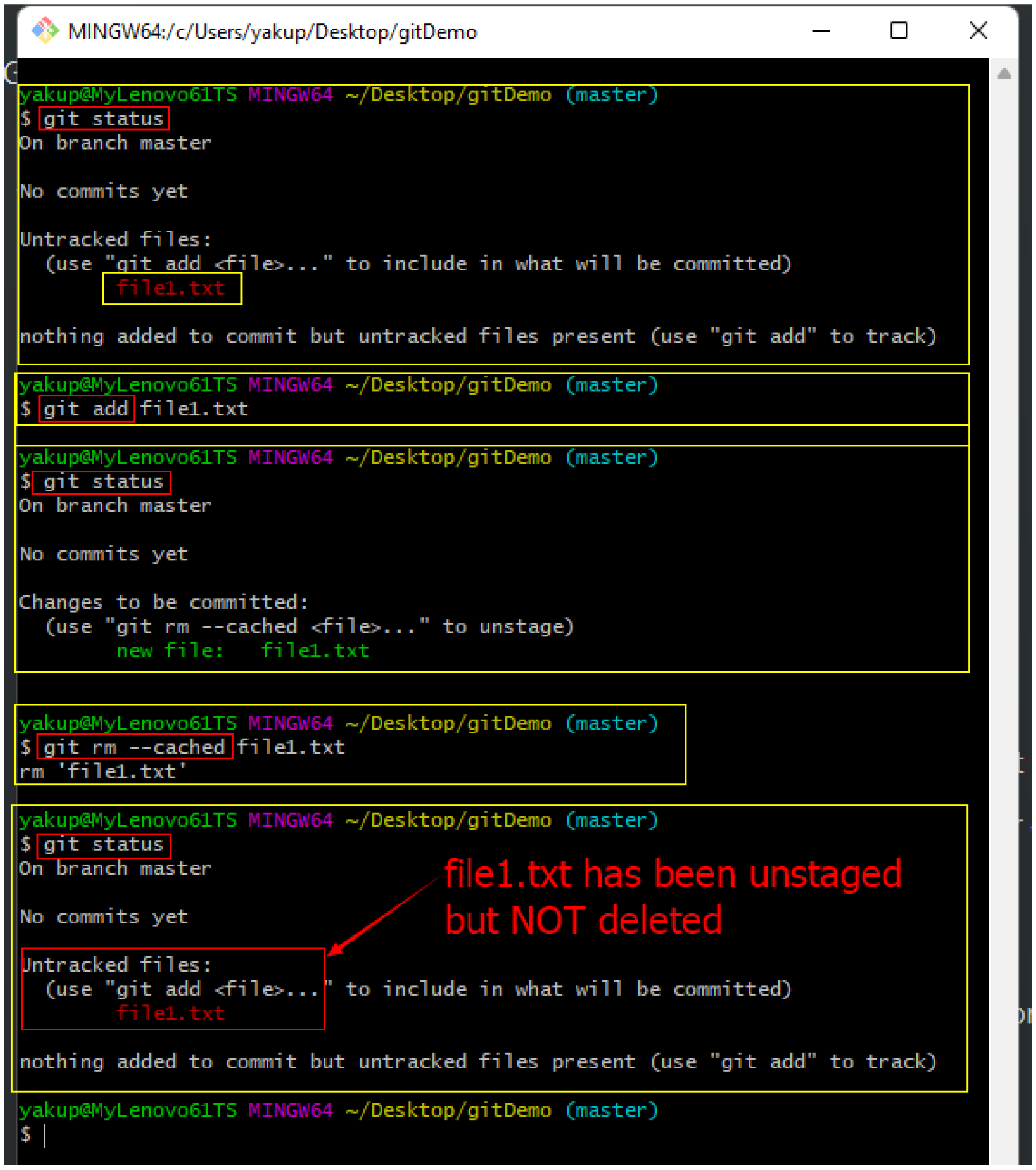
```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ touch file1.txt file2.txt file3.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git add .

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git commit -m "added three files"
[master c6b27a7] added three files
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1.txt
 create mode 100644 file3.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ ls
Githubfile1.txt  file1.txt  file2.txt  file3.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git ls-files
Githubfile1.txt
file1.txt
file2.txt
file3.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git rm -r .
rm 'Githubfile1.txt'
rm 'file1.txt'
rm 'file2.txt'
rm 'file3.txt'

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ ls

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

Removed all the files

▼ `git rm --cached <file_name>` : removes a file only from staging area

- It removes the file from staging area but NOT delete it from working directory

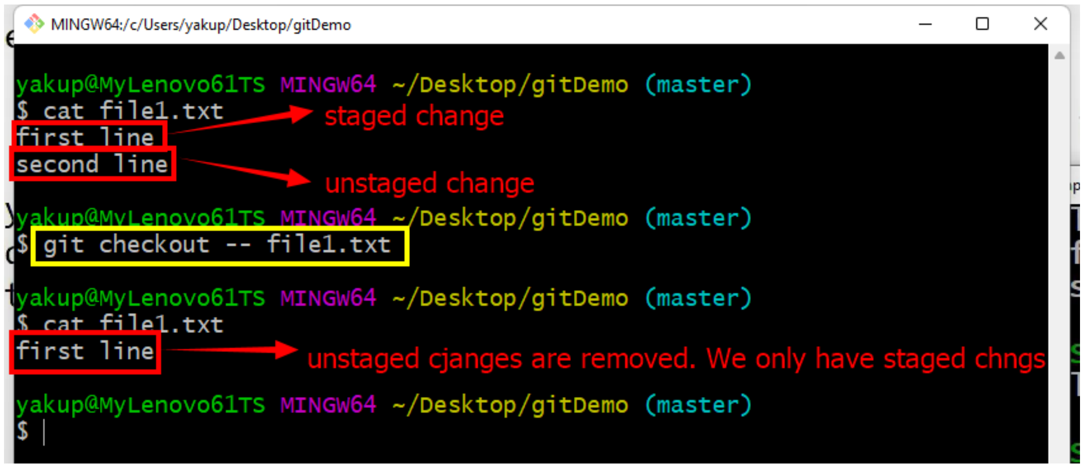
▼ screenshot



```
MINGW64:/c/Users/yakup/Desktop/gitDemo

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt

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

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git add file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt


yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git rm --cached file1.txt
rm 'file1.txt'

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt

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

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

▼ `git checkout -- <file_name>` : to remove unstaged changes

- Lets say we made some changes in the file, but not committed
- And we want to discard these recently added changes (go back to the state before adding)

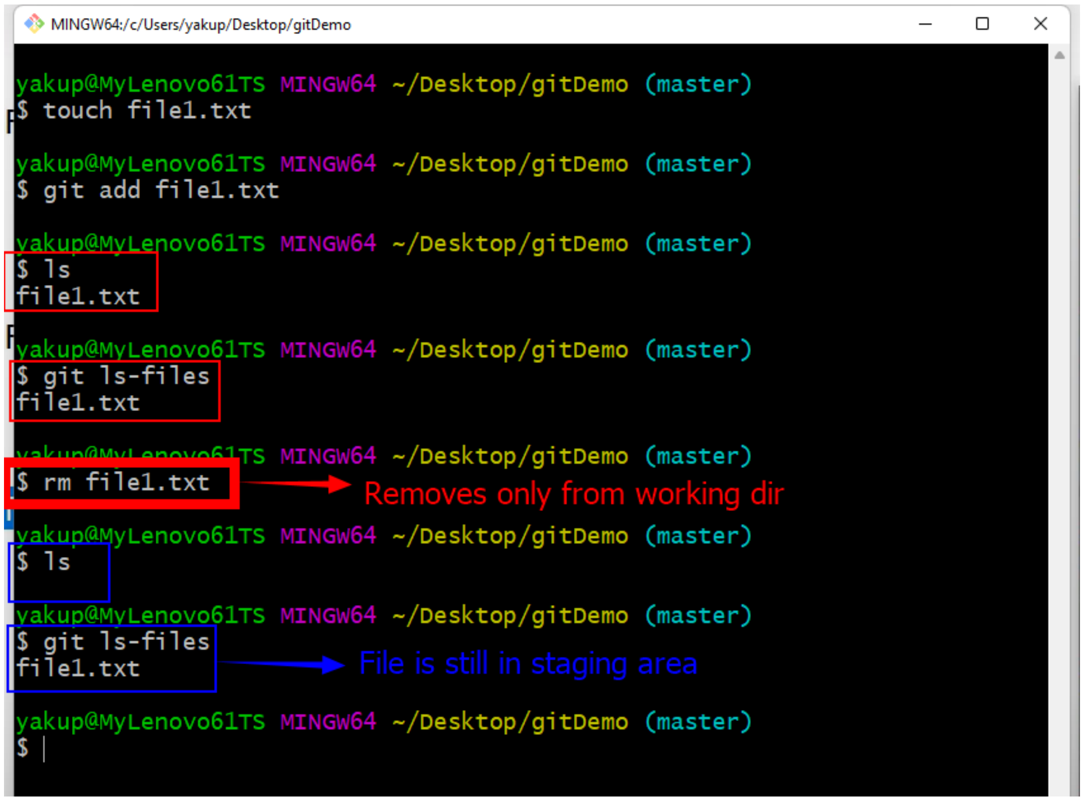

MINGW64:/c/Users/yakup/Desktop/gitDemo
staged change
unstaged change
unstaged cjanges are removed. We only have staged chngs

```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ cat file1.txt
first line
second line

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git checkout -- file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ cat file1.txt
first line

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

- In order to use checkout command;
  - file itself must be added to staging area
  - changes to be removed must not be added to staging area

▼ `rm <file_name>` : removes a file only from working dir.

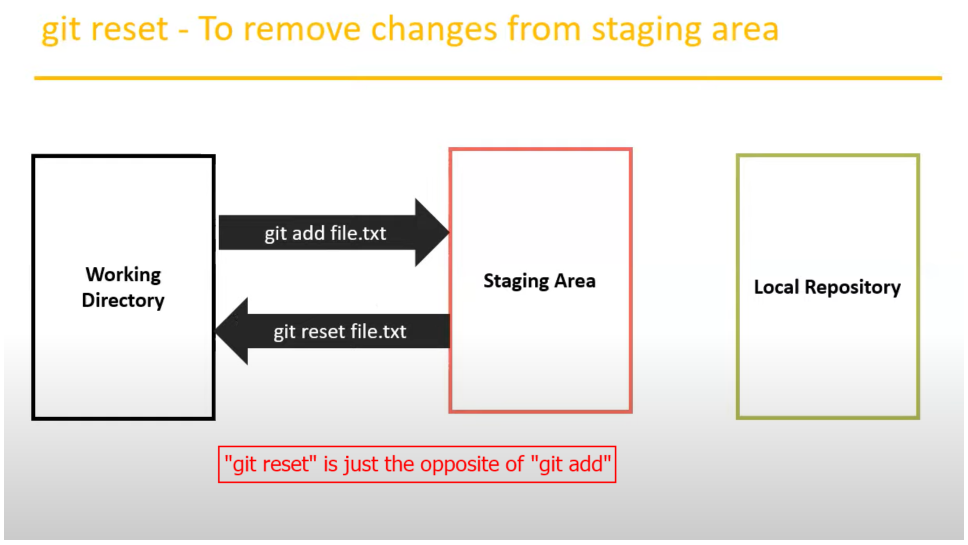
- only removes from working dir.
- But file is still in staging area

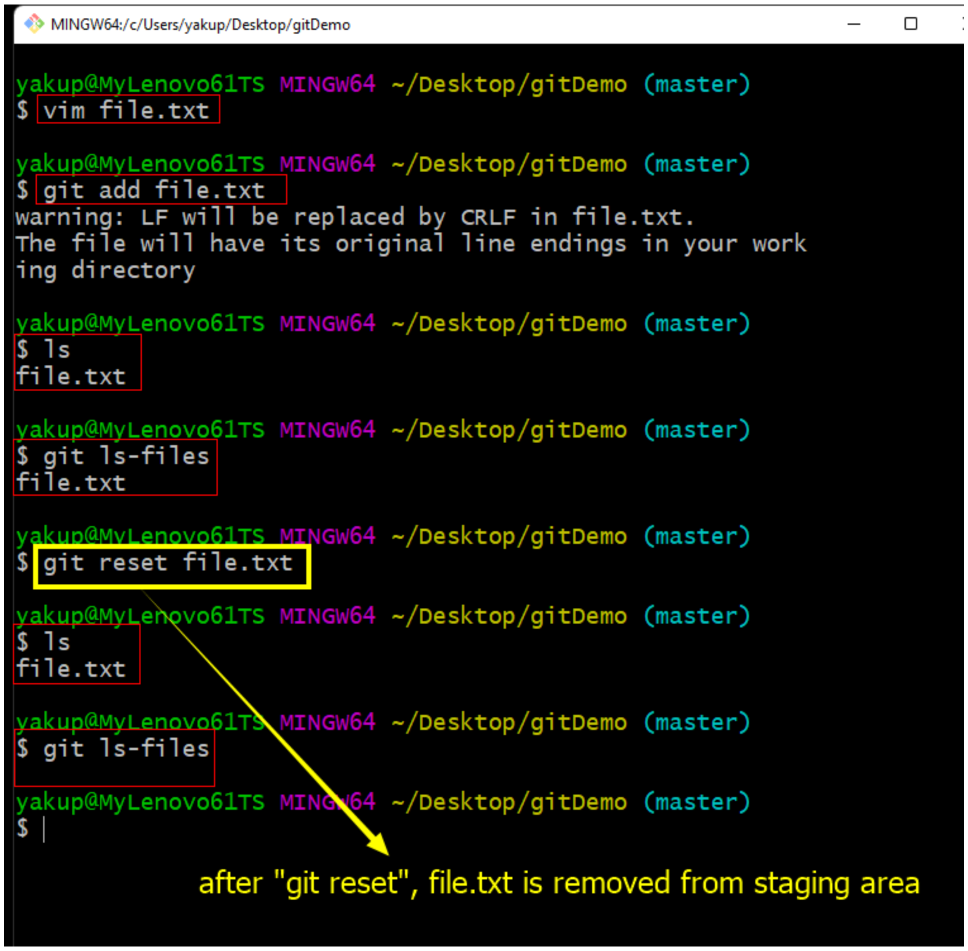


```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ touch file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git add file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ ls
file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git ls-files
file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ rm file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ ls

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git ls-files
file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

MINGW64:/c/Users/yakup/Desktop/gitDemo
Removes only from working dir
File is still in staging area

▼ `git reset <file_name>` : to remove changes from staging area

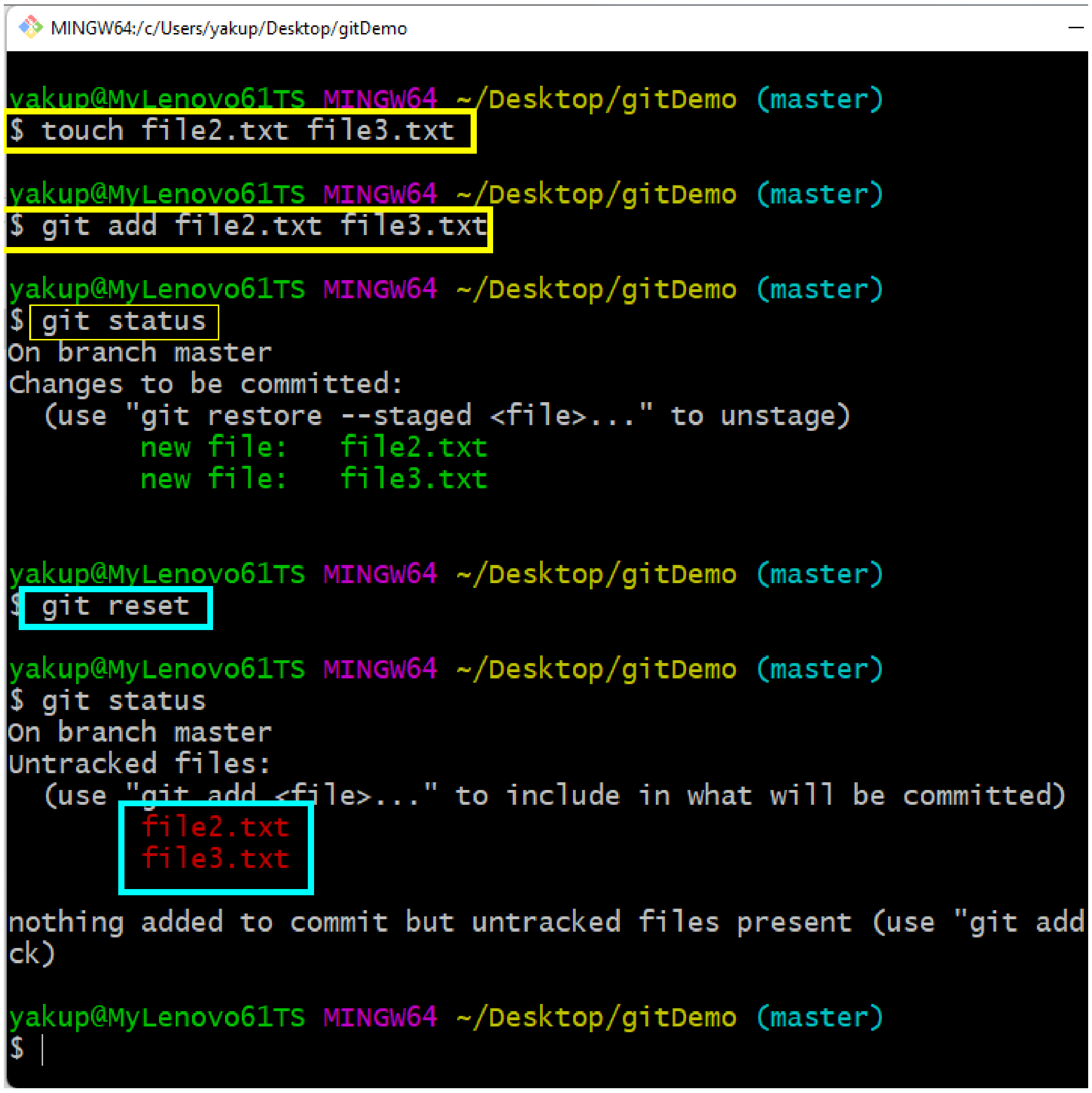

git reset - To remove changes from staging area
Working Directory
git add file.txt
git reset file.txt
Staging Area
Local Repository
"git reset" is just the opposite of "git add"



```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ vim file.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git add file.txt
warning: LF will be replaced by CRLF in file.txt.
The file will have its original line endings in your work
ing directory

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ ls
file.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git ls-files
file.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git reset file.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ ls
file.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git ls-files

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

after "git reset", file.txt is removed from staging area

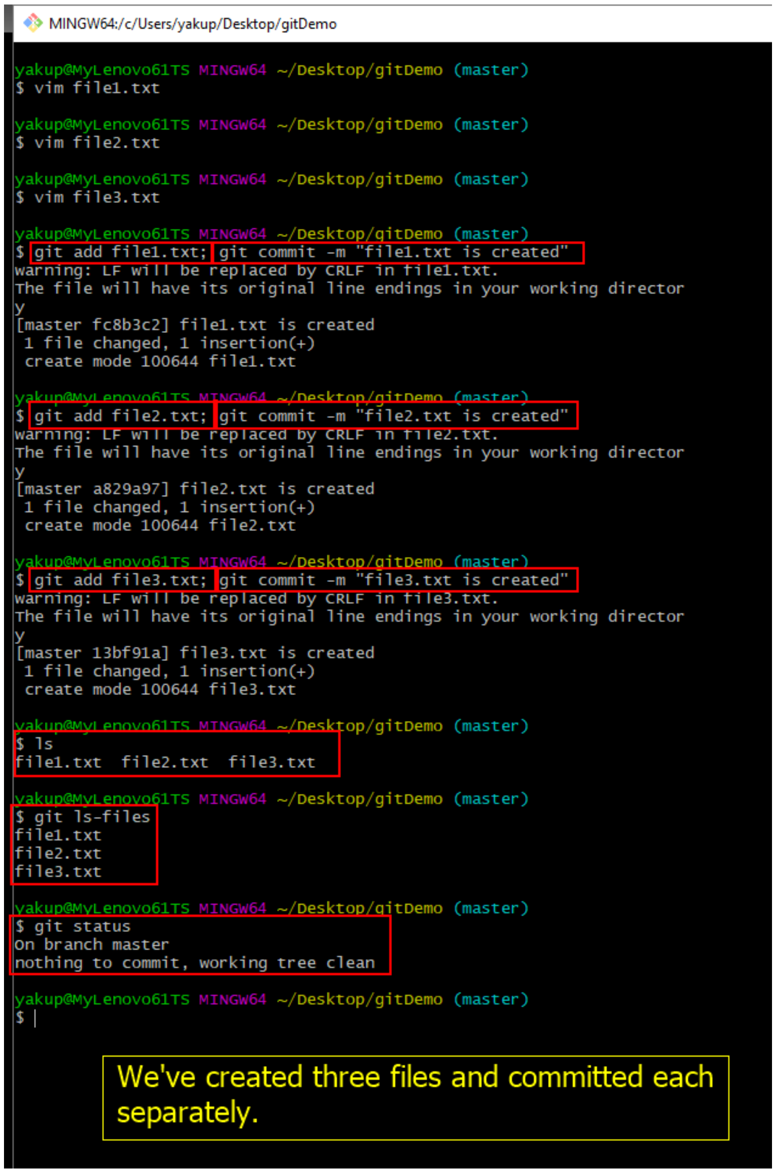
▼ `git reset` : to unstage all files



```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ touch file2.txt file3.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git add file2.txt file3.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   file2.txt
        new file:   file3.txt


yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git reset

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file2.txt
        file3.txt

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

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

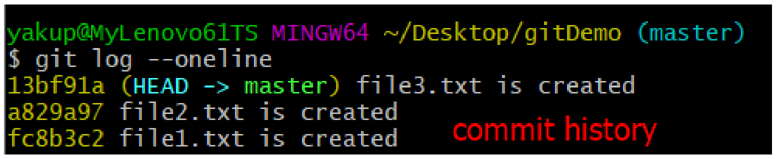
▼ `git reset --mixed <commit_id>` : to reset the file to the specified commit (also remove from staging

- to reset the file to the specified commit
- -- mixed: Removed changes from local repo also get removed from staging area, but NOT get removed from working dir.



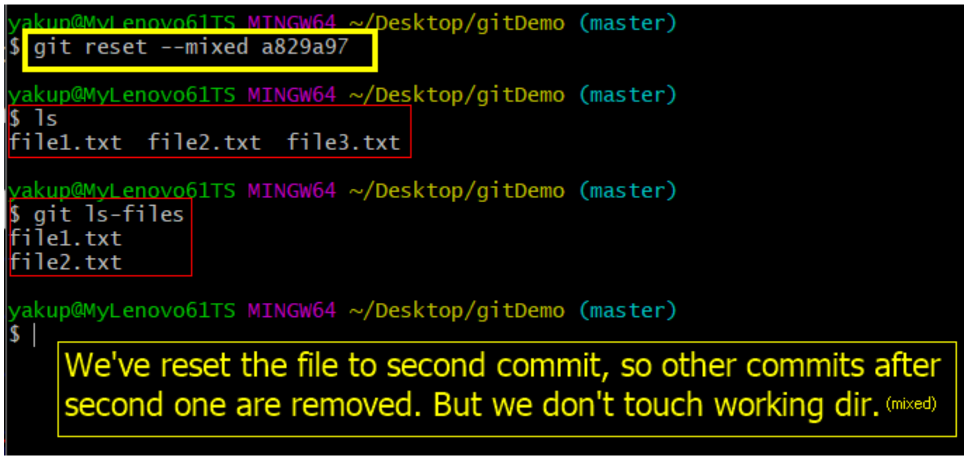
```
MINGW64:/c/Users/yakup/Desktop/gitDemo

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ vim file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ vim file2.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ vim file3.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git add file1.txt; git commit -m "file1.txt is created"
warning: LF will be replaced by CRLF in file1.txt.
The file will have its original line endings in your working director
y
[master fc8b3c2] file1.txt is created
 1 file changed, 1 insertion(+)
 create mode 100644 file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git add file2.txt; git commit -m "file2.txt is created"
warning: LF will be replaced by CRLF in file2.txt.
The file will have its original line endings in your working director
y
[master a829a97] file2.txt is created
 1 file changed, 1 insertion(+)
 create mode 100644 file2.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git add file3.txt; git commit -m "file3.txt is created"
warning: LF will be replaced by CRLF in file3.txt.
The file will have its original line endings in your working director
y
[master 13bf91a] file3.txt is created
 1 file changed, 1 insertion(+)
 create mode 100644 file3.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ ls
file1.txt  file2.txt  file3.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git ls-files
file1.txt
file2.txt
file3.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master
nothing to commit, working tree clean

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

We've created three files and committed each separately.



```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git log --oneline
13bf91a (HEAD -> master) file3.txt is created
a829a97 file2.txt is created
fc8b3c2 file1.txt is created
```

commit history

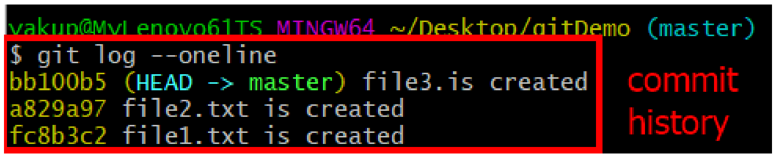


```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git reset --mixed a829a97

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ ls
file1.txt  file2.txt  file3.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git ls-files
file1.txt
file2.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

We've reset the file to second commit, so other commits after second one are removed. But we don't touch working dir. (mixed)

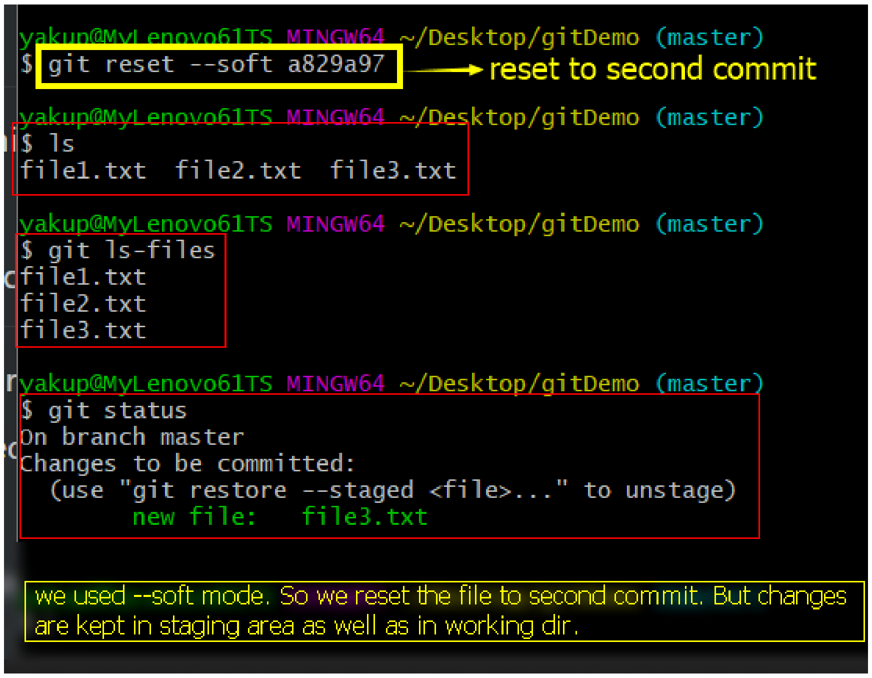
▼ `git reset --soft <commit_id>` **:** to reset the file to specified commit (keep in staging)

- This time, again we'll reset the file to specified commit
- But discarded changes will be kept in staging area as well as in working dir
- Only other commits are removed



```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git log --oneline
bb100b5 (HEAD -> master) file3.is created
a829a97 file2.txt is created
fc8b3c2 file1.txt is created
```

commit history

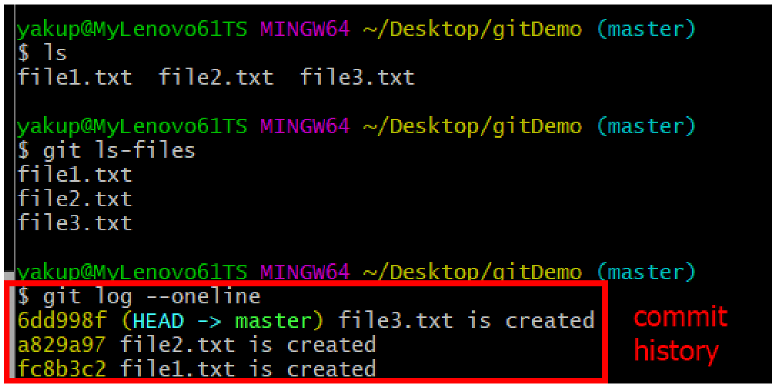


```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git reset --soft a829a97

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ ls
file1.txt  file2.txt  file3.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git ls-files
file1.txt
file2.txt
file3.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   file3.txt
```

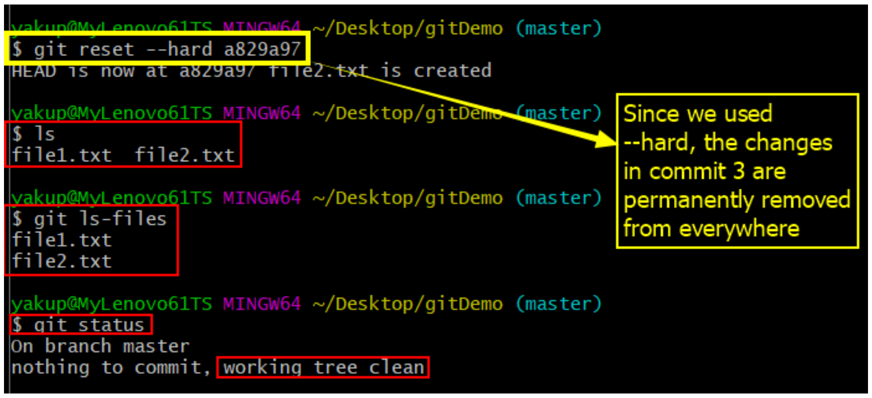
reset to second commit
we used --soft mode. So we reset the file to second commit. But changes are kept in staging area as well as in working dir.

▼ `git reset --hard <commit_id>` : to reset the file to specified commit (also remove from working dir)

- It's a dangerous command compared to other reset commands, there is no turning back
- This time, again we'll reset the file to specified commit
- But discarded changes will be removed from everywhere permanently (from working dir. and staging)
- Other commits are also removed



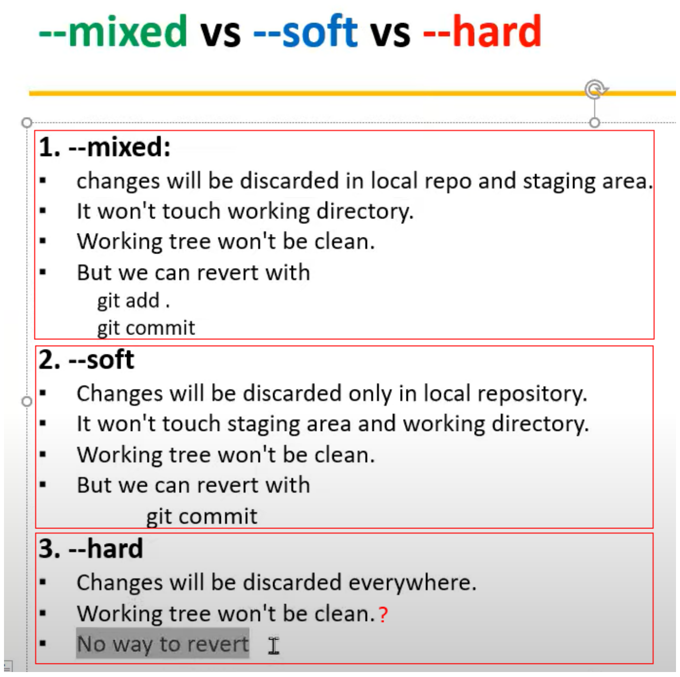
```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ ls
file1.txt  file2.txt  file3.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git ls-files
file1.txt
file2.txt
file3.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git log --oneline
6dd998f (HEAD -> master) file3.txt is created
a829a97 file2.txt is created
fc8b3c2 file1.txt is created
```



```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git reset --hard a829a97
HEAD is now at a829a97 file2.txt is created

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ ls
file1.txt  file2.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git ls-files
file1.txt
file2.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Since we used --hard, the changes in commit 3 are permanently removed from everywhere

- `git reset --mixed --sof --hard` : differences



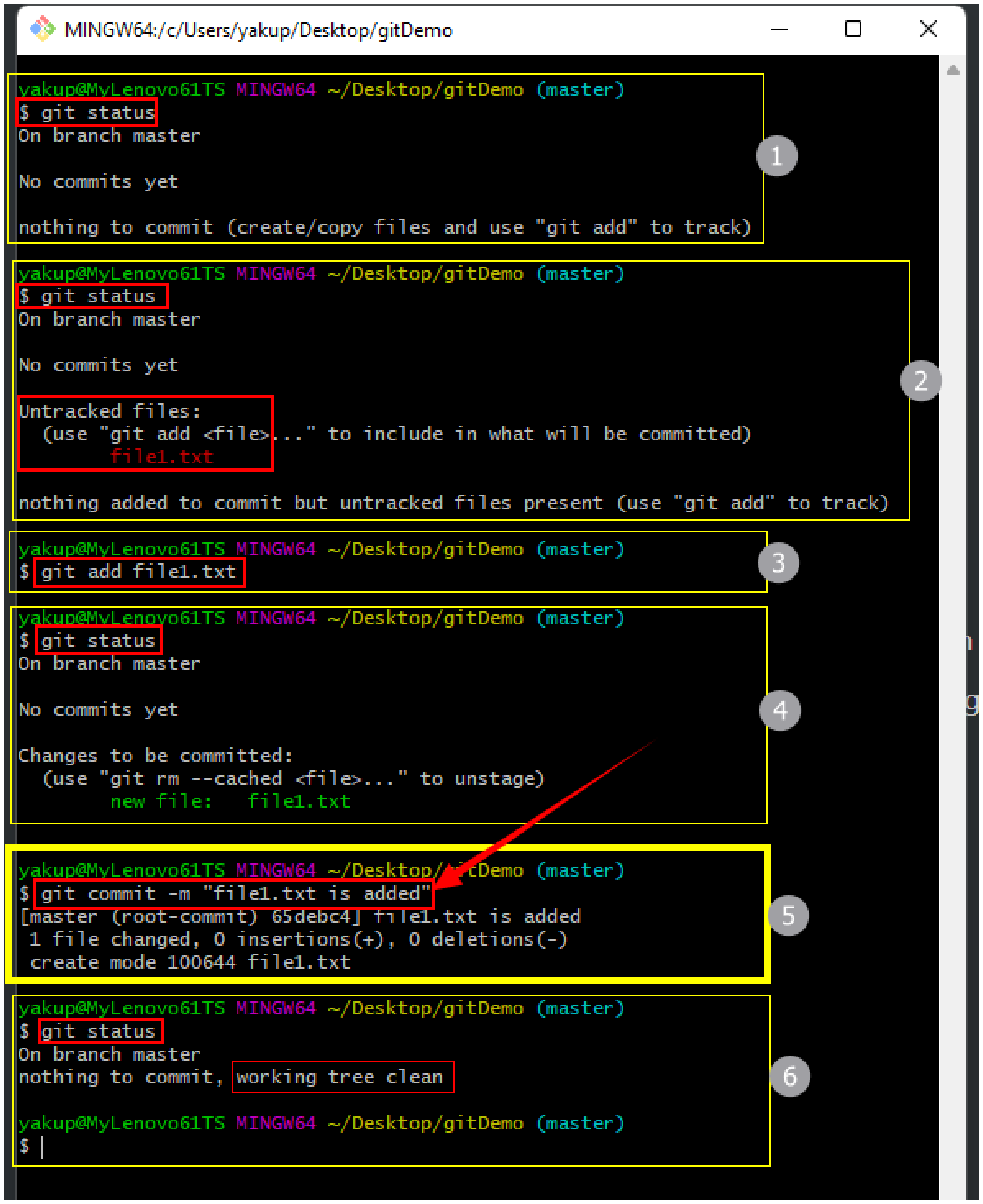
--mixed vs --soft vs --hard

1. --mixed:
- changes will be discarded in local repo and staging area.
- It won't touch working directory.
- Working tree won't be clean.
- But we can revert with
  git add .
  git commit

2. --soft
- Changes will be discarded only in local repository.
- It won't touch staging area and working directory.
- Working tree won't be clean.
- But we can revert with
  git commit

3. --hard
- Changes will be discarded everywhere.
- Working tree won't be clean.?
- No way to revert

- `git commit -m <commit_message>` : to commit a change
  - create a new file,
  - add it to staging area,
  - And commit it
  - *screenshot*

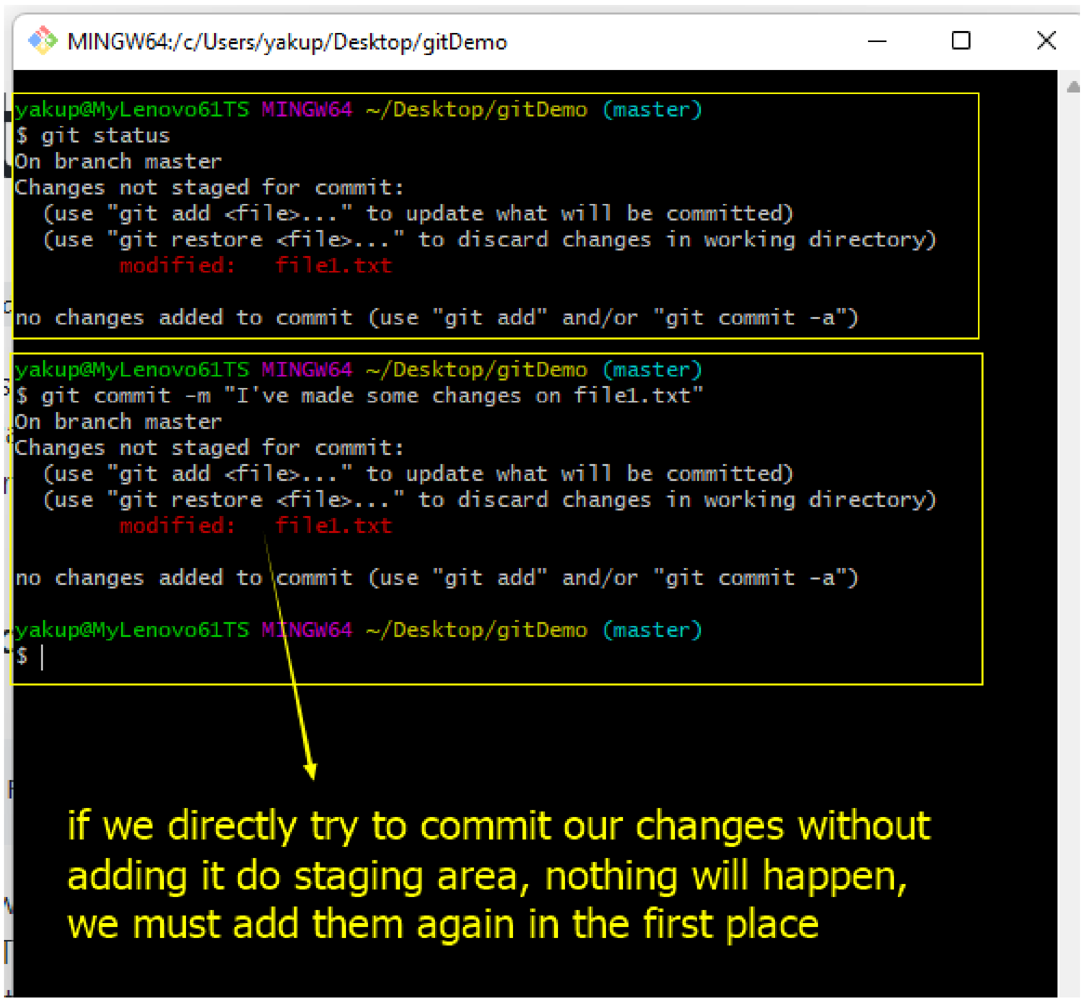


```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt

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

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git add file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt


yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git commit -m "file1.txt is added"
[master (root-commit) 65debc4] file1.txt is added
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master
nothing to commit, working tree clean

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

- Make some changes on the recently created file
- Add it to staging area again
- Commit it

💡 Even though we've already added our new file to staging area, we can't commit the changes in the new file unless we add it again to the staging area

▼ *screenshot1*

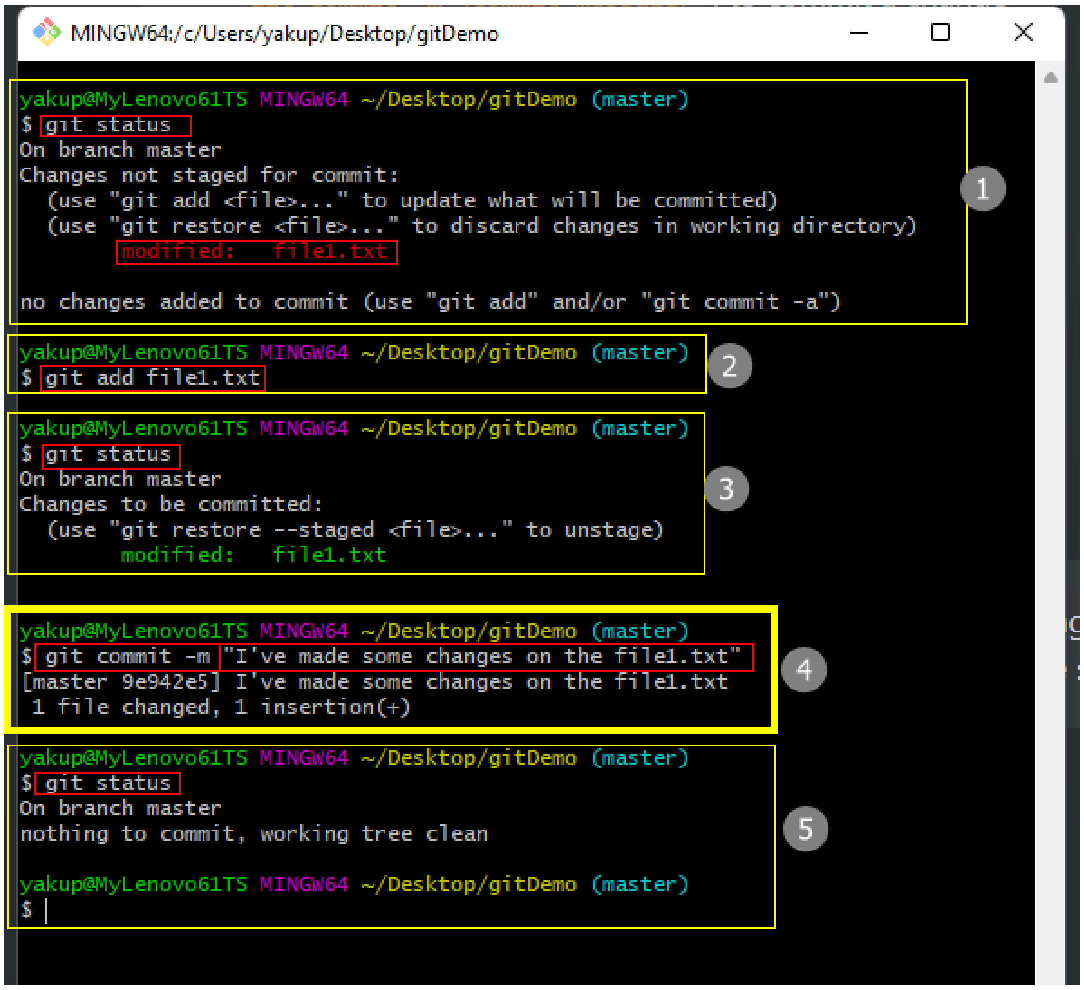


MINGW64:/c/Users/yakup/Desktop/gitDemo

```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt

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

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git commit -m "I've made some changes on file1.txt"
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt

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

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

if we directly try to commit our changes without adding it do staging area, nothing will happen, we must add them again in the first place

▼ *screenshot2*

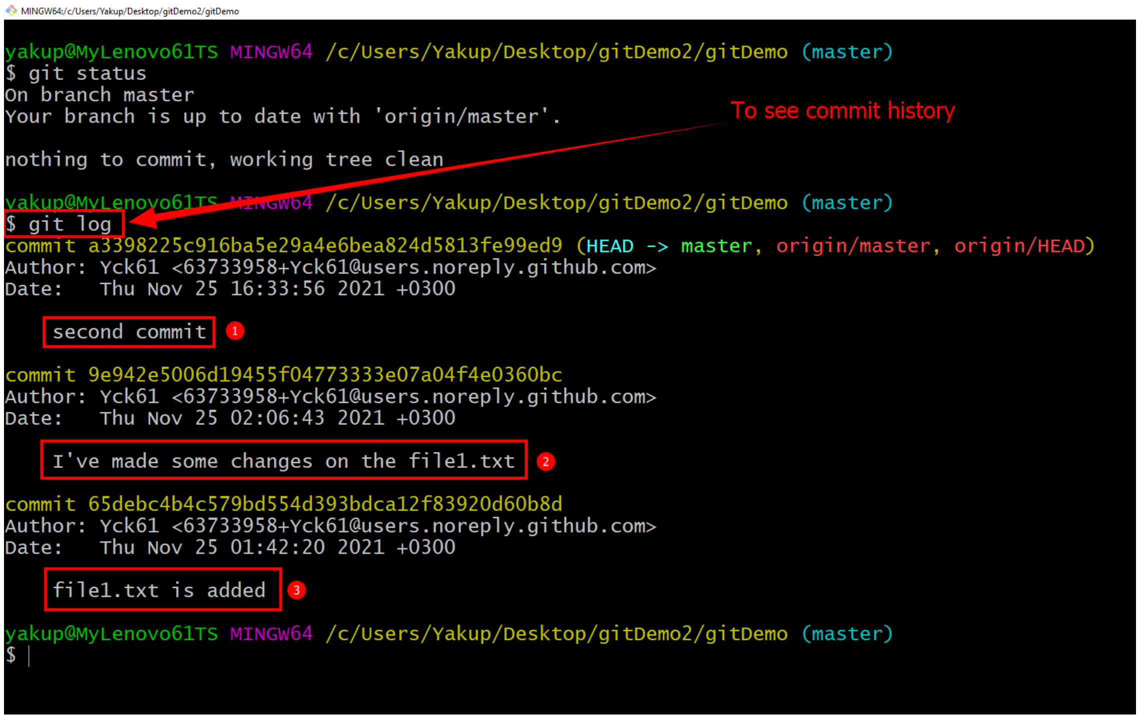


```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt

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

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git add file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git commit -m "I've made some changes on the file1.txt"
[master 9e942e5] I've made some changes on the file1.txt
 1 file changed, 1 insertion(+)

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master
nothing to commit, working tree clean

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

▼ `git commit -a -m <commit_message>` : add staging area and commit in one line

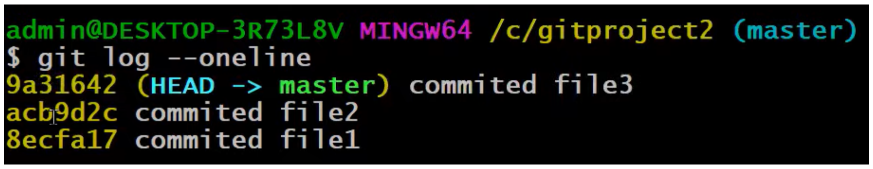
- When we make a change on an existing file, we can add it to staging area and commit in one line
- But file itself must be added to staging area before

▼ `git log` : to see commit history



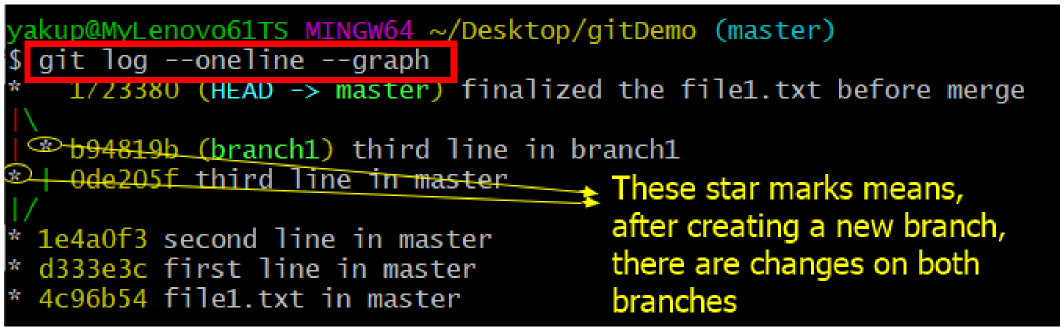
```
yakup@MyLenovo61TS MINGW64 /c/Users/Yakup/Desktop/gitDemo2/gitDemo (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

yakup@MyLenovo61TS MINGW64 /c/Users/Yakup/Desktop/gitDemo2/gitDemo (master)
$ git log
commit a3398225c916ba5e29a4e6bea824d5813fe99ed9 (HEAD -> master, origin/master, origin/HEAD)
Author: Yck61 <63733958+Yck61@users.noreply.github.com>
Date:   Thu Nov 25 16:33:56 2021 +0300

    second commit

commit 9e942e5006d19455f04773333e07a04f4e0360bc
Author: Yck61 <63733958+Yck61@users.noreply.github.com>
Date:   Thu Nov 25 02:06:43 2021 +0300

    I've made some changes on the file1.txt

commit 65debc4b4c579bd554d393bdca12f83920d60b8d
Author: Yck61 <63733958+Yck61@users.noreply.github.com>
Date:   Thu Nov 25 01:42:20 2021 +0300

    file1.txt is added

yakup@MyLenovo61TS MINGW64 /c/Users/Yakup/Desktop/gitDemo2/gitDemo (master)
$ |
```

- `git log --oneline` : to see the commit history in one line



```
admin@DESKTOP-3R73L8V MINGW64 /c/gitproject2 (master)
$ git log --oneline
9a31642 (HEAD -> master) commited file3
acb9d2c commited file2
8ecfa17 commited file1
```

- `git log --oneline <branch_name>` : to see commit history just for one particular branch
- `git log --oneline --graph` : to see commit history in graphical way

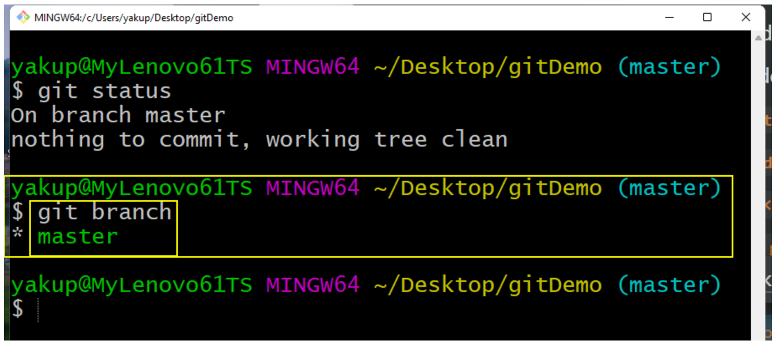


```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git log --oneline --graph
*   1723380 (HEAD -> master) finalized the file1.txt before merge
|\
| * b94819b (branch1) third line in branch1
* | 0de205f third line in master
|/
* 1e4a0f3 second line in master
* d333e3c first line in master
* 4c96b54 file1.txt in master
```

These star marks means, after creating a new branch, there are changes on both branches

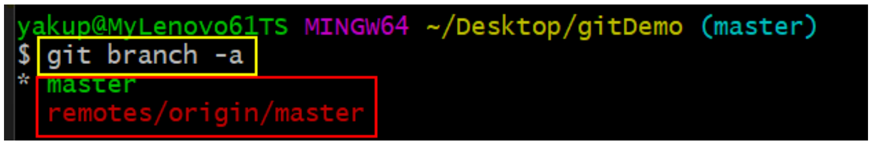
▼ `git branch` : to see all branches

- If we haven't specified any branch name, master would be the default one



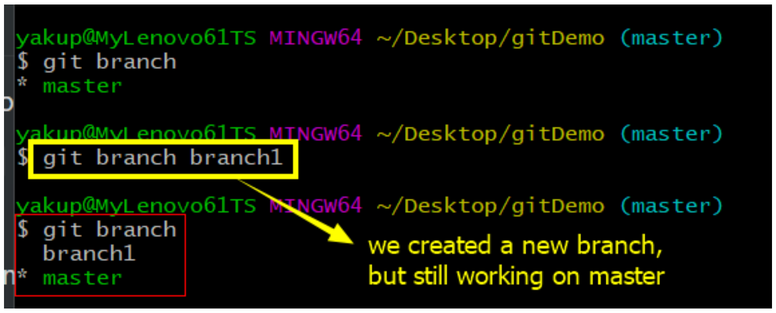
```
MINGW64:/c/Users/yakup/Desktop/gitDemo

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git status
On branch master
nothing to commit, working tree clean

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch
* master

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$
```

▼ `git branch -a` : to view all branches, including remote repo

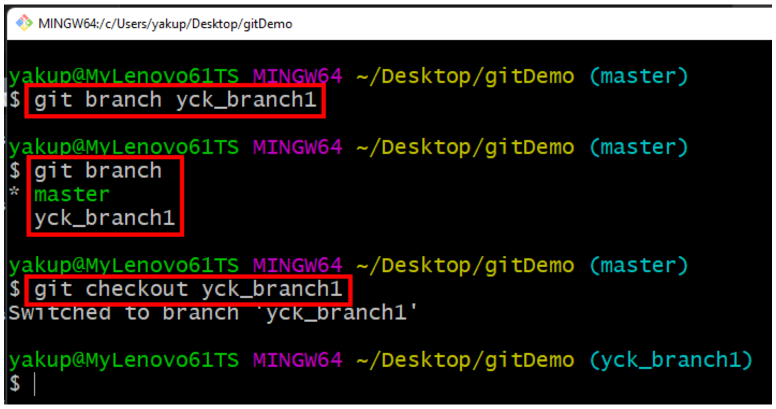


```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch -a
* master
  remotes/origin/master
```

▼ `git branch <branch_name>` : to create a new branch



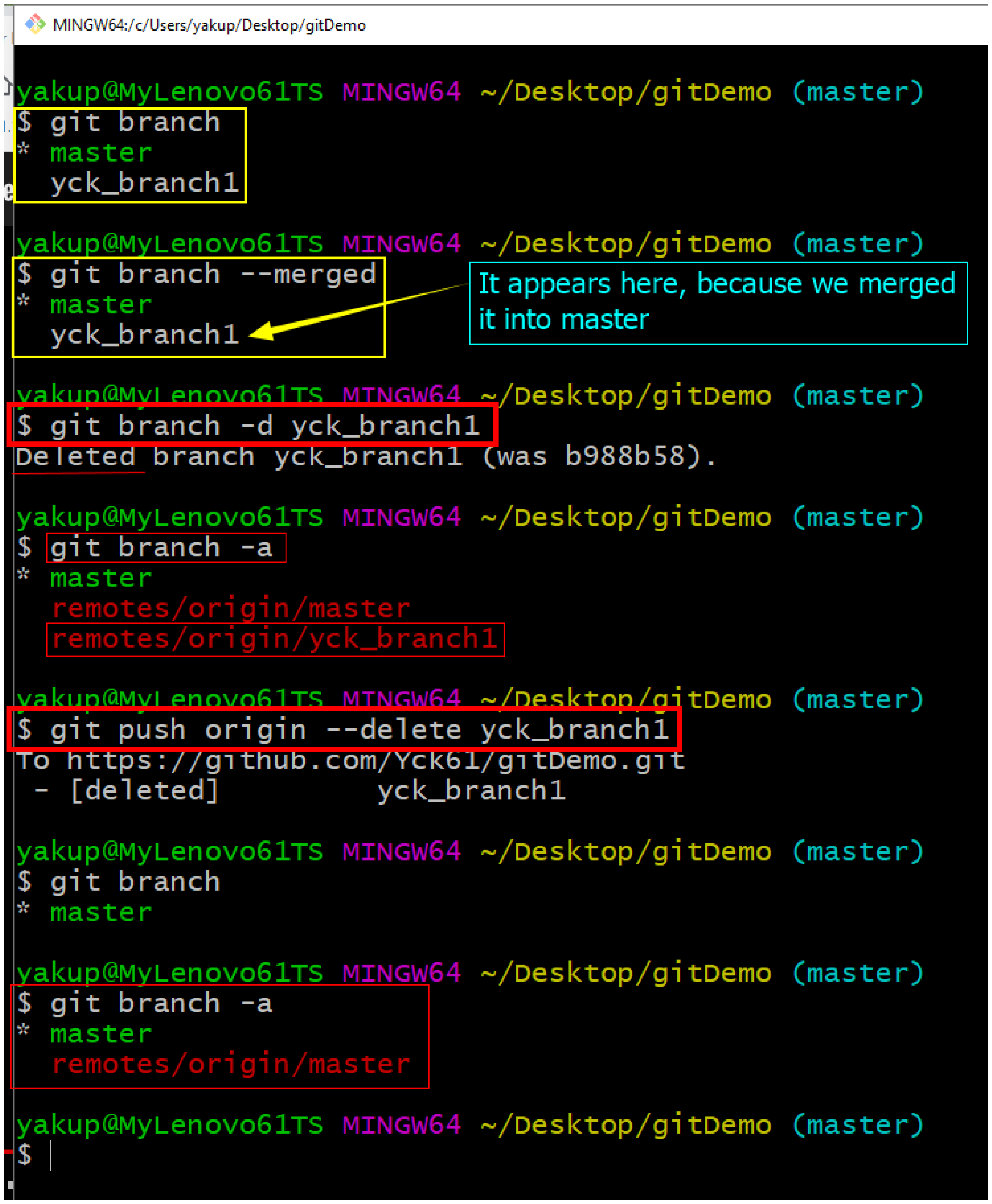
```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch
* master

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch branch1

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch
  branch1
* master
```



```
MINGW64:/c/Users/yakup/Desktop/gitDemo

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch yck_branch1

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch
* master
  yck_branch1

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git checkout yck_branch1
Switched to branch 'yck_branch1'

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (yck_branch1)
$
```

▼ `git branch -d <branch_name>` : to delete a branch locally

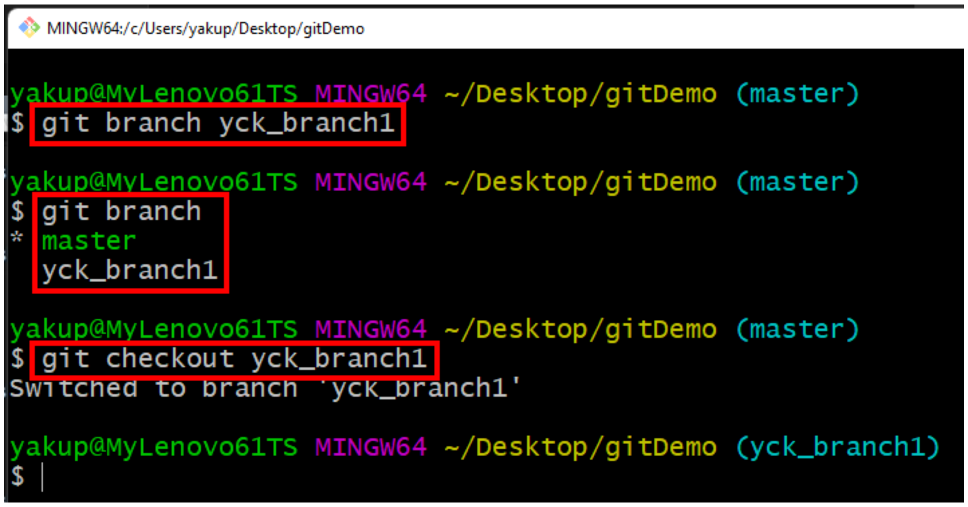
- After merging a branch into master, we can delete it from both local and remote repo



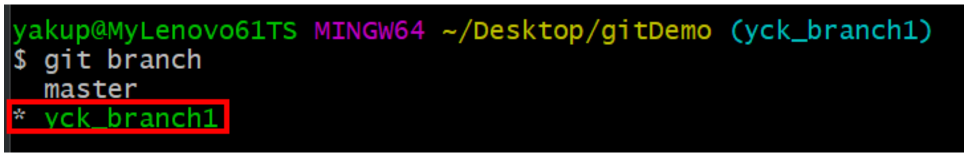
MINGW64:/c/Users/yakup/Desktop/gitDemo

```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch
* master
  yck_branch1

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch --merged
* master
  yck_branch1

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch -d yck_branch1
Deleted branch yck_branch1 (was b988b58).

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch -a
* master
  remotes/origin/master
  remotes/origin/yck_branch1

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git push origin --delete yck_branch1
To https://github.com/Yck61/gitDemo.git
 - [deleted]         yck_branch1

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch
* master

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch -a
* master
  remotes/origin/master

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

▼ `git checkout <branch_name>` : to checkout / switch to another branch



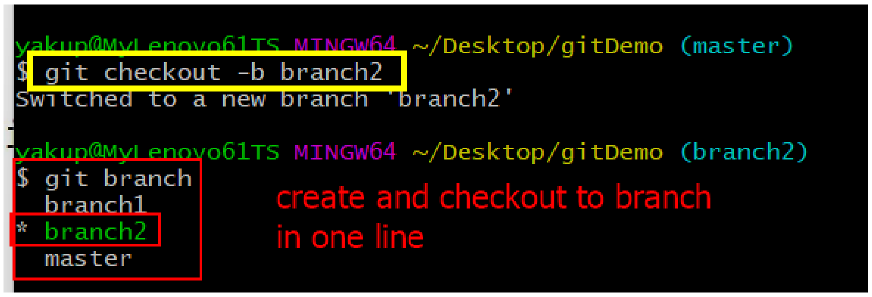
```
MINGW64:/c/Users/yakup/Desktop/gitDemo

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch yck_branch1

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch
* master
  yck_branch1

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git checkout yck_branch1
Switched to branch 'yck_branch1'

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (yck_branch1)
$ |
```



```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (yck_branch1)
$ git branch
  master
* yck_branch1
```

if we run again "git branch" command, we'll see that we've checked out to new branch

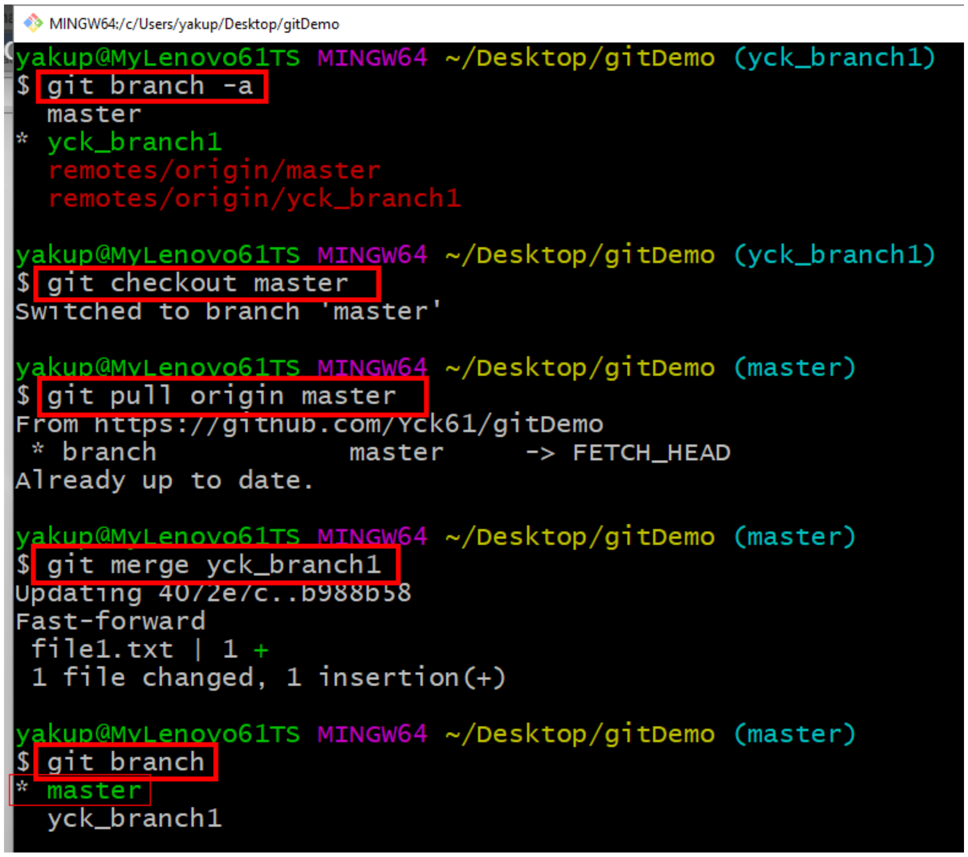
▼ `git checkout -b <branch_name>` : to create and checkout to new branch in one line

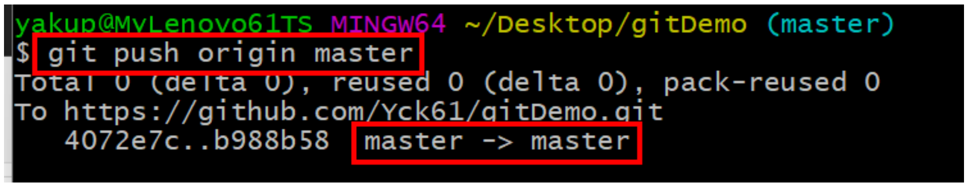


```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git checkout -b branch2
Switched to a new branch 'branch2'

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (branch2)
$ git branch
  branch1
* branch2
  master
```

▼ `git merge <branch_name>` : to merge specified branch into current branch

- How to merge other branches into master?
    - create a new branch and checkout to it
    - make some changes, add and commit them
    - checkout to master branch again
    - pull master from origin to see whether there is something new
    - finally merge other branch into current (here current is master)

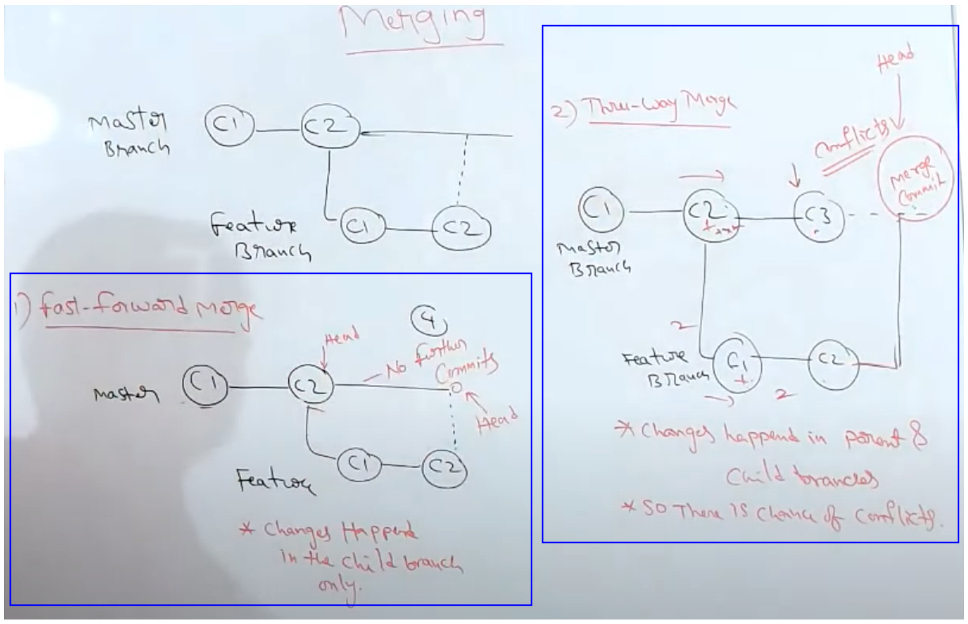

```
MINGW64:/c/Users/yakup/Desktop/gitDemo
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (yck_branch1)
$ git branch -a
  master
* yck_branch1
  remotes/origin/master
  remotes/origin/yck_branch1

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (yck_branch1)
$ git checkout master
Switched to branch 'master'

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git pull origin master
From https://github.com/Yck61/gitDemo
 * branch            master     -> FETCH_HEAD
Already up to date.

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git merge yck_branch1
Updating 4072e7c..b988b58
Fast-forward
 file1.txt | 1 +
 1 file changed, 1 insertion(+)

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch
* master
  yck_branch1
```


```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git push origin master
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Yck61/gitDemo.git
   4072e7c..b988b58  master -> master
```

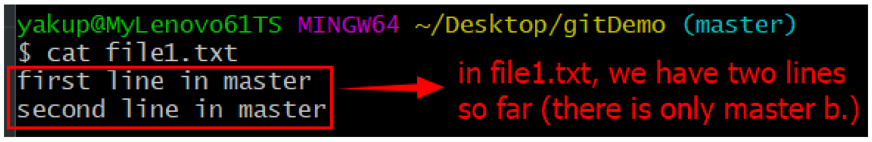
As a last step, push your updated master to origin / remote repo

---

- `git branch --merged`: to see what other branches are merged into current branch
- merging strategies: fast-forward and three ways (ort)

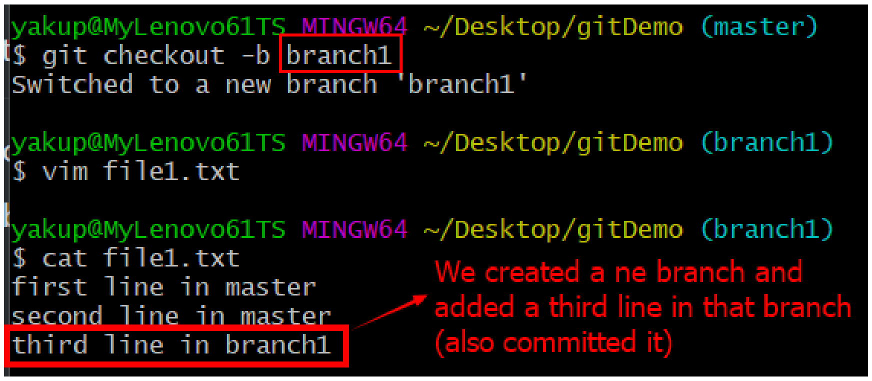


▼ resolving merge conflict:



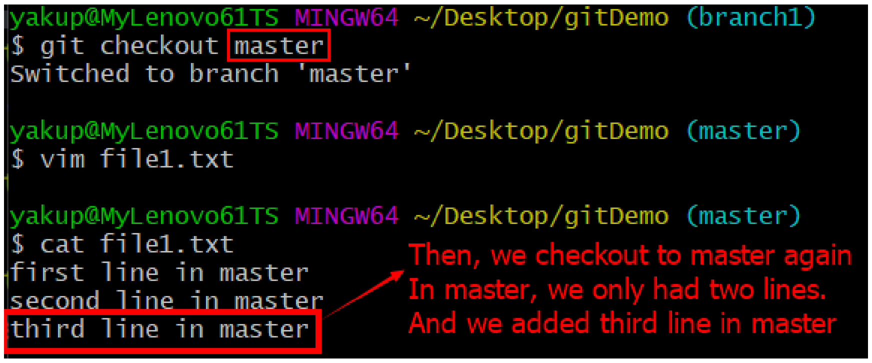
```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ cat file1.txt
first line in master
second line in master
```

in file1.txt, we have two lines so far (there is only master b.)



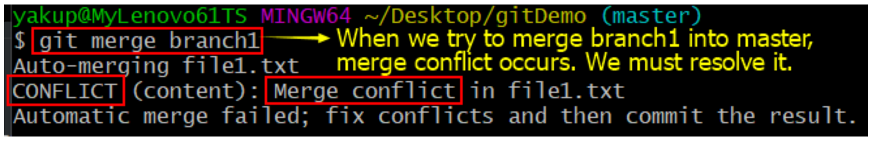
```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git checkout -b branch1
Switched to a new branch 'branch1'

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (branch1)
$ vim file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (branch1)
$ cat file1.txt
first line in master
second line in master
third line in branch1
```

We created a ne branch and added a third line in that branch (also committed it)



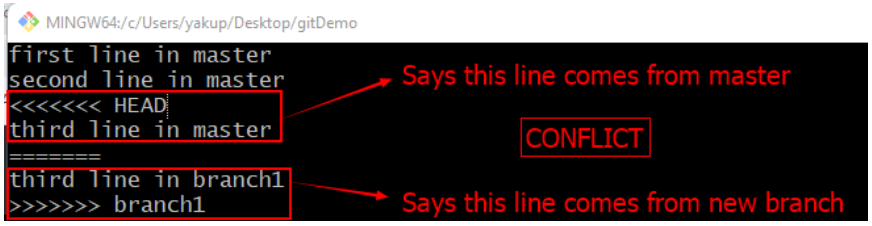
```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (branch1)
$ git checkout master
Switched to branch 'master'

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ vim file1.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ cat file1.txt
first line in master
second line in master
third line in master
```

Then, we checkout to master again
In master, we only had two lines.
And we added third line in master



```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git merge branch1
Auto-merging file1.txt
CONFLICT (content): Merge conflict in file1.txt
Automatic merge failed; fix conflicts and then commit the result.
```

When we try to merge branch1 into master, merge conflict occurs. We must resolve it.



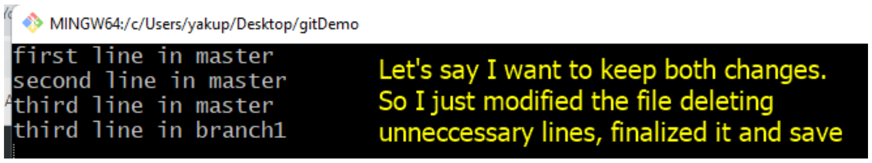
MINGW64:/c/Users/yakup/Desktop/gitDemo

```
first line in master
second line in master
<<<<<<< HEAD
third line in master
=======
third line in branch1
>>>>>>> branch1
```

Says this line comes from master

CONFLICT

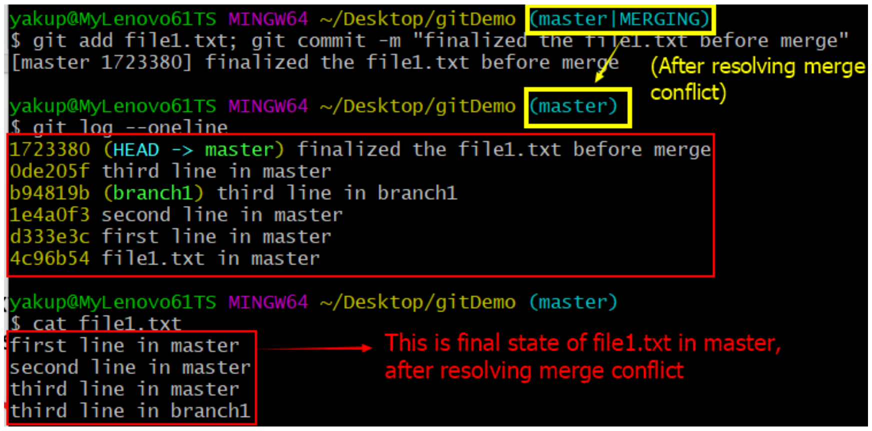
Says this line comes from new branch



MINGW64:/c/Users/yakup/Desktop/gitDemo

```
first line in master
second line in master
third line in master
third line in branch1
```

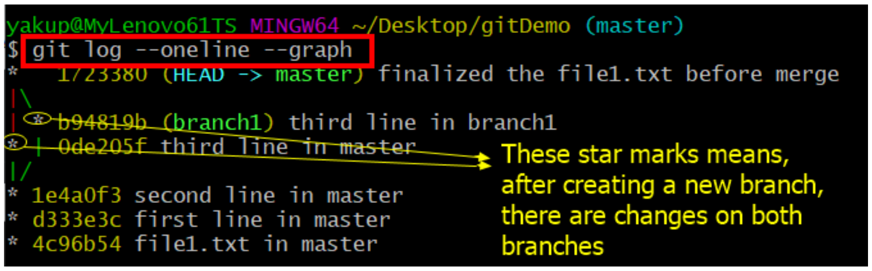
Let's say I want to keep both changes. So I just modified the file deleting unneccessary lines, finalized it and save



```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master|MERGING)
$ git add file1.txt; git commit -m "finalized the file1.txt before merge"
[master 1723380] finalized the file1.txt before merge

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git log --oneline
1723380 (HEAD -> master) finalized the file1.txt before merge
0de205f third line in master
b94819b (branch1) third line in branch1
1e4a0f3 second line in master
d333e3c first line in master
4c96b54 file1.txt in master

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ cat file1.txt
first line in master
second line in master
third line in master
third line in branch1
```

(After resolving merge conflict)

This is final state of file1.txt in master, after resolving merge conflict



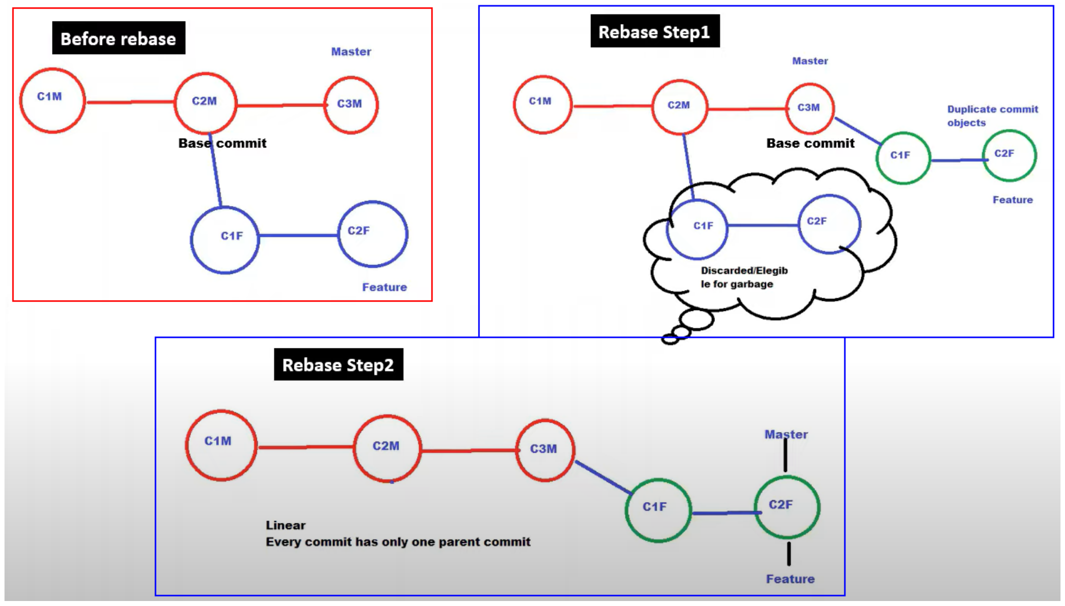
```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git log --oneline --graph
*   1723380 (HEAD -> master) finalized the file1.txt before merge
|\
| * b94819b (branch1) third line in branch1
* | 0de205f third line in master
|/
* 1e4a0f3 second line in master
* d333e3c first line in master
* 4c96b54 file1.txt in master
```

These star marks means, after creating a new branch, there are changes on both branches

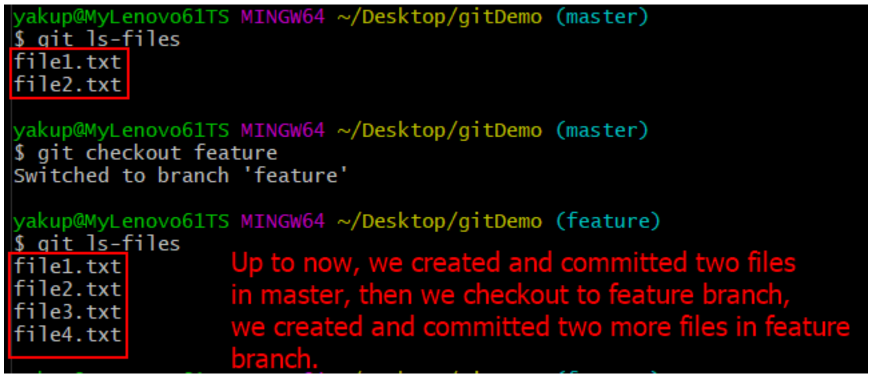
---

▼ rebase

- rebase is like another way of merging
- Rebasing creates one linear graphic in history
- All the commits of new branch will be added on top of master
- Then we need to execute merge command
- The thing is that, after rebase, it'll be really difficult to figure out what happened in what commit
- So we should not go for rebase in remote repo
- We should use it only for our local repo for our local branches
- The advantage of using rebase is, there is no change of getting conflict when merging
- Disadvantage is, it'll be really difficult to figure out what happened in what commit

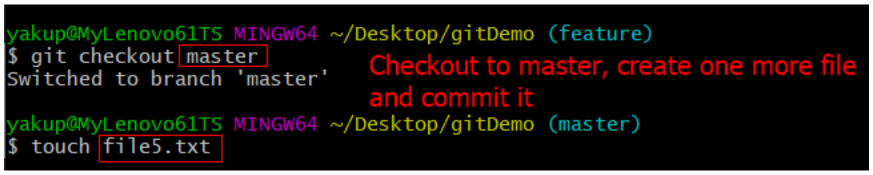


▼ `git rebase <branch_name>` : to rebase current branch based on another branch



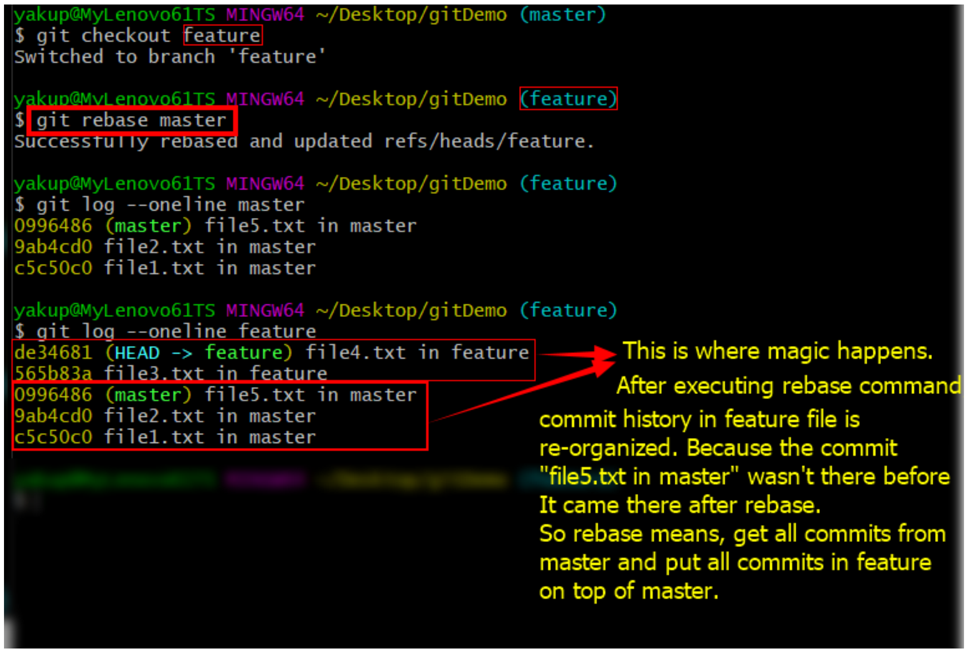
```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git ls-files
file1.txt
file2.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git checkout feature
Switched to branch 'feature'

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (feature)
$ git ls-files
file1.txt
file2.txt
file3.txt
file4.txt
```

Up to now, we created and committed two files in master, then we checkout to feature branch, we created and committed two more files in feature branch.



```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (feature)
$ git checkout master
Switched to branch 'master'

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ touch file5.txt
```

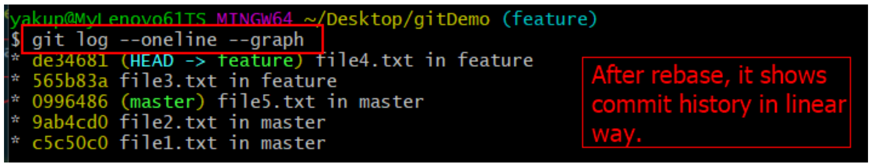
Checkout to master, create one more file and commit it



```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git checkout feature
Switched to branch 'feature'

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (feature)
$ git rebase master
Successfully rebased and updated refs/heads/feature.

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (feature)
$ git log --oneline master
0996486 (master) file5.txt in master
9ab4cd0 file2.txt in master
c5c50c0 file1.txt in master

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (feature)
$ git log --oneline feature
de34681 (HEAD -> feature) file4.txt in feature
565b83a file3.txt in feature
0996486 (master) file5.txt in master
9ab4cd0 file2.txt in master
c5c50c0 file1.txt in master
```

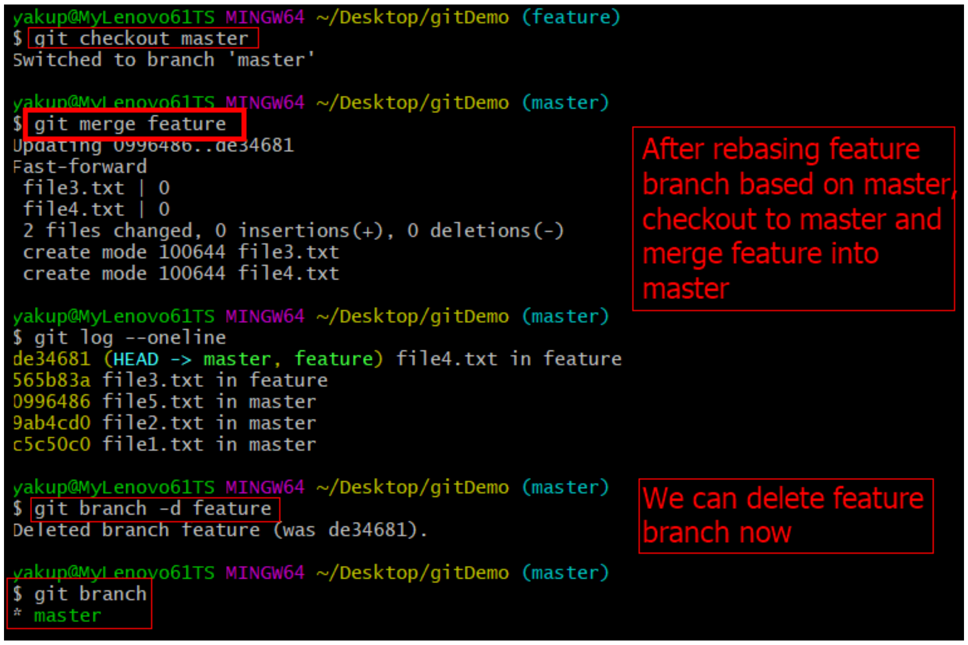
This is where magic happens.
After executing rebase command commit history in feature file is re-organized. Because the commit "file5.txt in master" wasn't there before It came there after rebase.
So rebase means, get all commits from master and put all commits in feature on top of master.

rebase/re-organize feature branch according to master



```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (feature)
$ git log --oneline --graph
* de34681 (HEAD -> feature) file4.txt in feature
* 565b83a file3.txt in feature
* 0996486 (master) file5.txt in master
* 9ab4cd0 file2.txt in master
* c5c50c0 file1.txt in master
```

After rebase, it shows commit history in linear way.



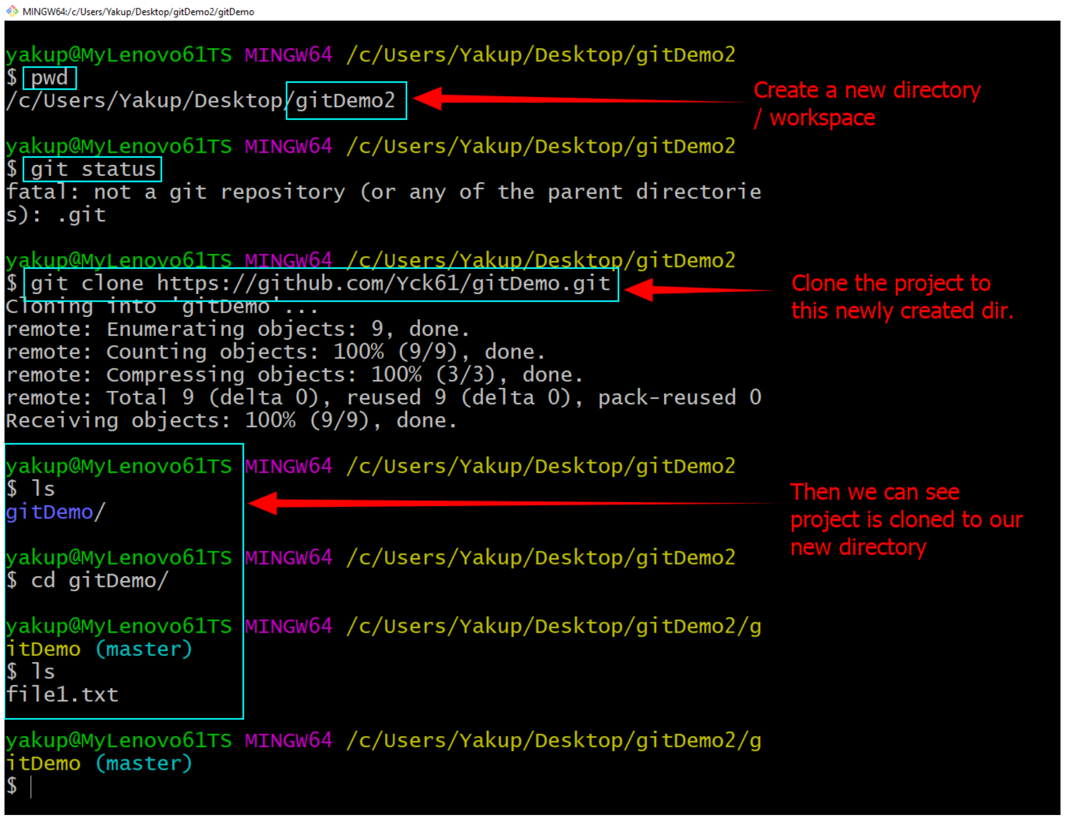
```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (feature)
$ git checkout master
Switched to branch 'master'

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git merge feature
Updating 0996486..de34681
Fast-forward
 file3.txt | 0
 file4.txt | 0
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file3.txt
 create mode 100644 file4.txt

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git log --oneline
de34681 (HEAD -> master, feature) file4.txt in feature
565b83a file3.txt in feature
0996486 file5.txt in master
9ab4cd0 file2.txt in master
c5c50c0 file1.txt in master

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch -d feature
Deleted branch feature (was de34681).

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch
* master
```

After rebasing feature branch based on master, checkout to master and merge feature into master

We can delete feature branch now

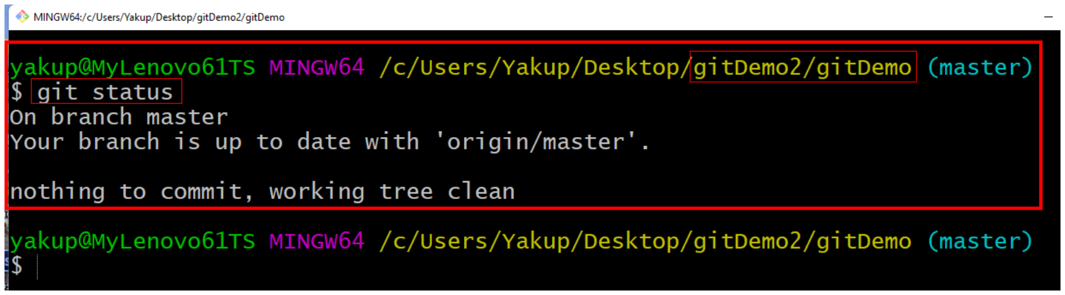
▼ `git clone <URL>` : Clone the repo from remote to your local system

- Create a new directory and clone the project here from the remote repo



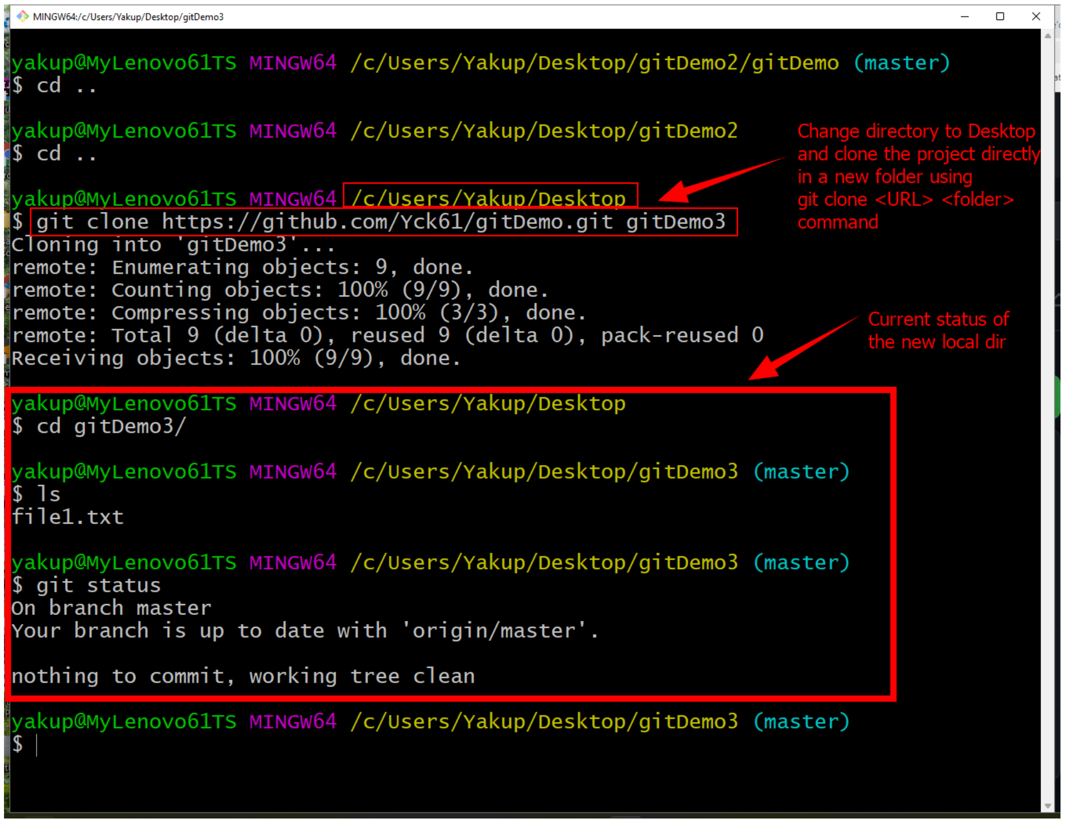
```
MINGW64:/c/Users/Yakup/Desktop/gitDemo2/gitDemo

yakup@MyLenovo61TS MINGW64 /c/Users/Yakup/Desktop/gitDemo2
$ pwd
/c/Users/Yakup/Desktop/gitDemo2

yakup@MyLenovo61TS MINGW64 /c/Users/Yakup/Desktop/gitDemo2
$ git status
fatal: not a git repository (or any of the parent directorie
s): .git

yakup@MyLenovo61TS MINGW64 /c/Users/Yakup/Desktop/gitDemo2
$ git clone https://github.com/Yck61/gitDemo.git
Cloning into 'gitDemo'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 9 (delta 0), reused 9 (delta 0), pack-reused 0
Receiving objects: 100% (9/9), done.

yakup@MyLenovo61TS MINGW64 /c/Users/Yakup/Desktop/gitDemo2
$ ls
gitDemo/

yakup@MyLenovo61TS MINGW64 /c/Users/Yakup/Desktop/gitDemo2
$ cd gitDemo/

yakup@MyLenovo61TS MINGW64 /c/Users/Yakup/Desktop/gitDemo2/g
itDemo (master)
$ ls
file1.txt

yakup@MyLenovo61TS MINGW64 /c/Users/Yakup/Desktop/gitDemo2/g
itDemo (master)
$
```

Create a new directory / workspace
Clone the project to this newly created dir.
Then we can see project is cloned to our new directory



```
MINGW64:/c/Users/Yakup/Desktop/gitDemo2/gitDemo

yakup@MyLenovo61TS MINGW64 /c/Users/Yakup/Desktop/gitDemo2/gitDemo (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

yakup@MyLenovo61TS MINGW64 /c/Users/Yakup/Desktop/gitDemo2/gitDemo (master)
$
```

▼ `git clone <URL> <target_folder_name>` : To clone a repo to desired folder directly


MINGW64:/c/Users/Yakup/Desktop/gitDemo3
Change directory to Desktop and clone the project directly in a new folder using git clone <URL> <folder> command
Current status of the new local dir

```
yakup@MyLenovo61TS MINGW64 /c/Users/Yakup/Desktop/gitDemo2/gitDemo (master)
$ cd ..

yakup@MyLenovo61TS MINGW64 /c/Users/Yakup/Desktop/gitDemo2
$ cd ..

yakup@MyLenovo61TS MINGW64 /c/Users/Yakup/Desktop
$ git clone https://github.com/Yck61/gitDemo.git gitDemo3
Cloning into 'gitDemo3'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 9 (delta 0), reused 9 (delta 0), pack-reused 0
Receiving objects: 100% (9/9), done.

yakup@MyLenovo61TS MINGW64 /c/Users/Yakup/Desktop
$ cd gitDemo3/

yakup@MyLenovo61TS MINGW64 /c/Users/Yakup/Desktop/gitDemo3 (master)
$ ls
file1.txt

yakup@MyLenovo61TS MINGW64 /c/Users/Yakup/Desktop/gitDemo3 (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean

yakup@MyLenovo61TS MINGW64 /c/Users/Yakup/Desktop/gitDemo3 (master)
$
```

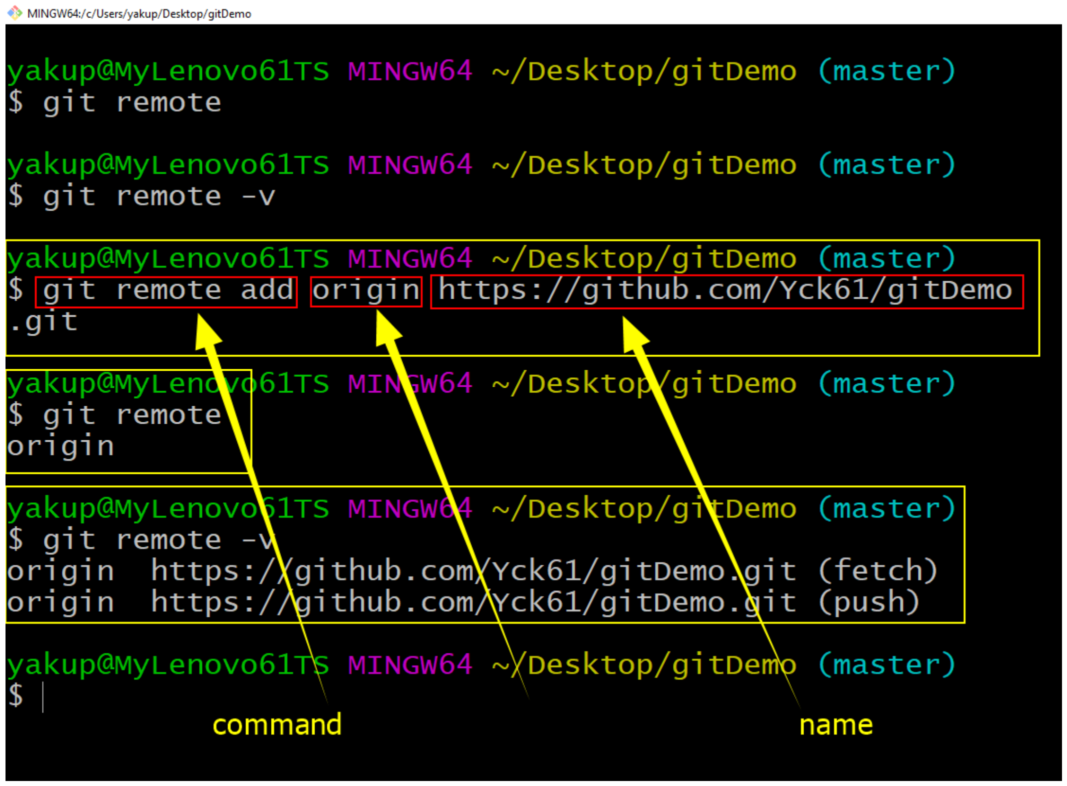
▼ **`git remote`** or **`git remote -v`** : to see linked remote repos

- Only difference is that `git remote -v` also shows URL of the remote repo

---

▼ **`git remote add <name> <URL>`** : to add new remote repo and give it a name

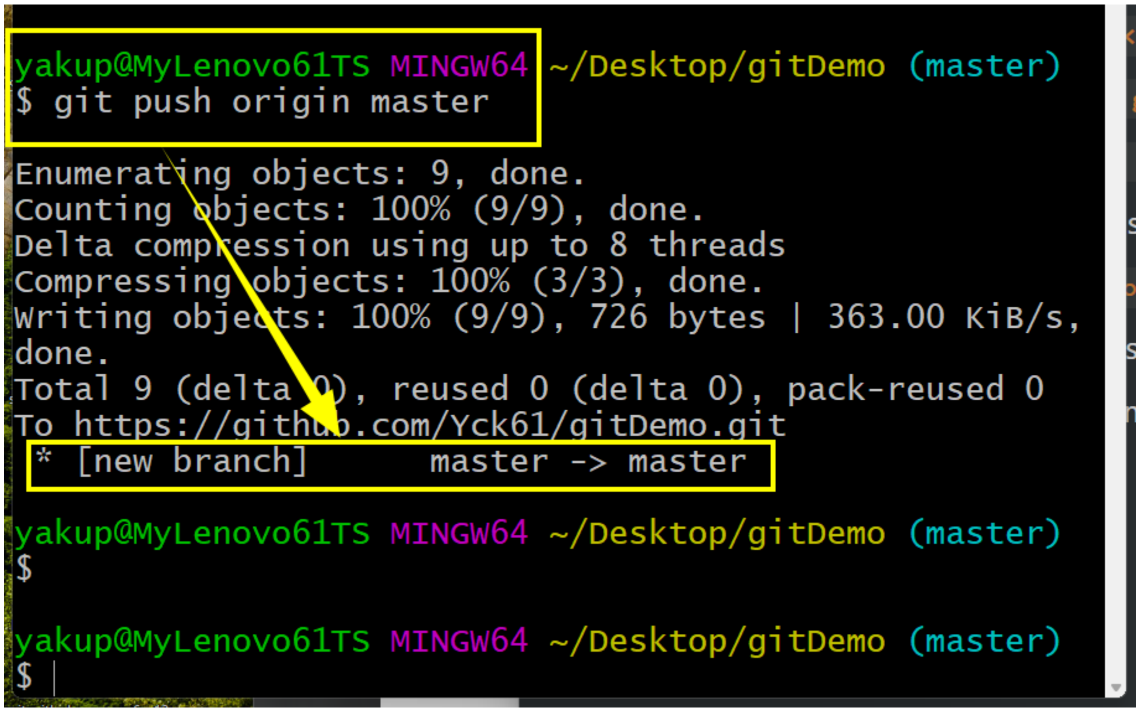
- To connect /upload our local repo to a remote repo
- Create a remote repo in Github and copy its URL
- Then add your local repo to this remote repo using `git remote add <name> <URL>` command



```
MINGW64:/c/Users/yakup/Desktop/gitDemo

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git remote

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git remote -v

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git remote add origin https://github.com/Yck61/gitDemo
.git

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git remote
origin

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git remote -v
origin  https://github.com/Yck61/gitDemo.git (fetch)
origin  https://github.com/Yck61/gitDemo.git (push)

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

▼ `git push <URL_or_name> <branch_name>` : to push a commit to remote repo

- If we do this for the first time, it will require authentication
- After implementing necessary steps, we will see this:



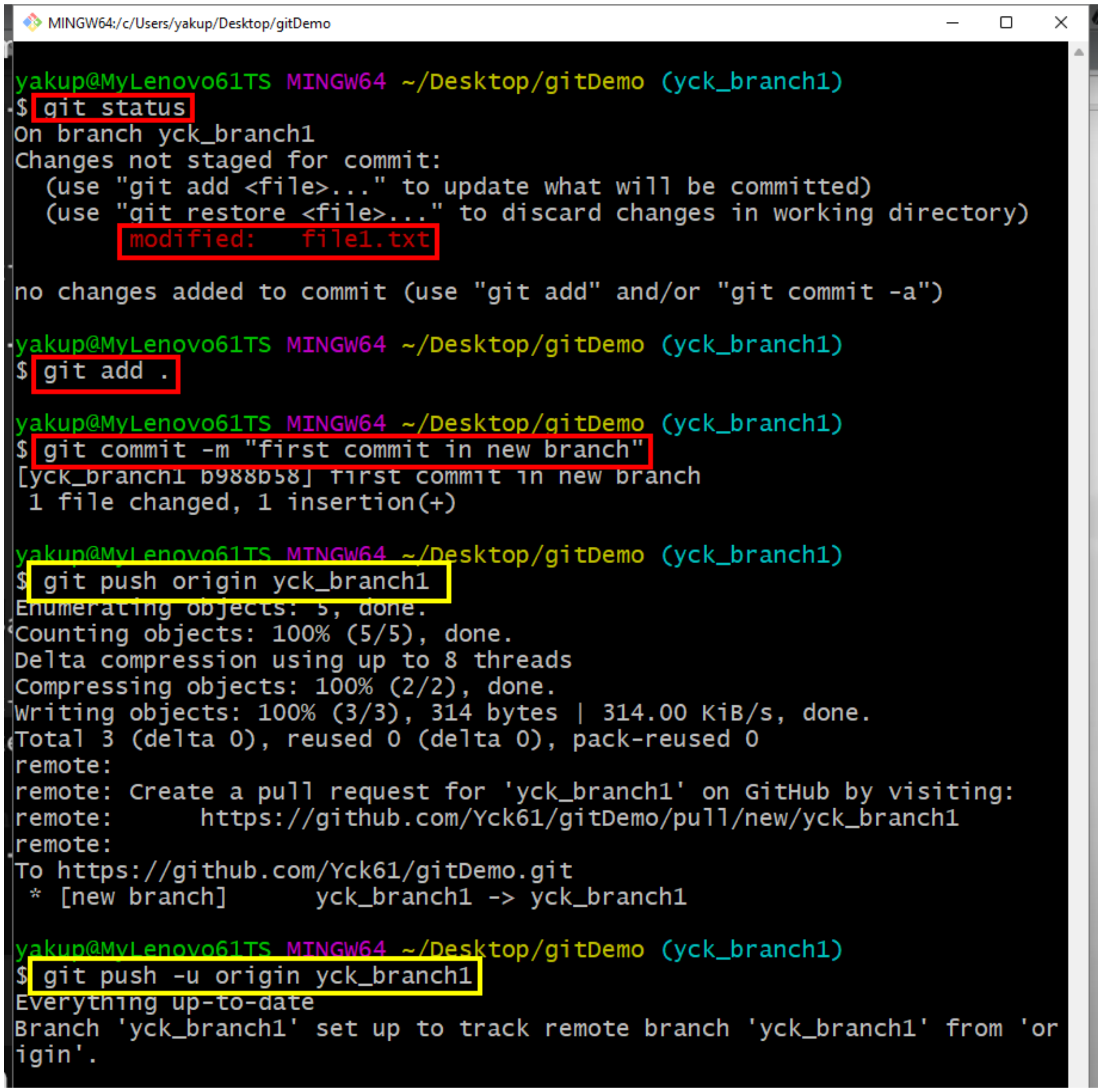
```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git push origin master
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (9/9), 726 bytes | 363.00 KiB/s,
done.
Total 9 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Yck61/gitDemo.git
 * [new branch]      master -> master

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ |
```

▼ `git pull <URL_or_name> <branch_name>` : to pull changes from remote repo

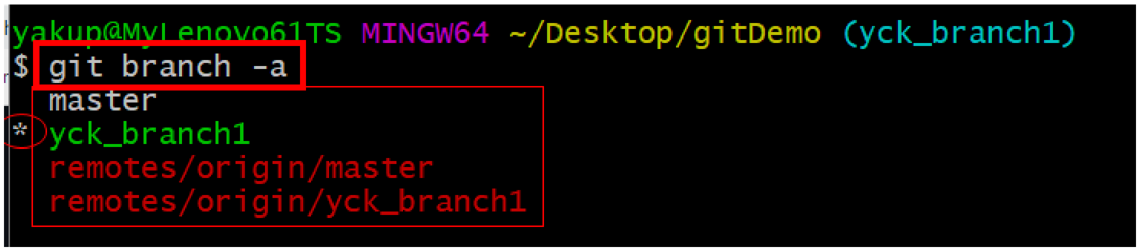
-

▼ `git push <URL_or_name> <new_branch_name>` : to push new branch to remote

- check out to new branch
- modify an existing file
- add it, commit and push



```
MINGW64:/c/Users/yakup/Desktop/gitDemo

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (yck_branch1)
$ git status
On branch yck_branch1
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt

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

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (yck_branch1)
$ git add .

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (yck_branch1)
$ git commit -m "first commit in new branch"
[yck_branch1 b988b58] first commit in new branch
 1 file changed, 1 insertion(+)

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (yck_branch1)
$ git push origin yck_branch1
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 314 bytes | 314.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'yck_branch1' on GitHub by visiting:
remote:      https://github.com/Yck61/gitDemo/pull/new/yck_branch1
remote:
To https://github.com/Yck61/gitDemo.git
 * [new branch]      yck_branch1 -> yck_branch1

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (yck_branch1)
$ git push -u origin yck_branch1
Everything up-to-date
Branch 'yck_branch1' set up to track remote branch 'yck_branch1' from 'or
igin'.
```

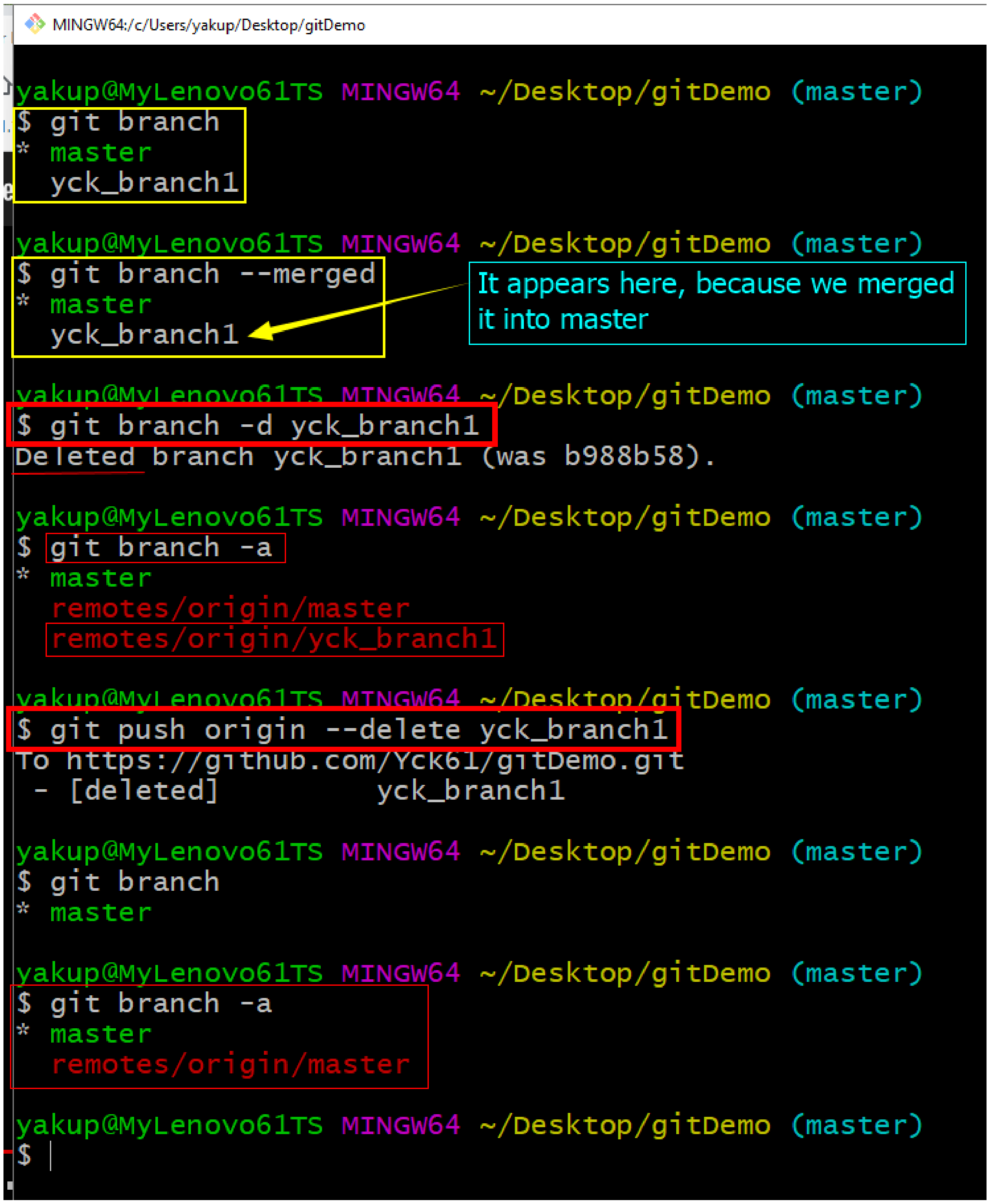


```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (yck_branch1)
$ git branch -a
  master
* yck_branch1
  remotes/origin/master
  remotes/origin/yck_branch1
```

git branch -a to see all present branches

▼ `git push origin --delete <branch_name>` : to delete a remote branch

- After merging a branch into master, we can delete it from both local and remote repo


MINGW64:/c/Users/yakup/Desktop/gitDemo
It appears here, because we merged it into master

```
yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch
* master
  yck_branch1

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch --merged
* master
  yck_branch1

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch -d yck_branch1
Deleted branch yck_branch1 (was b988b58).

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch -a
* master
  remotes/origin/master
  remotes/origin/yck_branch1

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git push origin --delete yck_branch1
To https://github.com/Yck61/gitDemo.git
 - [deleted]         yck_branch1

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch
* master

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ git branch -a
* master
  remotes/origin/master

yakup@MyLenovo61TS MINGW64 ~/Desktop/gitDemo (master)
$ 
```

▼ Practice

1. Create a new project
2. Enable VCS
3. Create .gitignore file and add:
    a. .idea folder
    b. target folder
    c. *.iml file
4. Create a package and a java class inside
5. Commit the changes
6. Share the project on Github (and share URL)
    a. Add others as a collaborator (your Github username is needed)
    b. Others, check your email and accept the invitation
7. Download the project
8. Create a local branch from master

> 💡 We never-ever work directly on master branch!
> Create a local branch from master
> And always work on your own branch
> Do not do anything on master branch!

9. Create a package with your name (under gitdemo package) and a java class inside
10. Do some changes and commit (at least three commits)

> 💡 ALWAYS commit your changes!

11. Push your changes to remote repo
12. Create a pull request (either on IntelliJ or Github)
13. Merge pull request (either on IntelliJ or Github)
14. Go back to IntelliJ and Checkout to master and update the project
15. Delete the local branches (both on IntelliJ and remote repo)
16. Create another local branch from master and re-start the cycle