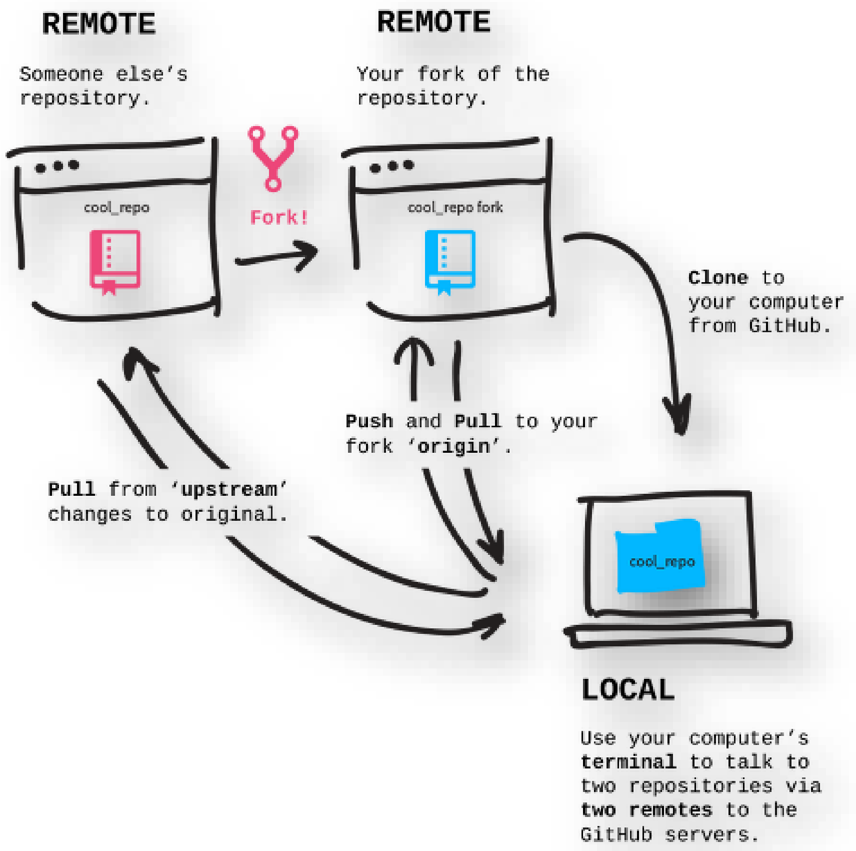
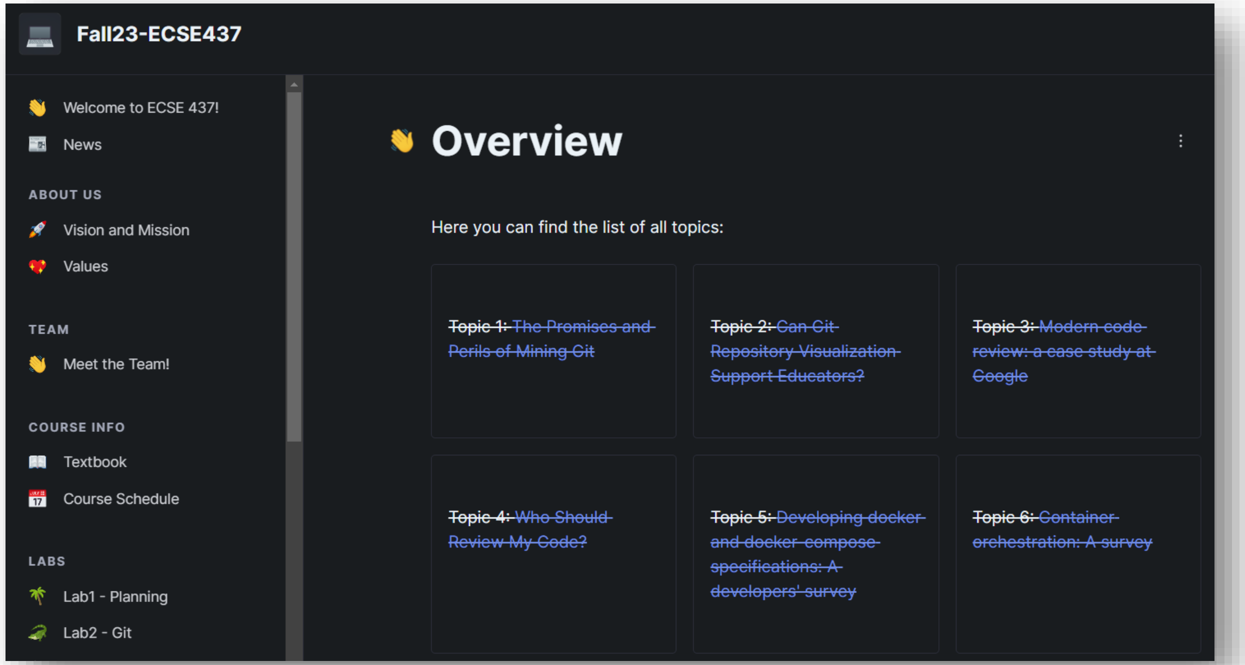
Basic git concepts #2

Majid Babaei

How to contribute to presentations via gitBook?

https://github.com/drmajidbabaei/ECSE437


REMOTE
Someone else's repository.
cool_repo
Fork!
REMOTE
Your fork of the repository.
cool_repo fork
Clone to your computer from GitHub.
Push and Pull to your fork 'origin'.
Pull from 'upstream' changes to original.
cool_repo
LOCAL
Use your computer's terminal to talk to two repositories via two remotes to the GitHub servers.



Overview

Here you can find the list of all topics:

~~Topic 1: The Promises and Perils of Mining Git~~

~~Topic 2: Can Git Repository Visualization Support Educators?~~

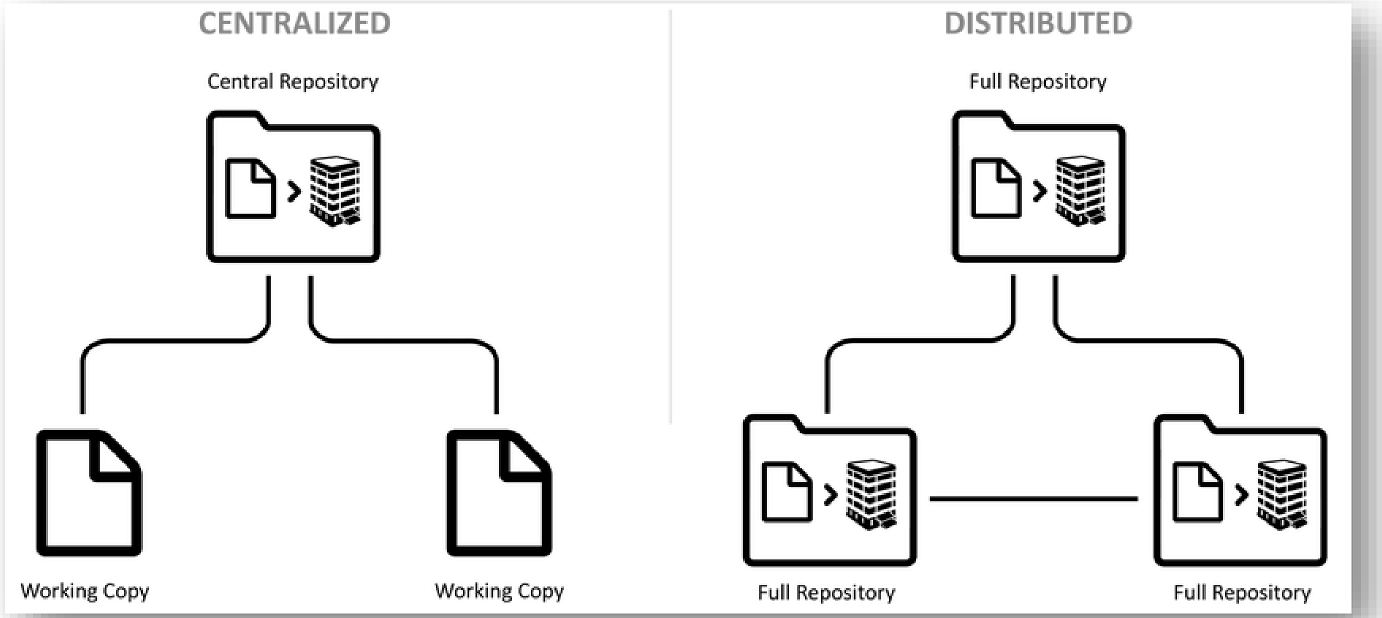
~~Topic 3: Modern code review: a case study at Google~~

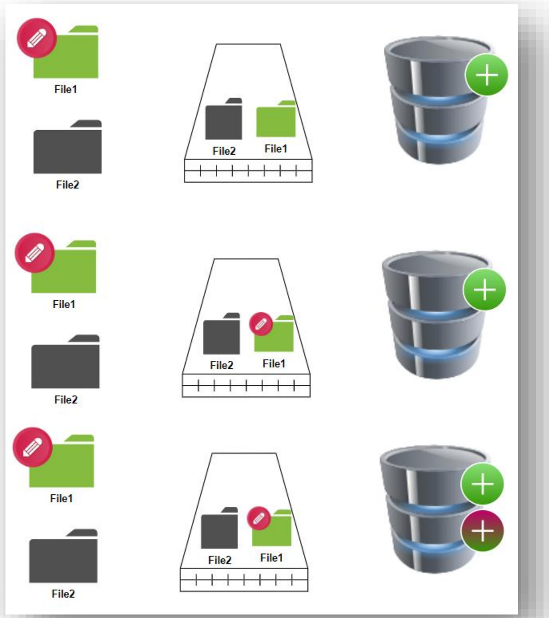
~~Topic 4: Who Should Review My Code?~~

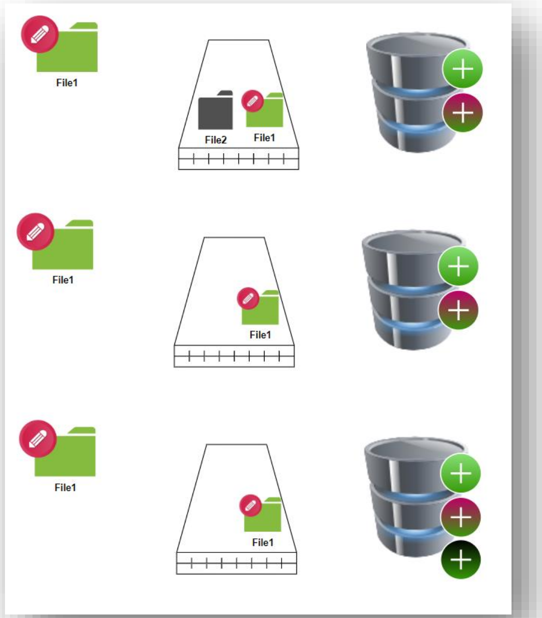
~~Topic 5: Developing docker and docker-compose specifications: A developers' survey~~

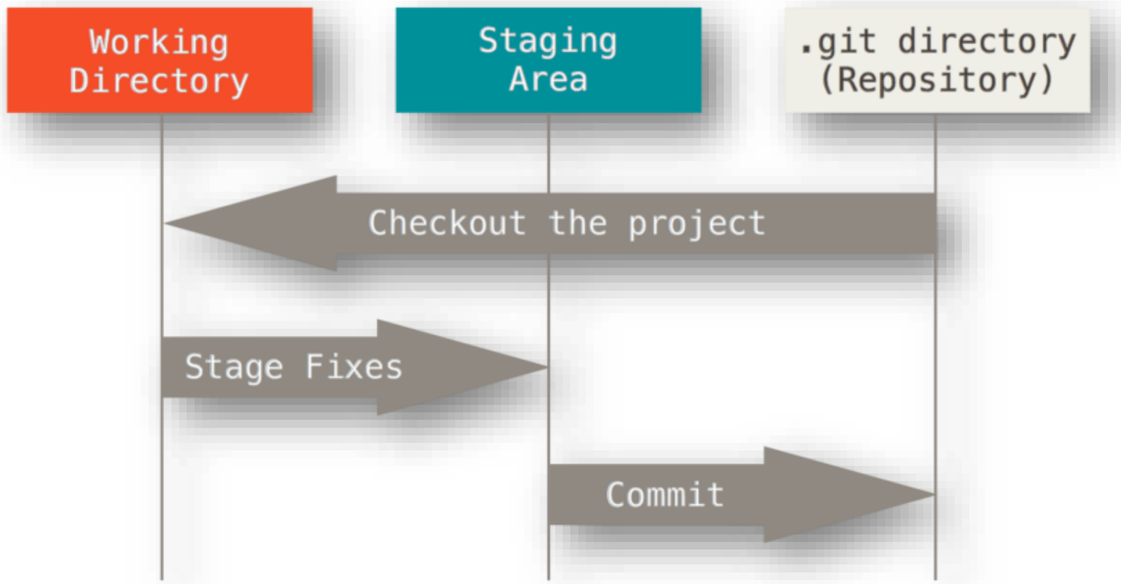
~~Topic 6: Container orchestration: A survey~~

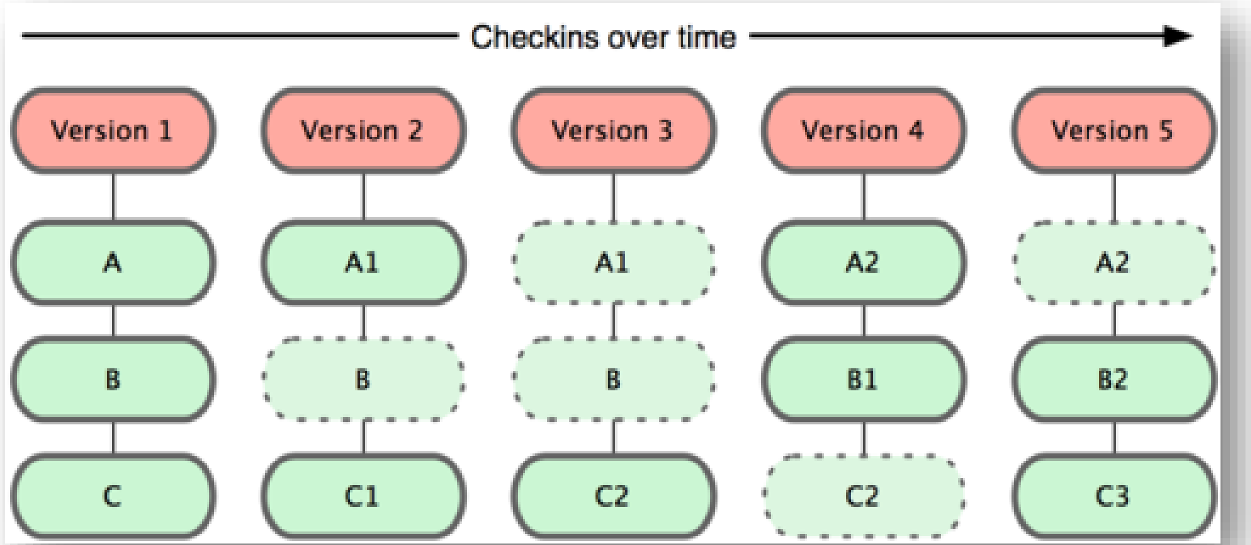
REVIEW

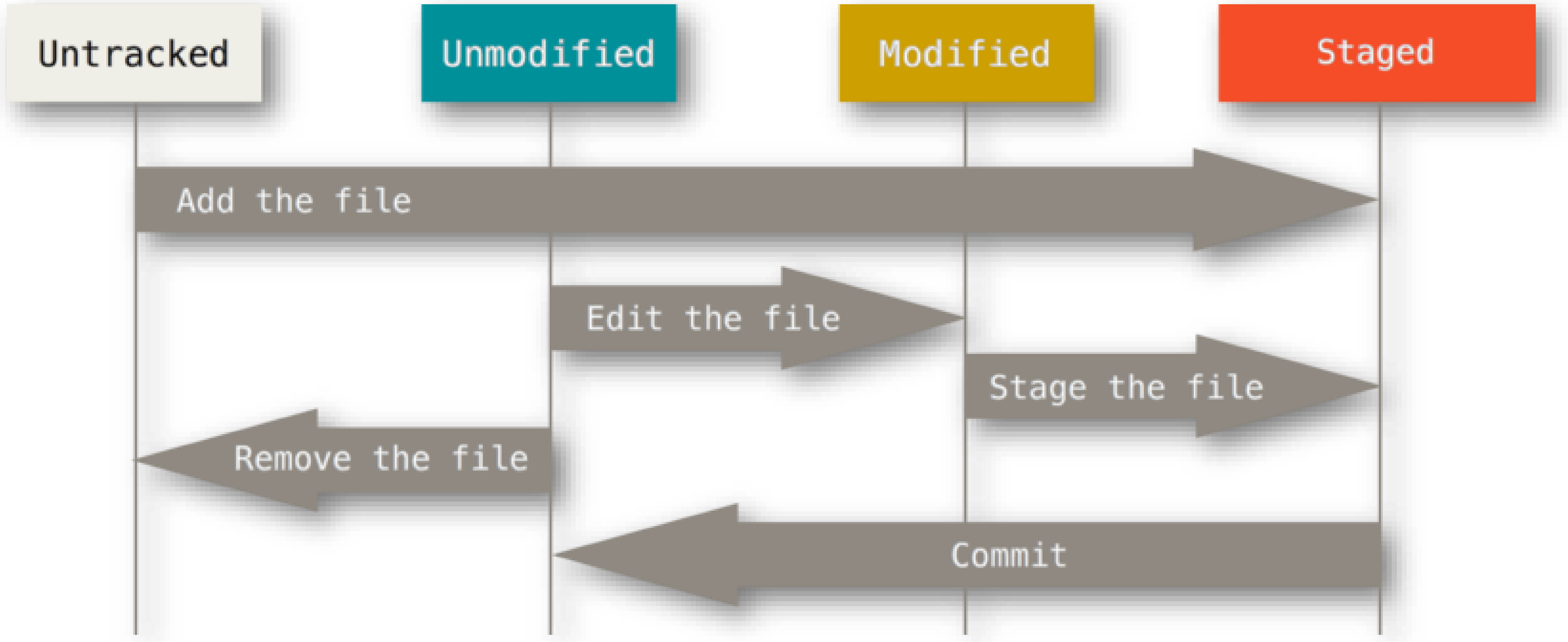

CENTRALIZED
Central Repository
Working Copy
Working Copy
DISTRIBUTED
Full Repository
Full Repository
Full Repository


File1
File2
File2
File1
File1
File2
File2
File1
File1
File2
File2
File1


File1
File2
File1
File1
File1
File1
File1


Working Directory
Staging Area
.git directory (Repository)
Checkout the project
Stage Fixes
Commit


Checkins over time
Version 1
Version 2
Version 3
Version 4
Version 5
A
A1
A1
A2
A2
B
B
B
B1
B2
C
C1
C2
C2
C3


Untracked
Unmodified
Modified
Staged
Add the file
Edit the file
Stage the file
Remove the file
Commit

#1: How can we skip the staging area?

How to skip the Staging Area

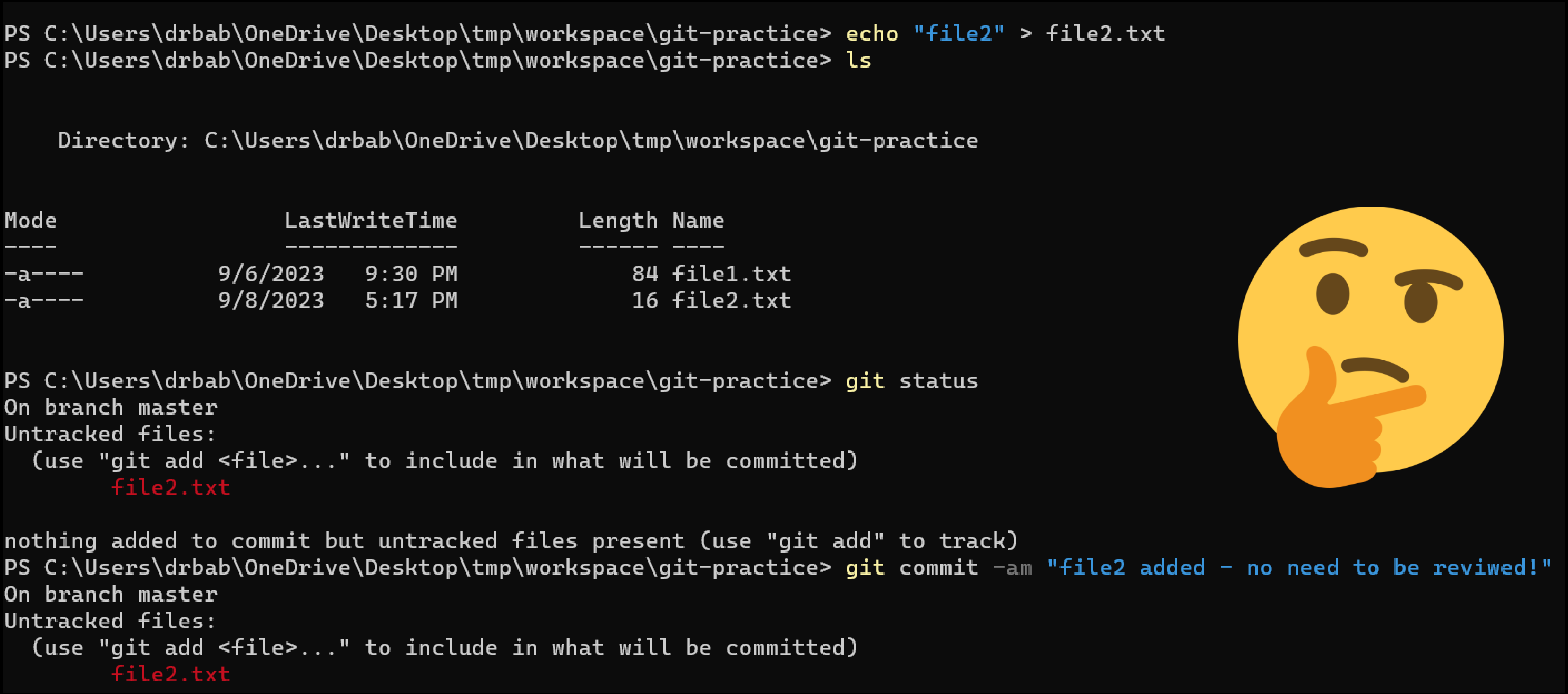
Make sure you check the history before making a new commit

Do it only if you know that your code doesn’t need to be reviewed!

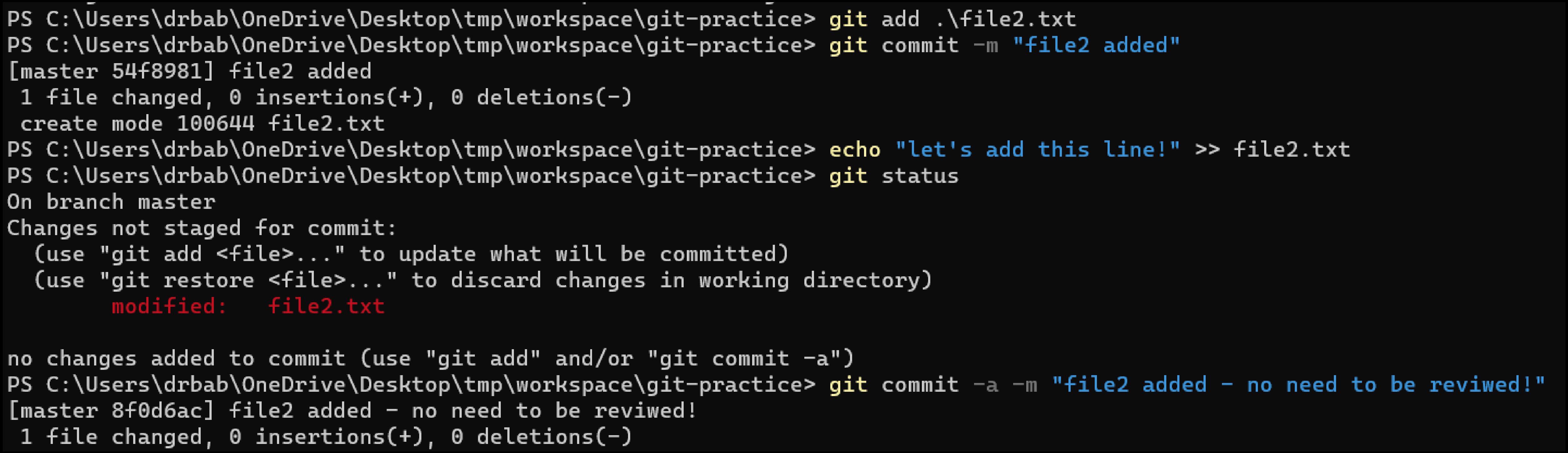
- *git commit –a – m “short message”*

OR

- *git commit –am “short message”*



```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> echo "file2" > file2.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> ls


    Directory: C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----          9/6/2023   9:30 PM             84 file1.txt
-a----          9/8/2023   5:17 PM             16 file2.txt


PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file2.txt

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git commit -am "file2 added - no need to be reviwed!"
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file2.txt
```


```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git add .\file2.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git commit -m "file2 added"
[master 54f8981] file2 added
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file2.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> echo "let's add this line!" >> file2.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> 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:   file2.txt

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git commit -a -m "file2 added - no need to be reviwed!"
[master 8f0d6ac] file2 added - no need to be reviwed!
 1 file changed, 0 insertions(+), 0 deletions(-)
```

#2: How can we remove a file from WD and SA?

A common question when getting started with Git is "How do I tell Git not to track a file (or files) anymore”?

Removing a file both from WD and SA

We need to get the list of the file in the staging area use: git ls-files

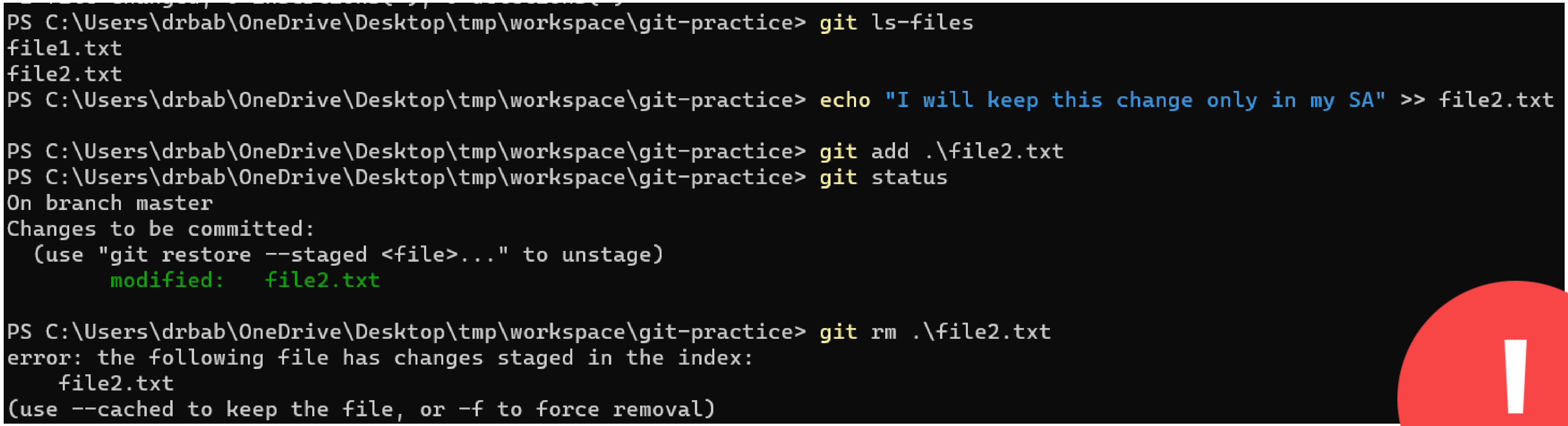
- Normally this process requires two step:
  - Removing the file from the working directory: *rm file1.txt*
  - Removing the file from the staging area: *git add file1.txt*

Is there any faster way?

Removing a file both from WD and SA

- The faster way is to use *git rm file1.txt*

what if I didn't have any copies of this in my repo?



```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git ls-files
file1.txt
file2.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> echo "I will keep this change only in my SA" >> file2.txt

PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git add .\file2.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   file2.txt

PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git rm .\file2.txt
error: the following file has changes staged in the index:
    file2.txt
(use --cached to keep the file, or -f to force removal)
```

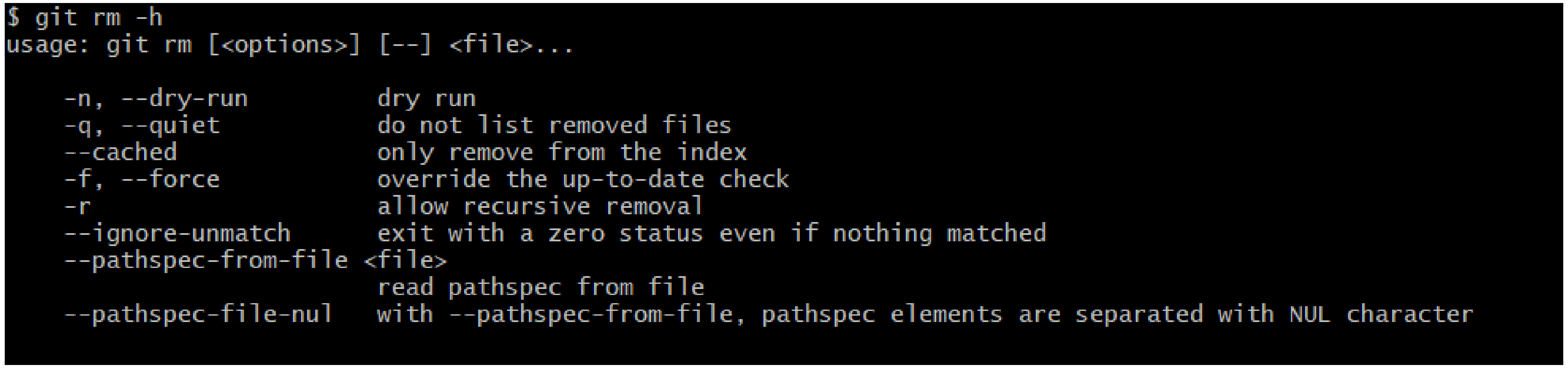
- If there is a discrepancy between the HEAD version of a file and the staging area or working tree version, Git will block the removal.
- This block is a safety mechanism to prevent removal of in-progress changes.

Removing a file both from WD and SA

- What if you only wanted to remove a **file** from the staging area but not from the working directory?

I really don’t know!

- How can I get a quick help from Git?!
  - *git rm –h*

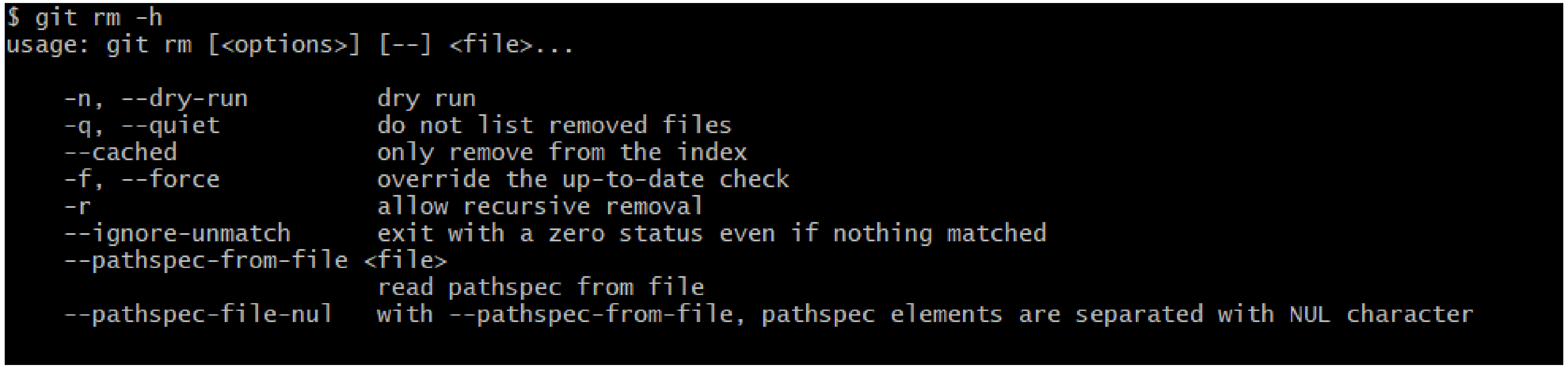


```
$ git rm -h
usage: git rm [<options>] [--] <file>...

    -n, --dry-run         dry run
    -q, --quiet           do not list removed files
    --cached              only remove from the index
    -f, --force           override the up-to-date check
    -r                    allow recursive removal
    --ignore-unmatch      exit with a zero status even if nothing matched
    --pathspec-from-file <file>
                          read pathspec from file
    --pathspec-file-nul   with --pathspec-from-file, pathspec elements are separated with NUL character
```

Removing a file both from WD and SA

- What if you only wanted to remove a **file** from the staging area but not from the working directory?
  - *git rm --cached file1.txt*

- What if you only wanted to remove a **directory** from the staging area but not from the working directory?
  - *git rm --cached –r dev/*

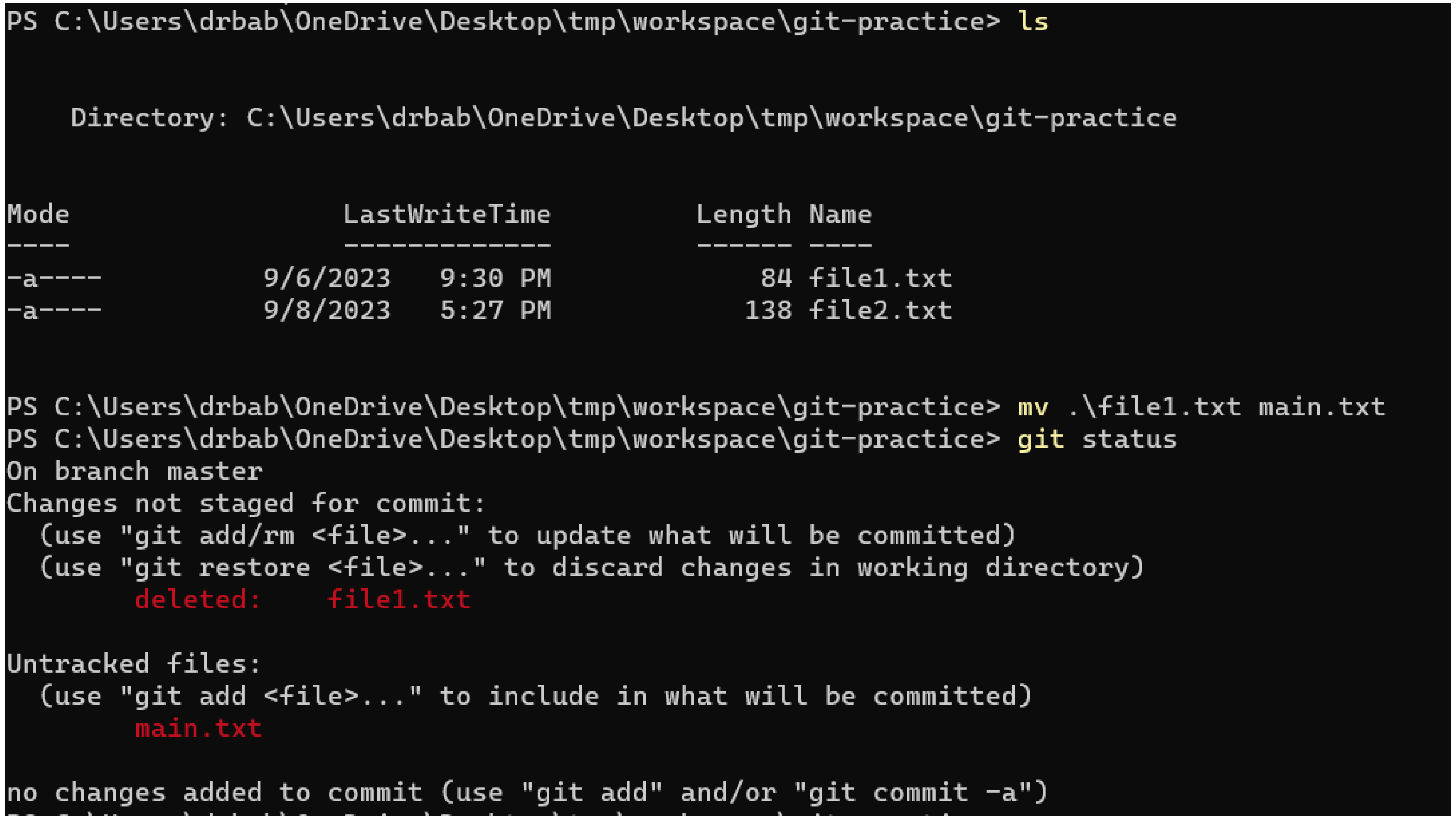


```
$ git rm -h
usage: git rm [<options>] [--] <file>...

    -n, --dry-run         dry run
    -q, --quiet           do not list removed files
    --cached              only remove from the index
    -f, --force           override the up-to-date check
    -r                    allow recursive removal
    --ignore-unmatch      exit with a zero status even if nothing matched
    --pathspec-from-file <file>
                          read pathspec from file
    --pathspec-file-nul   with --pathspec-from-file, pathspec elements are separated with NUL character
```

Question!

What happens if you renamed a file that is tracked by git?

- Suppose you have *file1.txt* and *file2.txt*
- Then you just ran *mv file1.txt main.txt*



```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> ls


    Directory: C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----          9/6/2023   9:30 PM             84 file1.txt
-a----          9/8/2023   5:27 PM            138 file2.txt


PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> mv .\file1.txt main.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    file1.txt

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

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

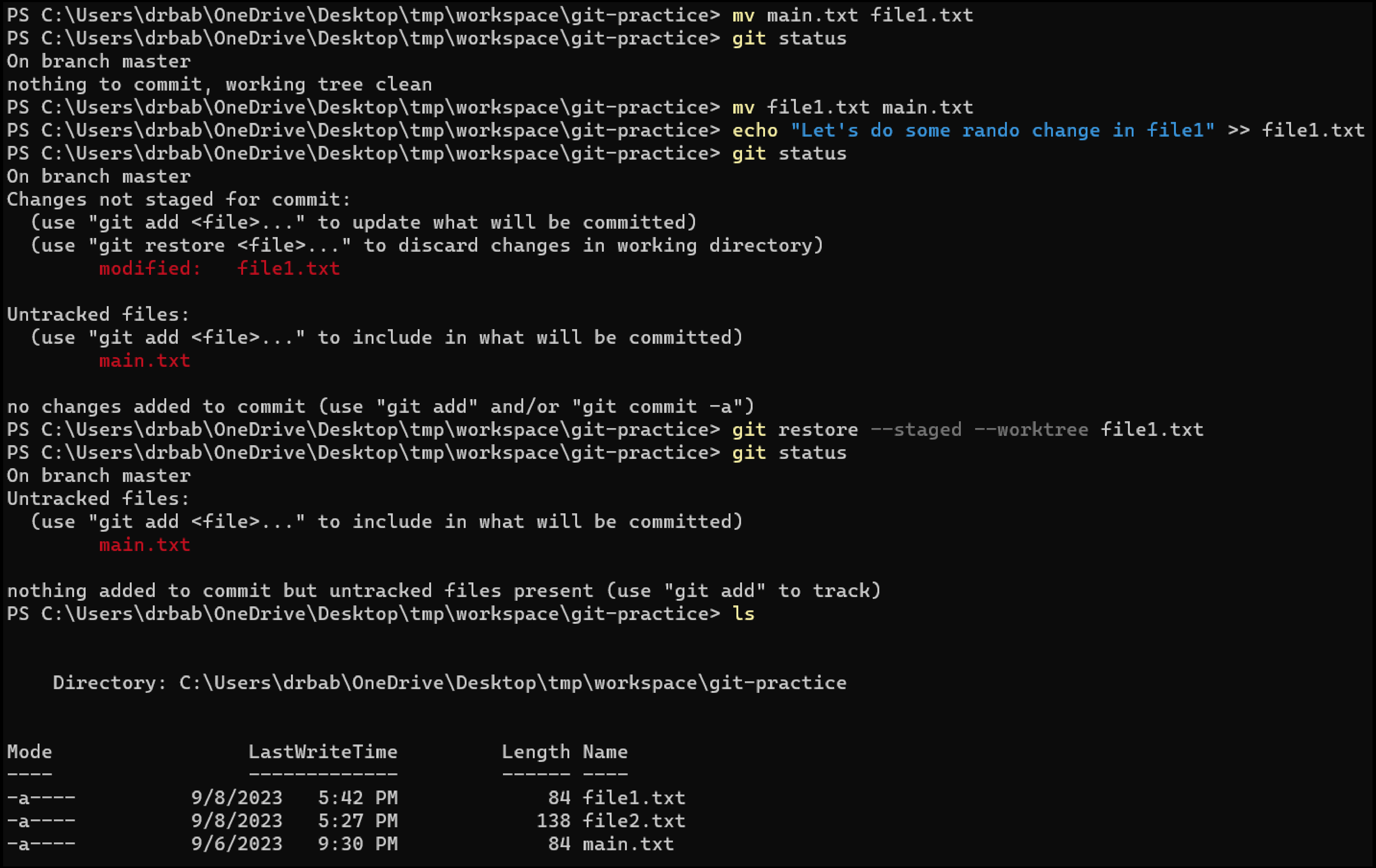
Question!

What happens if you renamed a file that is tracked by git?

- Suppose you have *file1.txt* and *file2.txt*
- Then you just ran *mv file1.txt main.txt*

1) You want to undo this change

2) You wan to stick with this (but do it faster!)



```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> mv main.txt file1.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git status
On branch master
nothing to commit, working tree clean
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> mv file1.txt main.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> echo "Let's do some rando change in file1" >> file1.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> 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

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

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git restore --staged --worktree file1.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        main.txt

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> ls


    Directory: C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----          9/8/2023   5:42 PM             84 file1.txt
-a----          9/8/2023   5:27 PM            138 file2.txt
-a----          9/6/2023   9:30 PM             84 main.txt
```

#3: How can we ignore some files?

gitignore

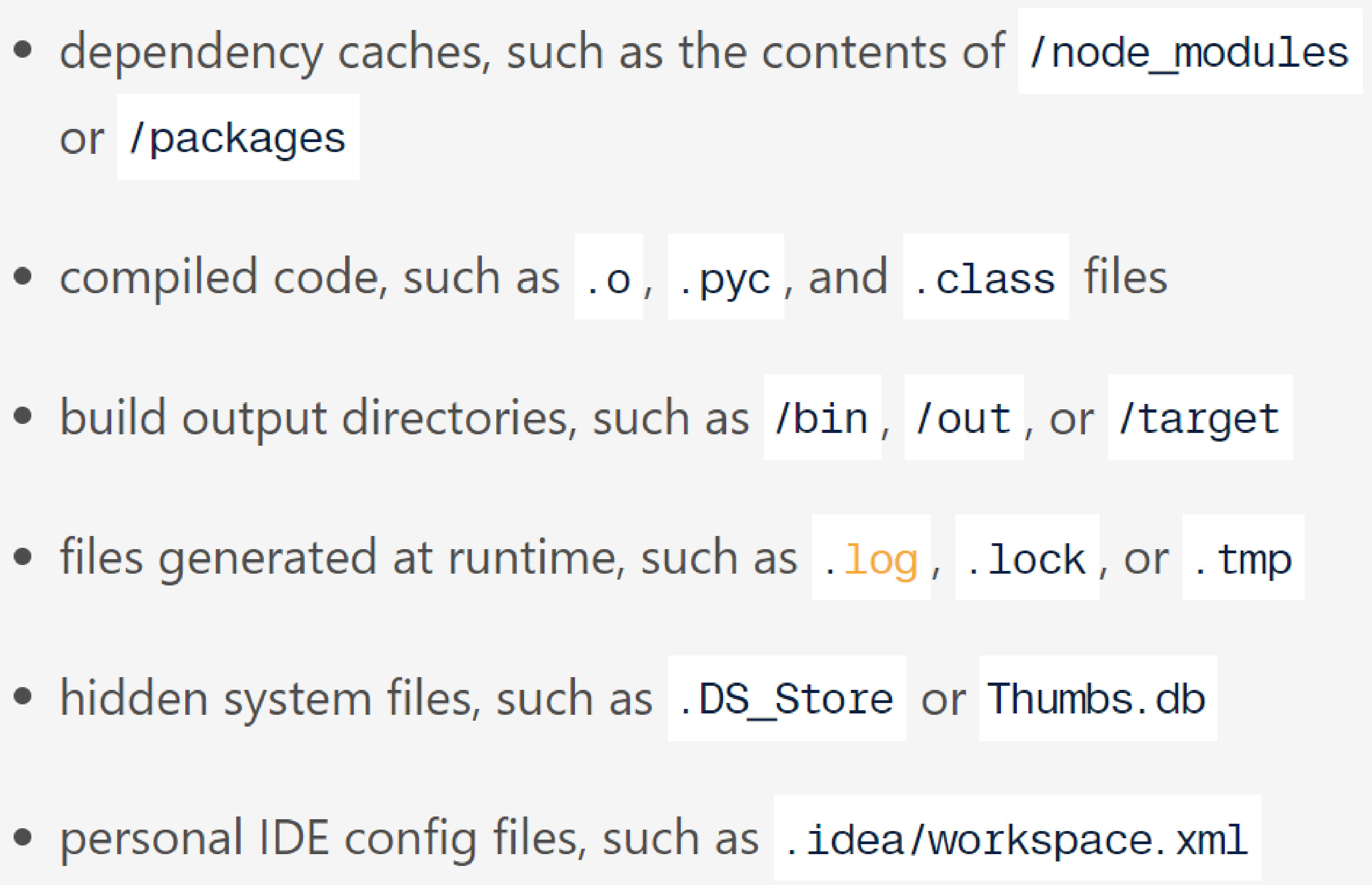
How to ignore files in git?

- Git sees every file in your working tree as one of three things:
  - *tracked* - a file which has been previously staged or committed;
  - *untracked* - a file which has not been staged or committed;
  - *ignored* - a file which Git has been explicitly told to ignore.

Ignored files are usually build artifacts and machine generated files that can be derived from your repository source or should otherwise not be committed.

How to ignore files in git?

- Some common examples are:



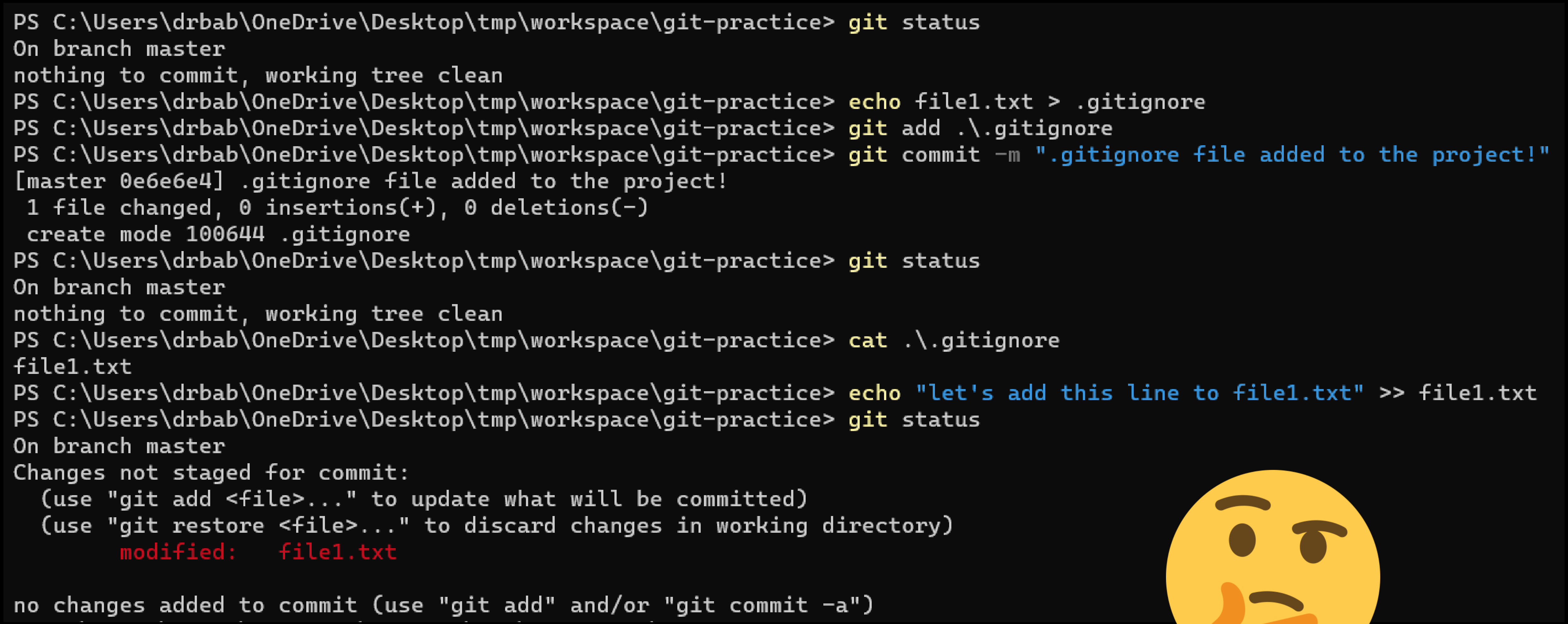
- dependency caches, such as the contents of `/node_modules` or `/packages`
- compiled code, such as `.o`, `.pyc`, and `.class` files
- build output directories, such as `/bin`, `/out`, or `/target`
- files generated at runtime, such as `.log`, `.lock`, or `.tmp`
- hidden system files, such as `.DS_Store` or `Thumbs.db`
- personal IDE config files, such as `.idea/workspace.xml`

How to ignore files in git?

- Create *.gitignore* file and place it at the root of your project
- Edit this file by adding whatever you want to force git to ignore!
    - *Mylog.log*
    - **.log*
    - *dev/*
    - *output/app.exe*

- *git add .gitignore*
- *git commit –m ".gitignore added"*

There is no explicit git ignore command: the .gitignore file must be edited and committed by hand when you have new files that you wish to ignore.



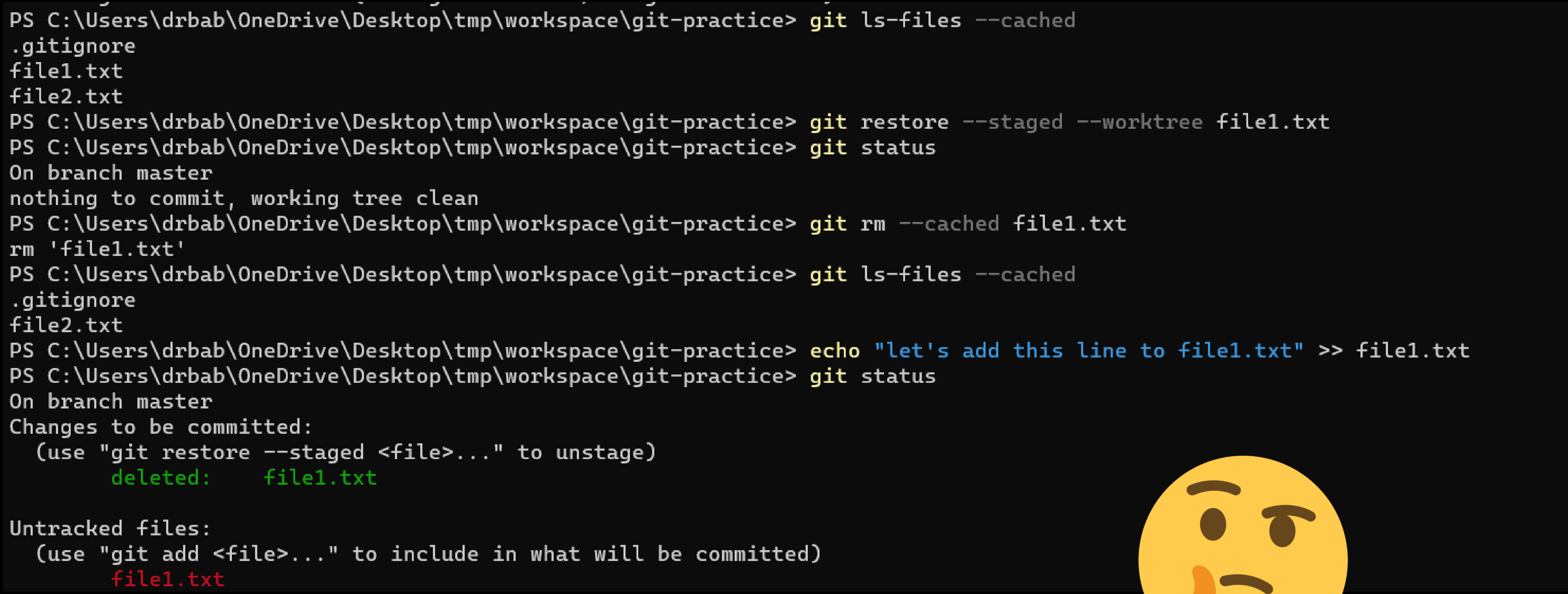
```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git status
On branch master
nothing to commit, working tree clean
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> echo file1.txt > .gitignore
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git add .\.gitignore
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git commit -m ".gitignore file added to the project!"
[master 0e6e6e4] .gitignore file added to the project!
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 .gitignore
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git status
On branch master
nothing to commit, working tree clean
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> cat .\.gitignore
file1.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> echo "let's add this line to file1.txt" >> file1.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> 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")
```

What is the issue?!

- This only works if you haven’t already included the file1.txt to the staged area.
- If you accidentally included them into the SA and then later added them to *.gitignore* , git not going to ignore them!

If you want to ignore a file that you've committed in the past, you'll need to delete the file from your SA and then add a .gitignore rule for it.

git rm --cached [file’s name]



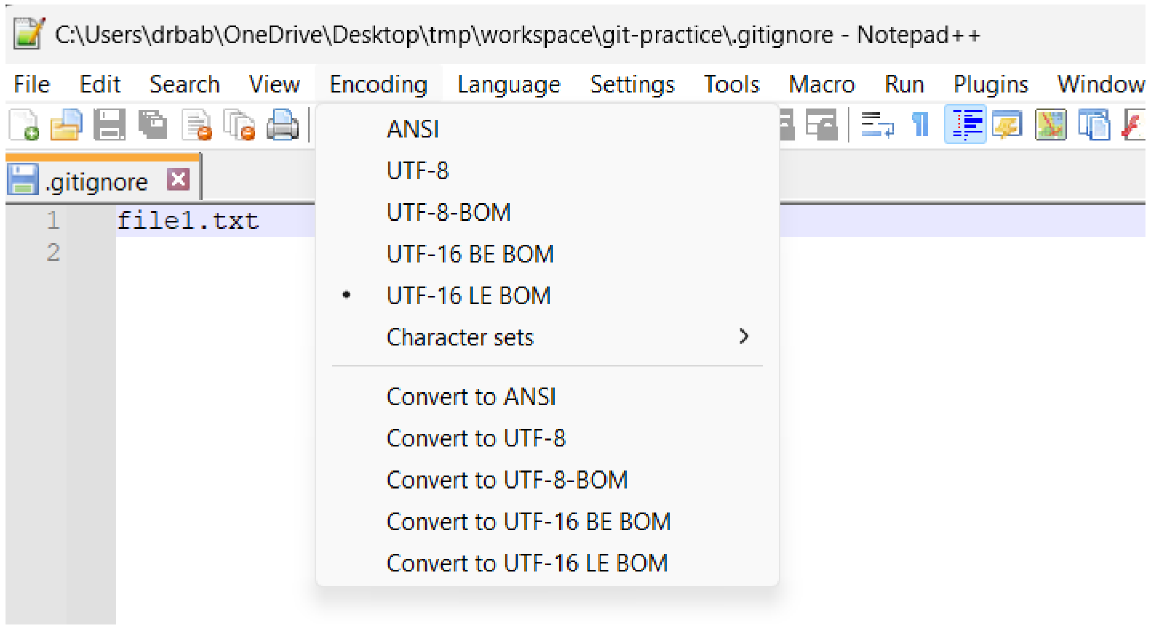
```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git ls-files --cached
.gitignore
file1.txt
file2.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git restore --staged --worktree file1.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git status
On branch master
nothing to commit, working tree clean
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git rm --cached file1.txt
rm 'file1.txt'
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git ls-files --cached
.gitignore
file2.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> echo "let's add this line to file1.txt" >> file1.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    file1.txt

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

What is the issue?!

- Something irrelevant to our git technique!

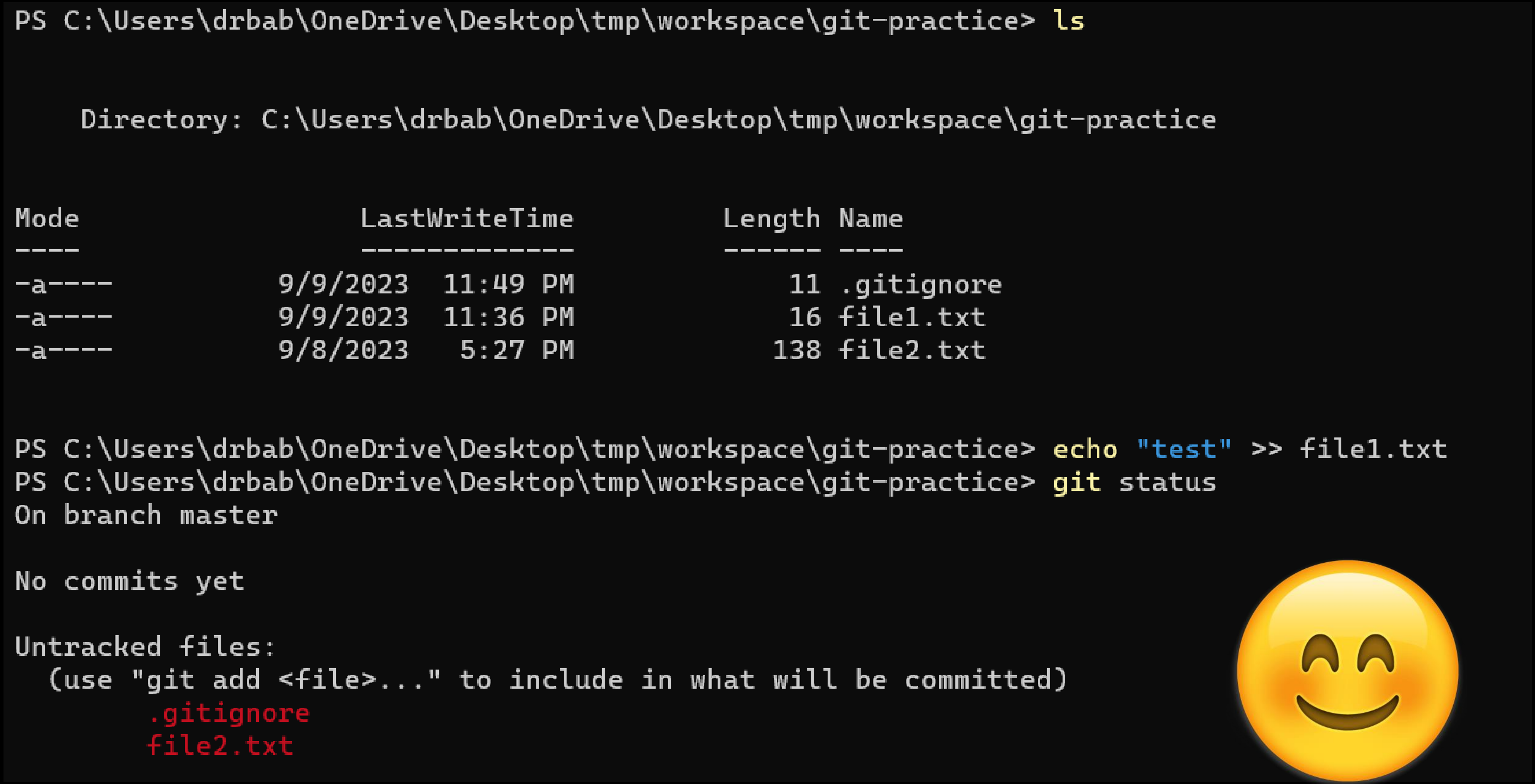
- It turns out PowerShell stores data in UTF-16 LE BOM format that is not an expected format by git.

- It has to be converted to UTF-8



C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice\.gitignore - Notepad++

```
file1.txt
```

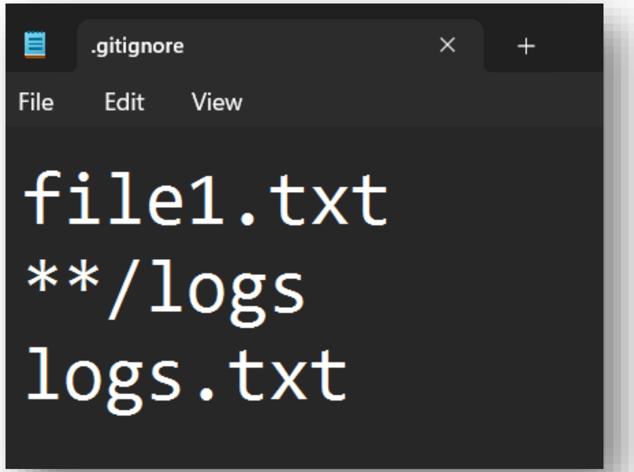


```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> ls


    Directory: C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----          9/9/2023  11:49 PM             11 .gitignore
-a----          9/9/2023  11:36 PM             16 file1.txt
-a----          9/8/2023   5:27 PM            138 file2.txt


PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> echo "test" >> file1.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git status
On branch master

No commits yet

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

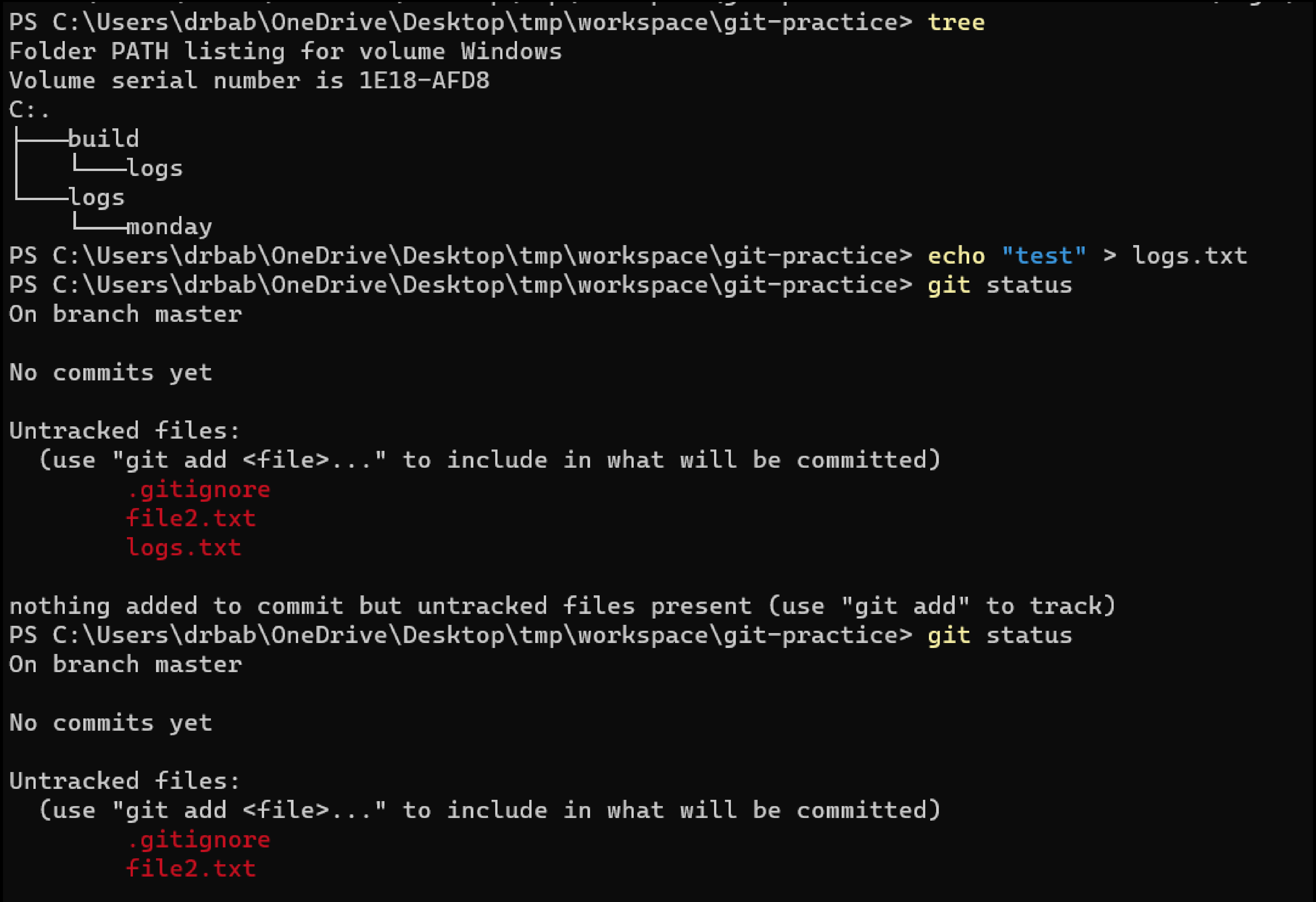
Question 1

- How to remove the content of the *logs* directory regardless of where it is located?

```
logs/debug.log
logs/monday/foo.bar
build/logs/debug.log
```



```
file1.txt
**/logs
logs.txt
```



```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> tree
Folder PATH listing for volume Windows
Volume serial number is 1E18-AFD8
C:.
├───build
│   └───logs
└───logs
    └───monday
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> echo "test" > logs.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git status
On branch master

No commits yet

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

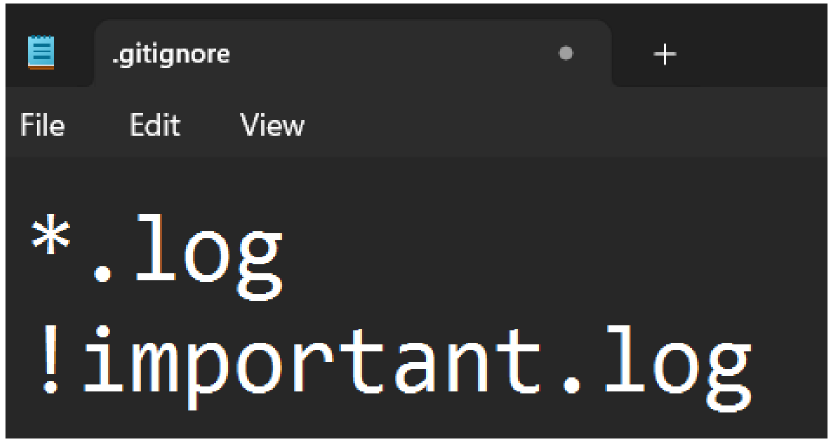
Question 2

- How to remove all files with *.log* extension except the *important.log* file?

debug.log
trace.log

but not

important.log
logs/important.log



```
*.log
!important.log
```

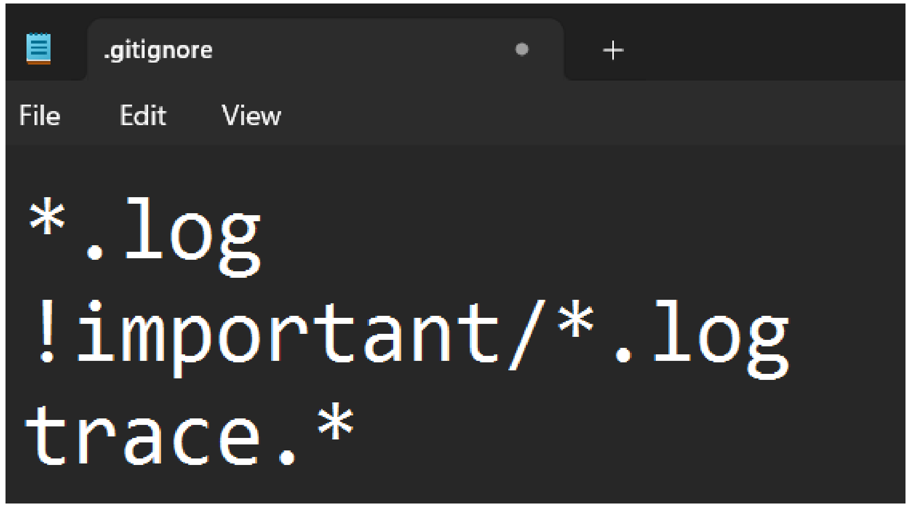
Question 3

- How to remove all files with *.log* extension except the *all log* files in the important directory?

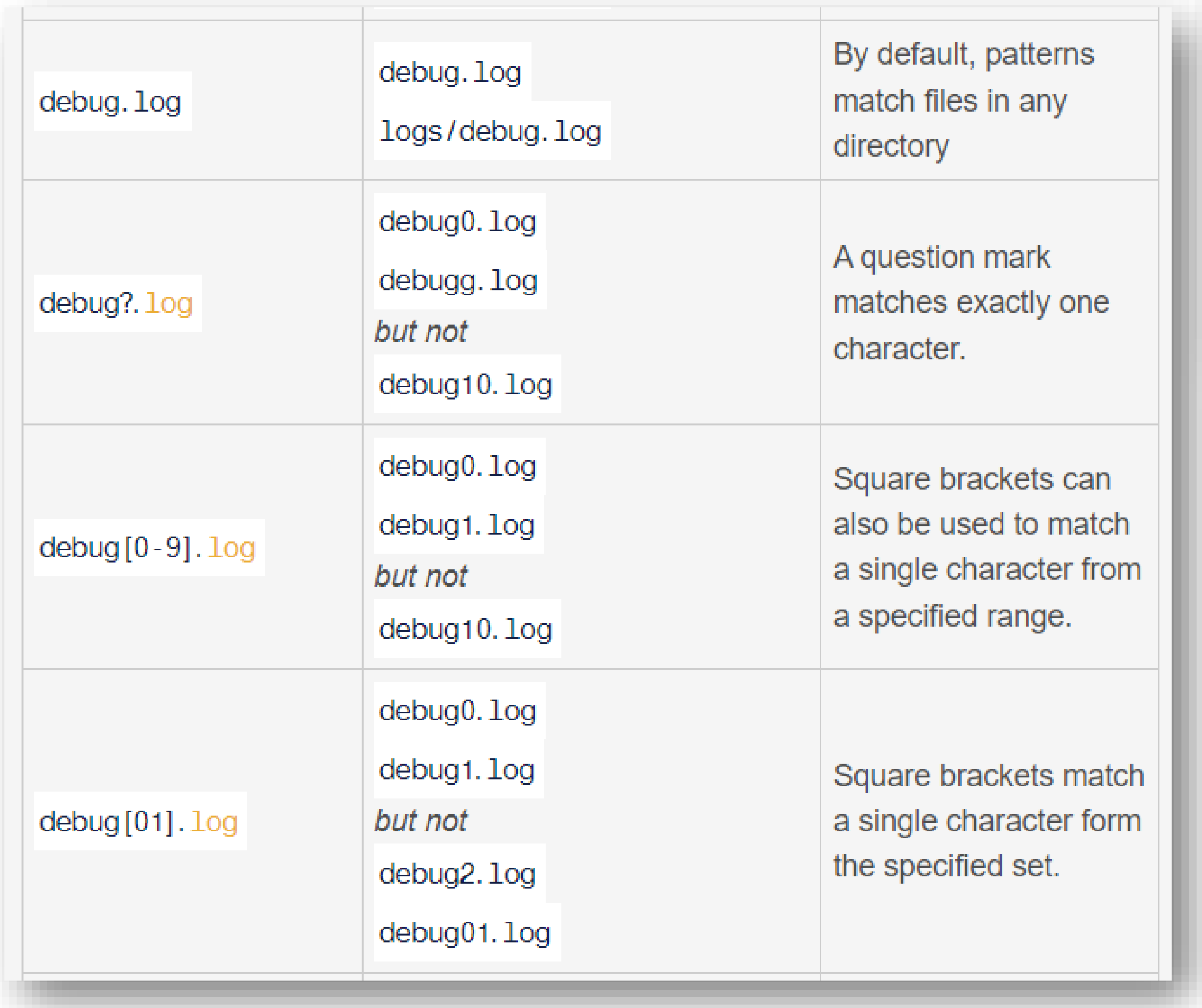
debug.log
important/trace.log

but not

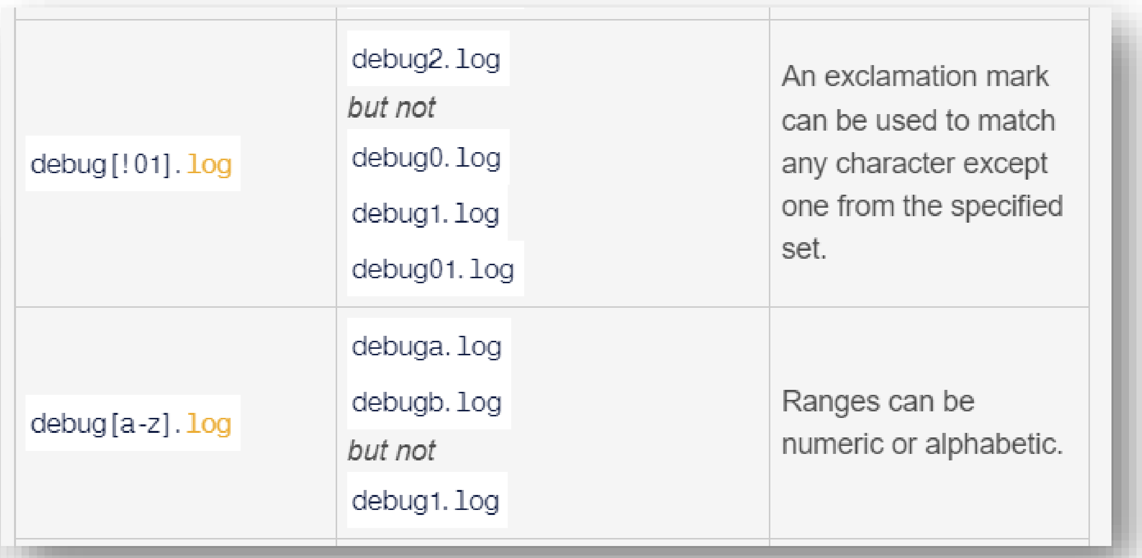
important/debug.log



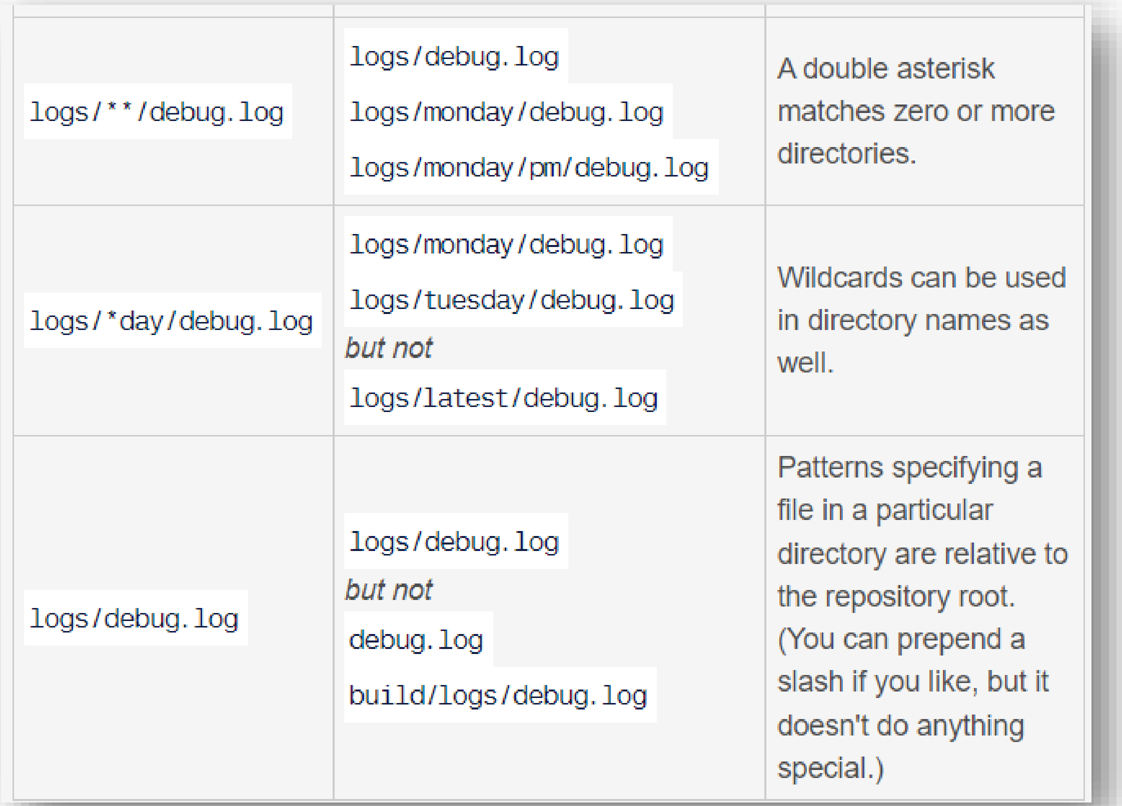
```
*.log
!important/*.log
trace.*
```



debug.log	debug.log logs/debug.log	By default, patterns match files in any directory
debug?.log	debug0.log debugg.log *but not* debug10.log	A question mark matches exactly one character.
debug[0-9].log	debug0.log debug1.log *but not* debug10.log	Square brackets can also be used to match a single character from a specified range.
debug[01].log	debug0.log debug1.log *but not* debug2.log debug01.log	Square brackets match a single character form the specified set.



debug[!01].log	debug2.log *but not* debug0.log debug1.log debug01.log	An exclamation mark can be used to match any character except one from the specified set.
debug[a-z].log	debuga.log debugb.log *but not* debug1.log	Ranges can be numeric or alphabetic.



logs/**/debug.log	logs/debug.log logs/monday/debug.log logs/monday/pm/debug.log	A double asterisk matches zero or more directories.
logs/*day/debug.log	logs/monday/debug.log logs/tuesday/debug.log *but not* logs/latest/debug.log	Wildcards can be used in directory names as well.
logs/debug.log	logs/debug.log *but not* debug.log build/logs/debug.log	Patterns specifying a file in a particular directory are relative to the repository root. (You can prepend a slash if you like, but it doesn't do anything special.)

#4: How can we get a file status in git?

Get short status in *git*

- What is the problem with the output in *git status* ?

```
$ 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
```

- The output shown above is verbose and descriptive. At times we may not want a detailed output.

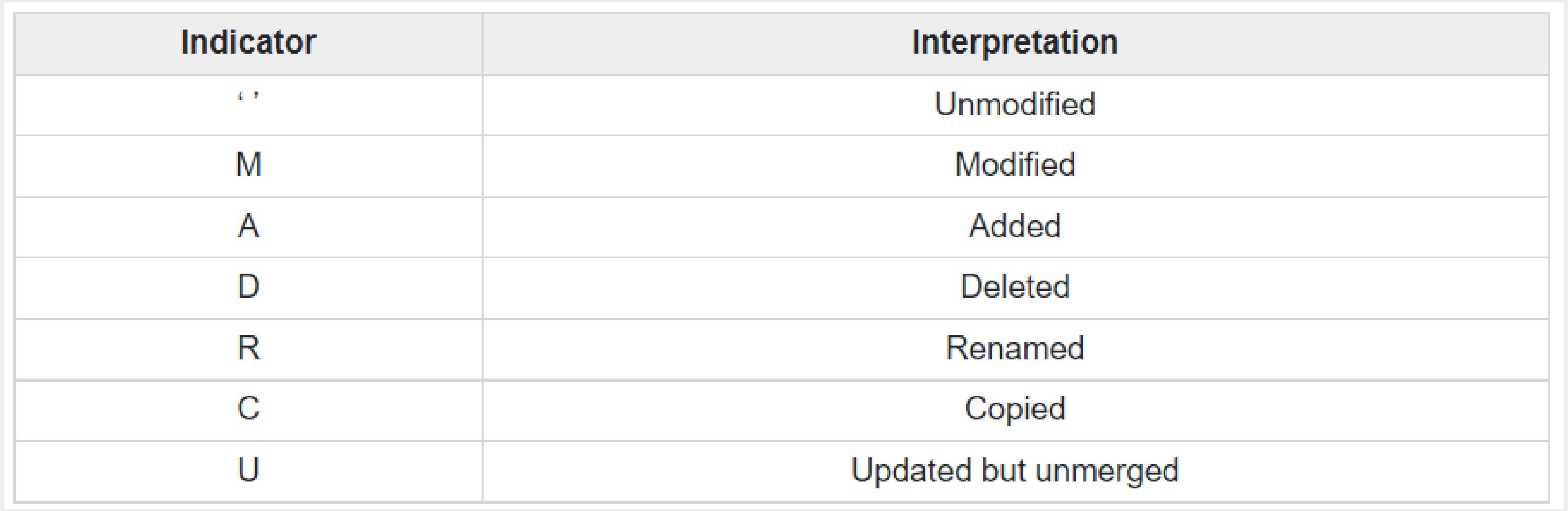
- The *--short* or *-s* flag should can be supplied to the git status command to get the output in short format.

Get short status in *git*

- The short status of each path is shown as one of the following `XY PATH`

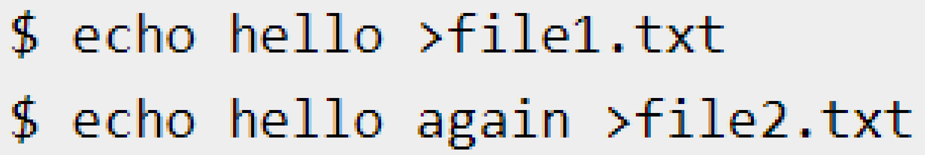
- XY is a two–letter status code, where:
  - X represents status of the file in the staging area
  - Y represents status of the file in the working directory.

- The following table lists the possible values for X and Y.



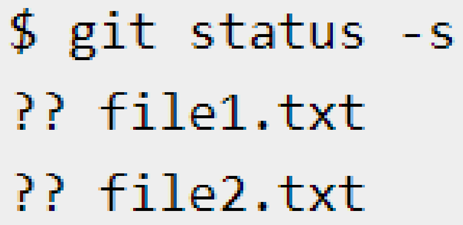
Indicator	Interpretation
‘ ’	Unmodified
M	Modified
A	Added
D	Deleted
R	Renamed
C	Copied
U	Updated but unmerged

Get short status in *git* – output interpretation



```
$ echo hello >file1.txt
$ echo hello again >file2.txt
```

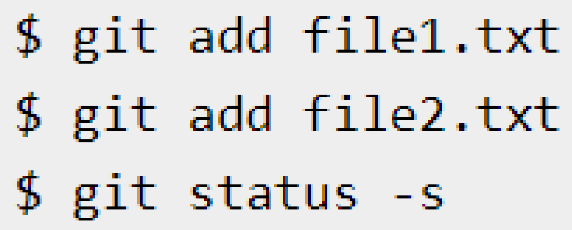
- Use the short status format to view the status: *git status –s*



```
$ git status -s
?? file1.txt
?? file2.txt
```

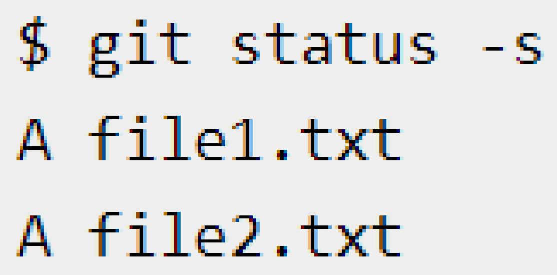
- (**??**) indicates that the files are untracked.

Get short status in *git* – output interpretation



```
$ git add file1.txt
$ git add file2.txt
$ git status -s
```

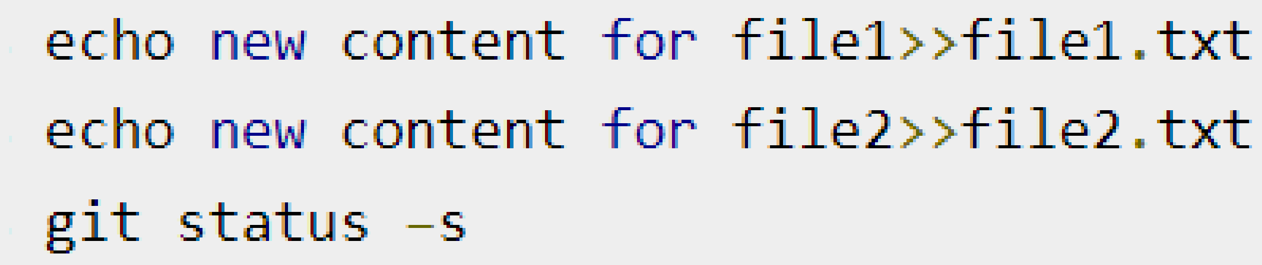
- The output (A) indicates that the files have been added to the staging area.
- *Note that the indicator A is left aligned and hence, denotes the status of the staging area*.



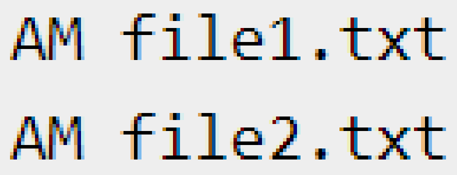
```
$ git status -s
A file1.txt
A file2.txt
```

- Let us modify the files in the working tree and verify the status.

Get short status in *git* – output interpretation



```
echo new content for file1>>file1.txt
echo new content for file2>>file2.txt
git status -s
```



```
AM file1.txt
AM file2.txt
```

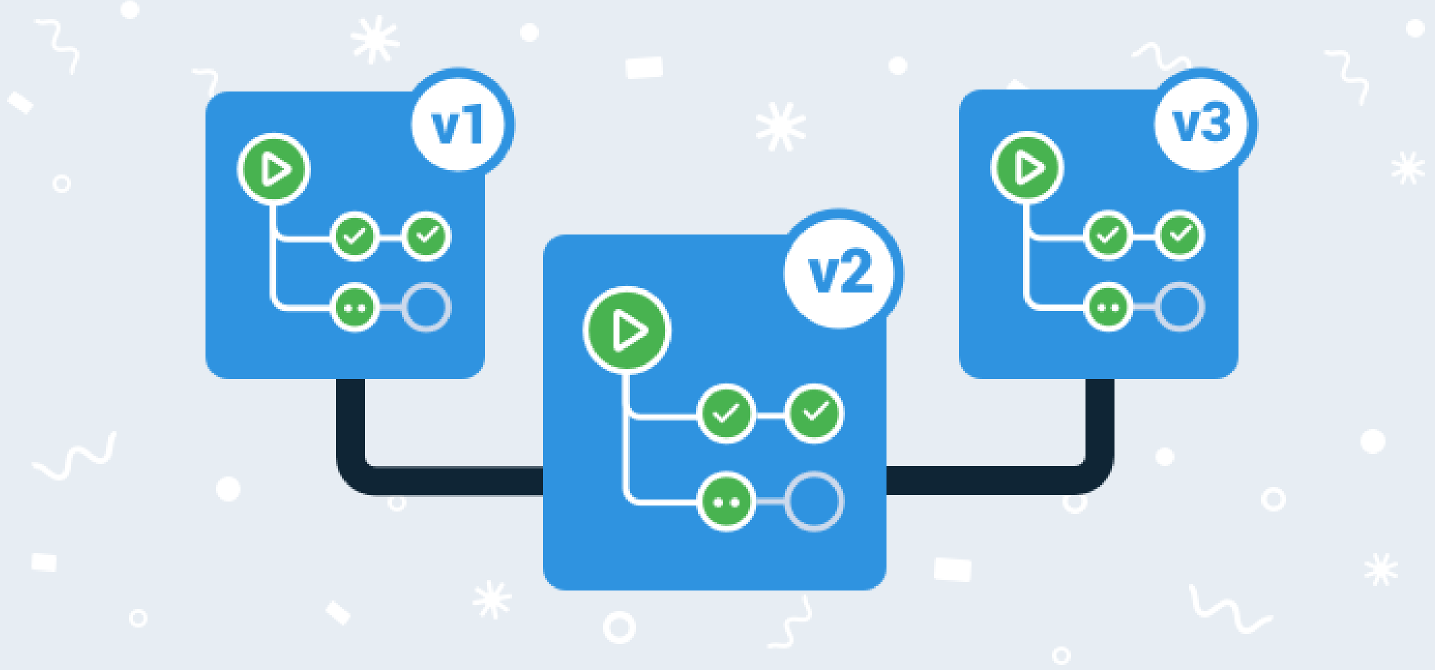
- The output (AM) indicates that the content of the file in the staging area and the working tree are different.
- *It means that the file's content has been modified in the working tree, but the changes to the file are not staged.*

Get short status in *git* – output interpretation

- Let us now commit the changes and observe the status.
  - *git commit –m "files added"*
  - *git status –s*

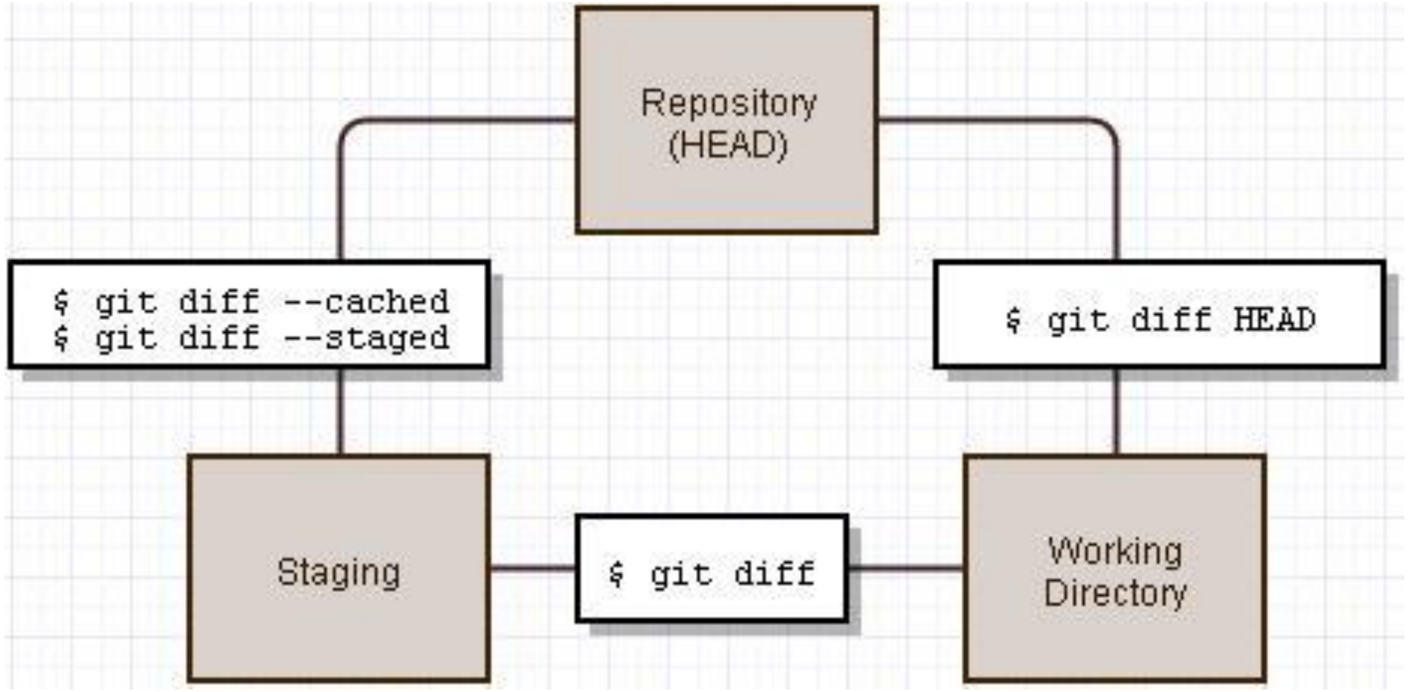
- *A blank output indicates that the working directory is clean.*

#5: How can we compare different versions?

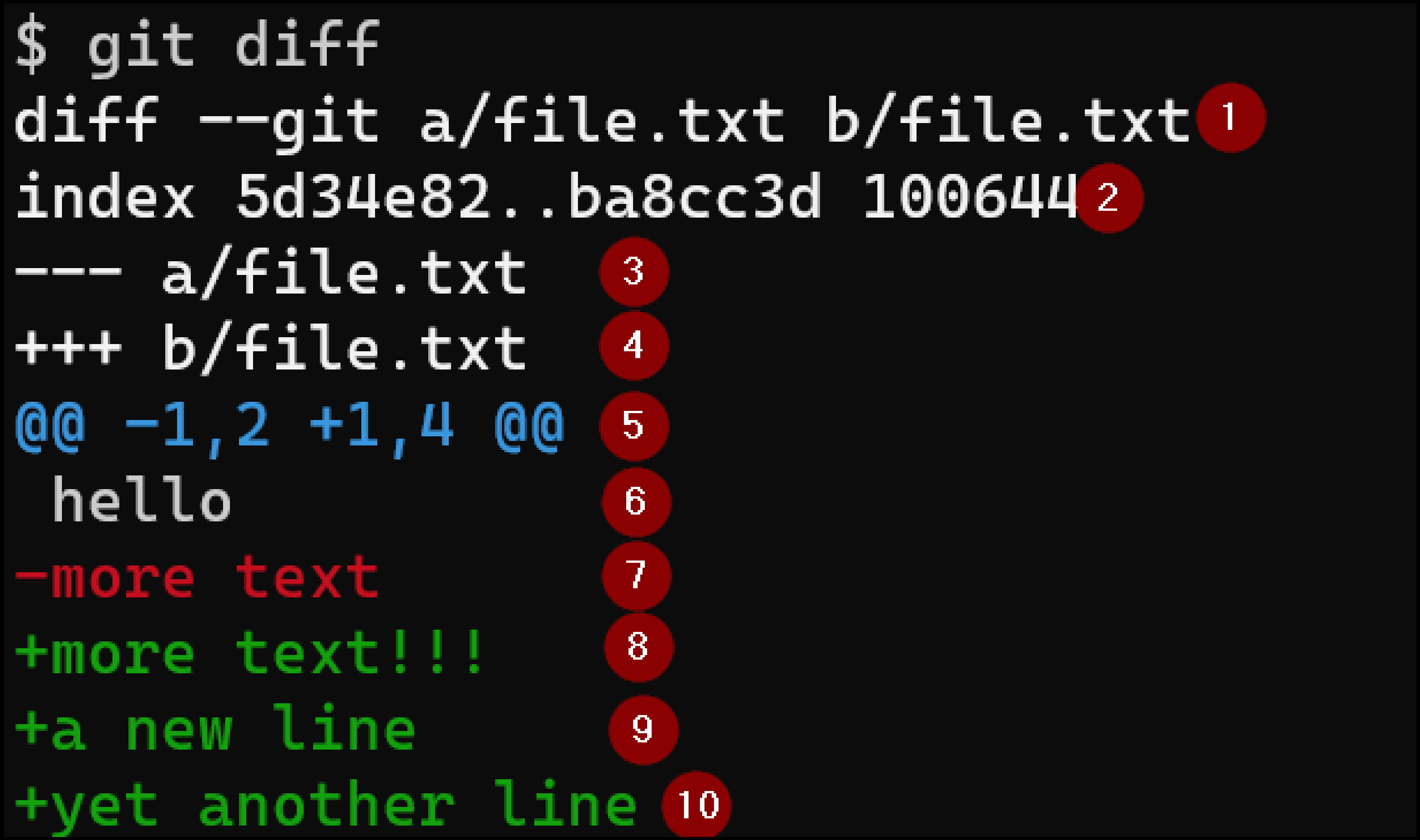


Git diff

- Always review what you have in the staging area before make any commits
- The status command shows only the files that have been affected
- To get more detail and check the exact lines of code that we have staged: use *diff* command



How to interpret the output of a *git diff* command



```
$ git diff
diff --git a/file.txt b/file.txt
index 5d34e82..ba8cc3d 100644
--- a/file.txt
+++ b/file.txt
@@ -1,2 +1,4 @@
 hello
-more text
+more text!!!
+a new line
+yet another line
```

How to interpret the output of a *git diff* command

1. The output starts by declaring the files. Here, we have the traditional *a* and *b* as prefixes.

2. This line shows us the *short versions of the hash* of the original and changed versions of the file. We also see the mode (100644), which indicates this is a regular, non-executable, file.

3. In the source file (a), the minus sign means it is the original file.

4. In the destination (or changed) file (b), the plus sign means it the new file.

5. We'll skip this one now and get back to it later.

How to interpret the output of a *git diff* command

6. Here we have a line that didn’t change.
7. This line indicates that the “more text” line was removed.
8. This line indicates that the “more text!!!” line was added.

“more text” ==> “more text!!!”

9. Here we have an indication of a new line “added.”
10. This indicates the first line added.
11. Finally, this indicates the last line added.

How to interpret the output of a *git diff* command

- Let's now cover line 5. It certainly looks the most cryptic one, but it's actually quite easy to understand. Here we have **hunks**, which are defined as "**groups of differing lines**" in the files.

  - The -1,2 part indicates we'll be seeing two lines from the original file, starting at line 1.

  - The minus sign, as you've seen, means "old" or "original."

  - The +1,4 indicates that, regarding the new version, we'll be seeing four lines, starting at line 1.

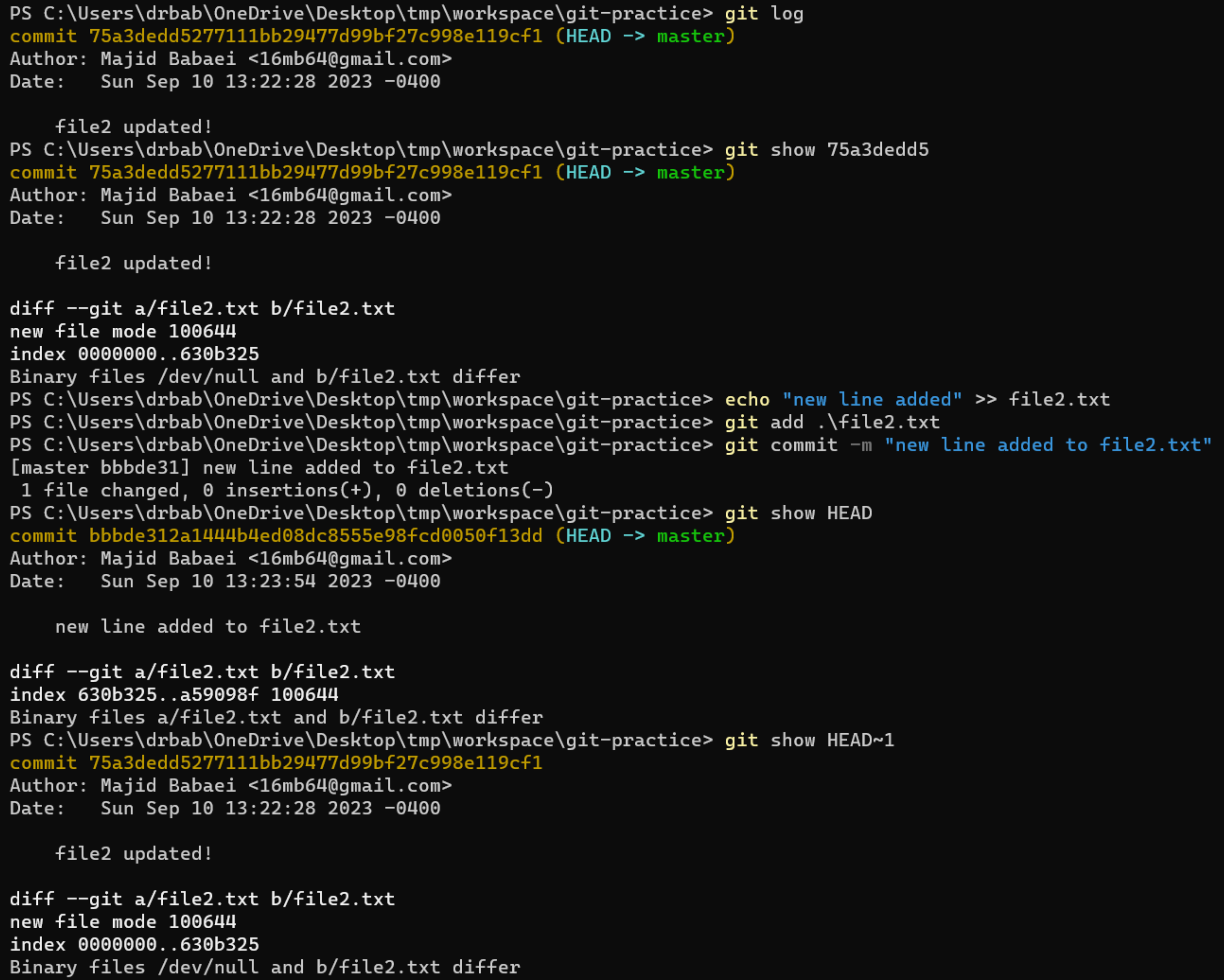
  - The plus sign indicates new.

#6: How can we view content of a commit?

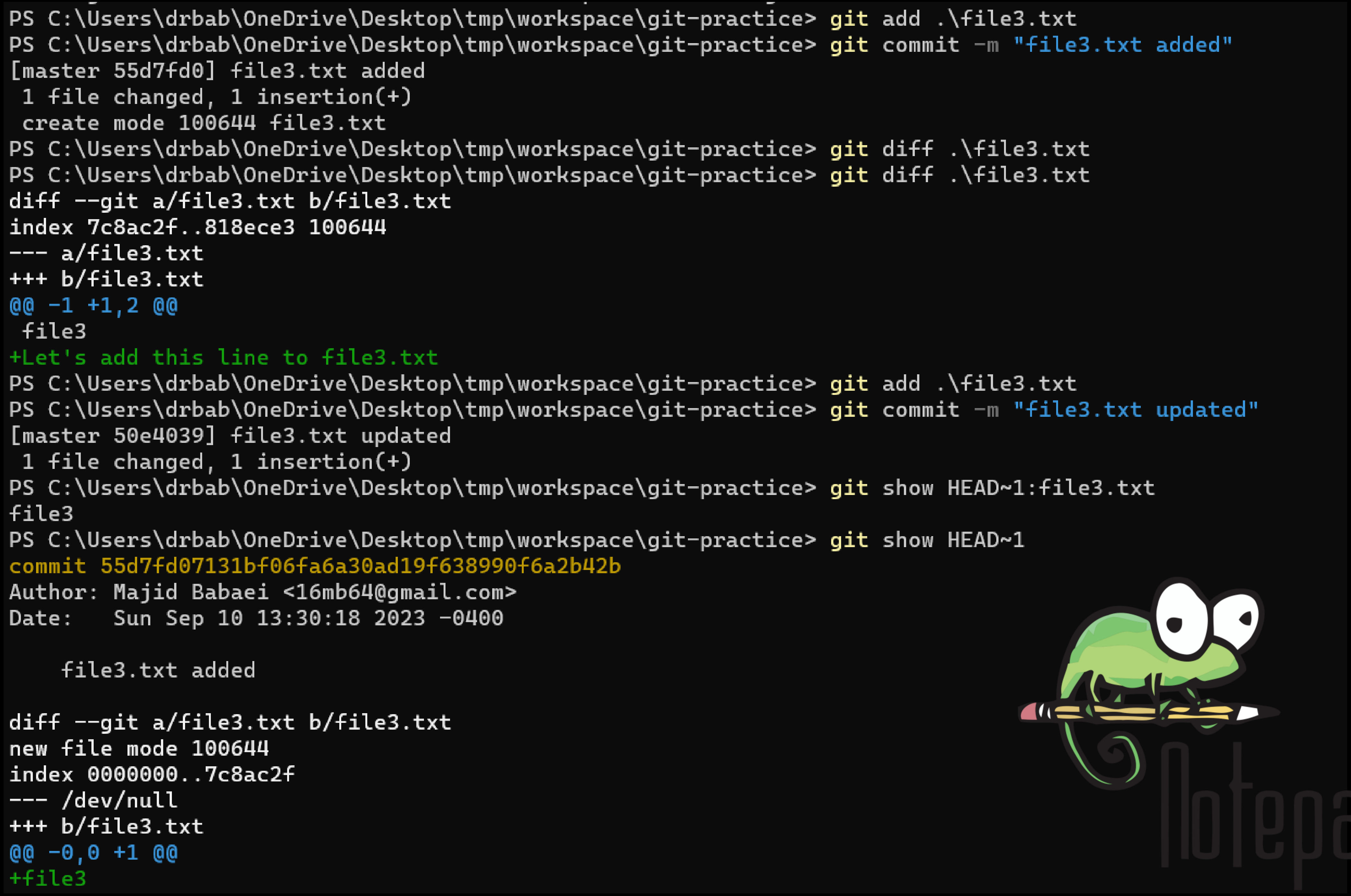
Viewing a commit

- What if we want to see what exactly we have changed in a commit?
  - *git show [ref to a commit]*

- There are a couple of ways you can refer to a commit
  - Using its unique ID
  - Using HEAD~[the number of steps we want to go back]



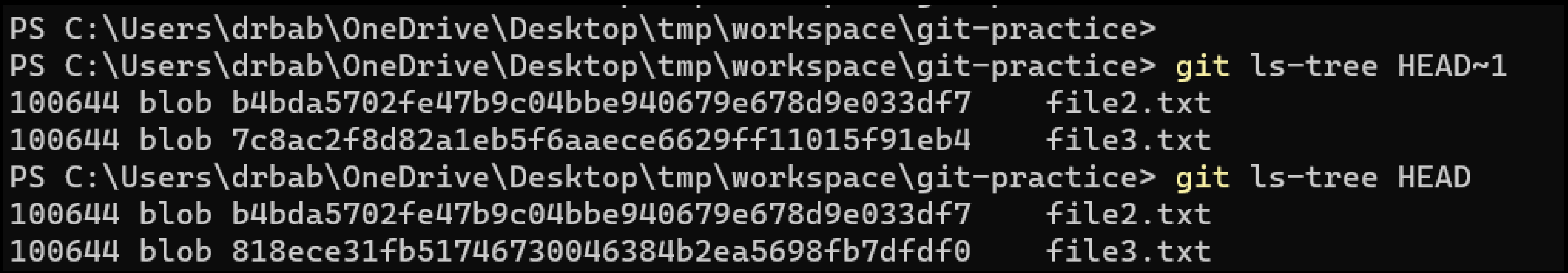
```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git log
commit 75a3dedd5277111bb29477d99bf27c998e119cf1 (HEAD -> master)
Author: Majid Babaei <16mb64@gmail.com>
Date:   Sun Sep 10 13:22:28 2023 -0400

    file2 updated!
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git show 75a3dedd5
commit 75a3dedd5277111bb29477d99bf27c998e119cf1 (HEAD -> master)
Author: Majid Babaei <16mb64@gmail.com>
Date:   Sun Sep 10 13:22:28 2023 -0400

    file2 updated!

diff --git a/file2.txt b/file2.txt
new file mode 100644
index 0000000..630b325
Binary files /dev/null and b/file2.txt differ
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> echo "new line added" >> file2.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git add .\file2.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git commit -m "new line added to file2.txt"
[master bbbde31] new line added to file2.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git show HEAD
commit bbbde312a1444b4ed08dc8555e98fcd0050f13dd (HEAD -> master)
Author: Majid Babaei <16mb64@gmail.com>
Date:   Sun Sep 10 13:23:54 2023 -0400

    new line added to file2.txt

diff --git a/file2.txt b/file2.txt
index 630b325..a59098f 100644
Binary files a/file2.txt and b/file2.txt differ
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git show HEAD~1
commit 75a3dedd5277111bb29477d99bf27c998e119cf1
Author: Majid Babaei <16mb64@gmail.com>
Date:   Sun Sep 10 13:22:28 2023 -0400

    file2 updated!

diff --git a/file2.txt b/file2.txt
new file mode 100644
index 0000000..630b325
Binary files /dev/null and b/file2.txt differ
```



```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git add .\file3.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git commit -m "file3.txt added"
[master 55d7fd0] file3.txt added
 1 file changed, 1 insertion(+)
 create mode 100644 file3.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git diff .\file3.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git diff .\file3.txt
diff --git a/file3.txt b/file3.txt
index 7c8ac2f..818ece3 100644
--- a/file3.txt
+++ b/file3.txt
@@ -1 +1,2 @@
 file3
+Let's add this line to file3.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git add .\file3.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git commit -m "file3.txt updated"
[master 50e4039] file3.txt updated
 1 file changed, 1 insertion(+)
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git show HEAD~1:file3.txt
file3
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git show HEAD~1
commit 55d7fd07131bf06fa6a30ad19f638990f6a2b42b
Author: Majid Babaei <16mb64@gmail.com>
Date:   Sun Sep 10 13:30:18 2023 -0400

    file3.txt added

diff --git a/file3.txt b/file3.txt
new file mode 100644
index 0000000..7c8ac2f
--- /dev/null
+++ b/file3.txt
@@ -0,0 +1 @@
+file3
```

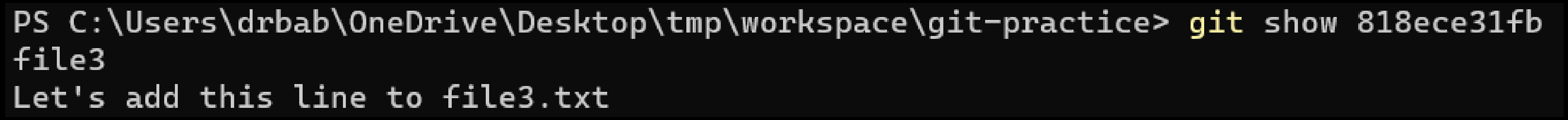
Viewing the list of the files in a commit

- What if we want to see all the files in a commit



```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice>
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git ls-tree HEAD~1
100644 blob b4bda5702fe47b9c04bbe940679e678d9e033df7    file2.txt
100644 blob 7c8ac2f8d82a1eb5f6aaece6629ff11015f91eb4    file3.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git ls-tree HEAD
100644 blob b4bda5702fe47b9c04bbe940679e678d9e033df7    file2.txt
100644 blob 818ece31fb51746730046384b2ea5698fb7dfdf0    file3.txt
```

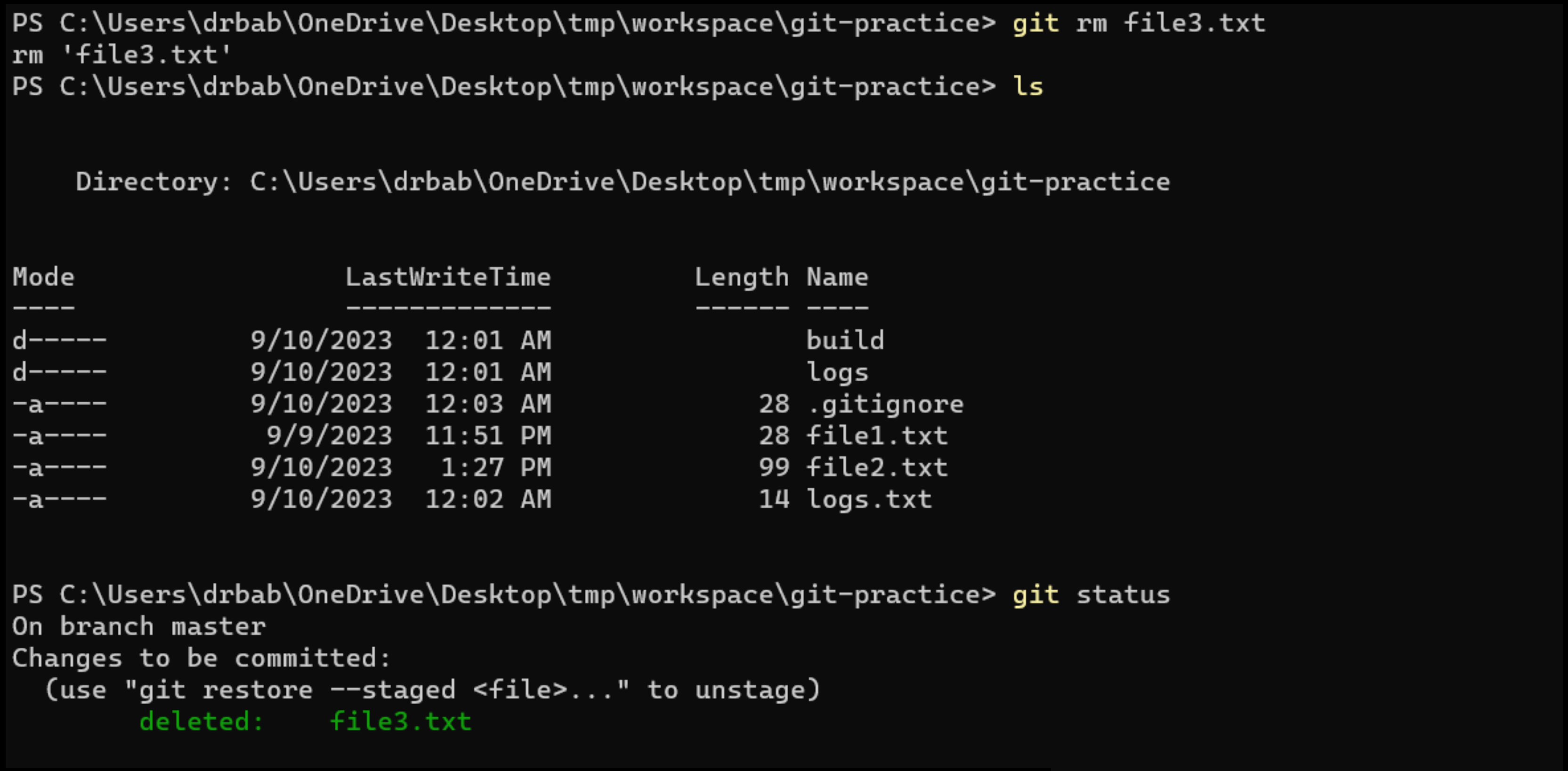
- *blob* represents files, *tree* represents directory
- To see the content of the file
  - *git show [id from the command ls-tree]*



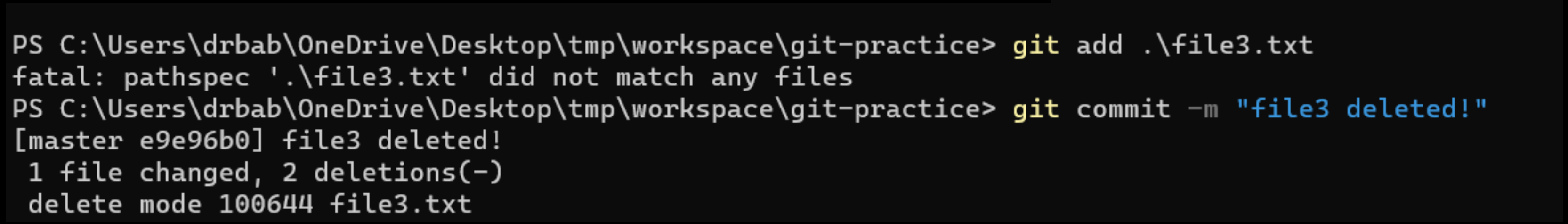
```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git show 818ece31fb
file3
Let's add this line to file3.txt
```

#7: How can we restore a file using the git history?

We removed a file accidently and committed the changes!



```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git rm file3.txt
rm 'file3.txt'
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> ls


    Directory: C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         9/10/2023  12:01 AM                build
d-----         9/10/2023  12:01 AM                logs
-a----         9/10/2023  12:03 AM             28 .gitignore
-a----          9/9/2023  11:51 PM             28 file1.txt
-a----         9/10/2023   1:27 PM             99 file2.txt
-a----         9/10/2023  12:02 AM             14 logs.txt


PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    file3.txt
```

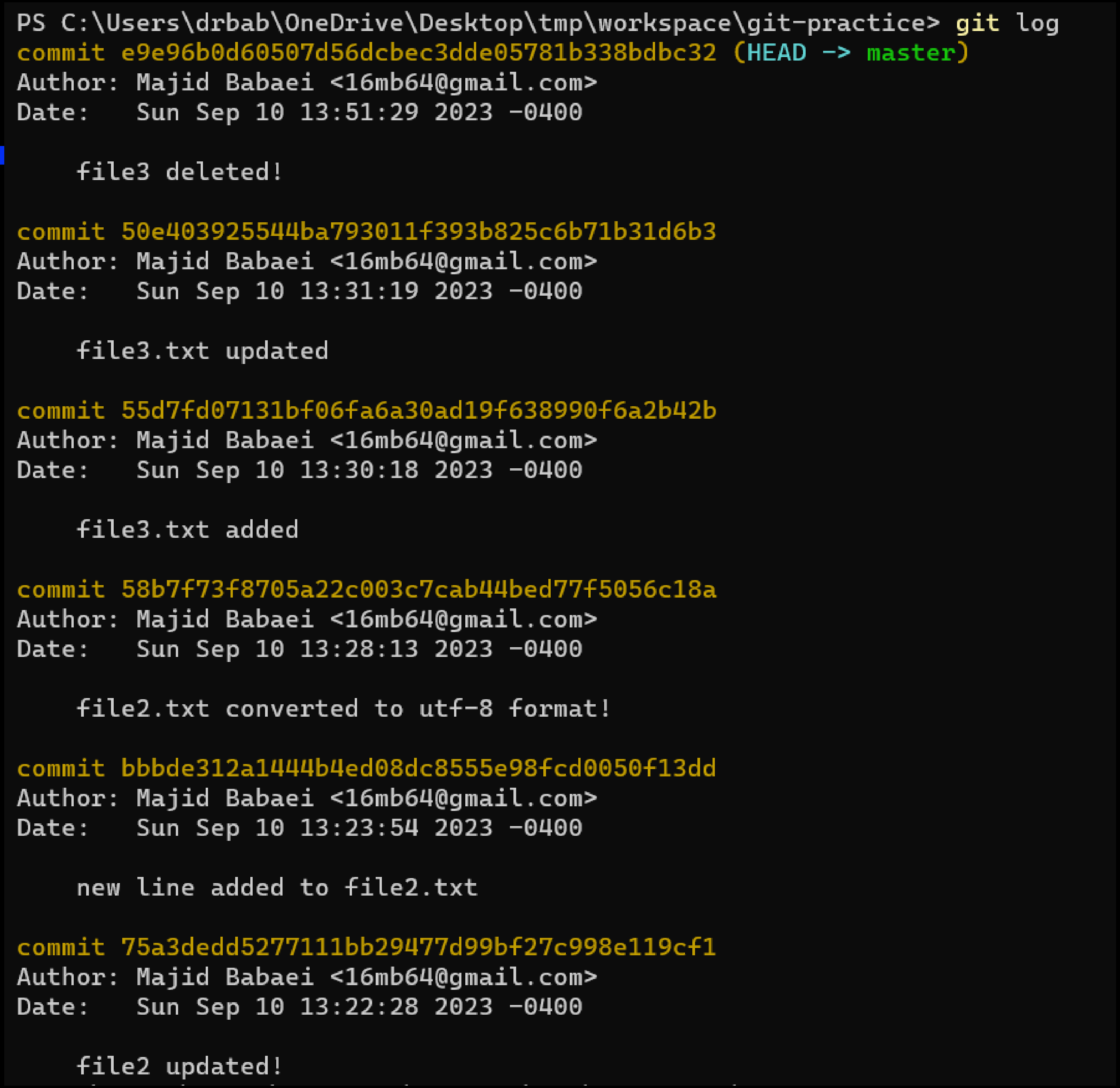


```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git add .\file3.txt
fatal: pathspec '.\file3.txt' did not match any files
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git commit -m "file3 deleted!"
[master e9e96b0] file3 deleted!
 1 file changed, 2 deletions(-)
 delete mode 100644 file3.txt
```

Let's look at the history

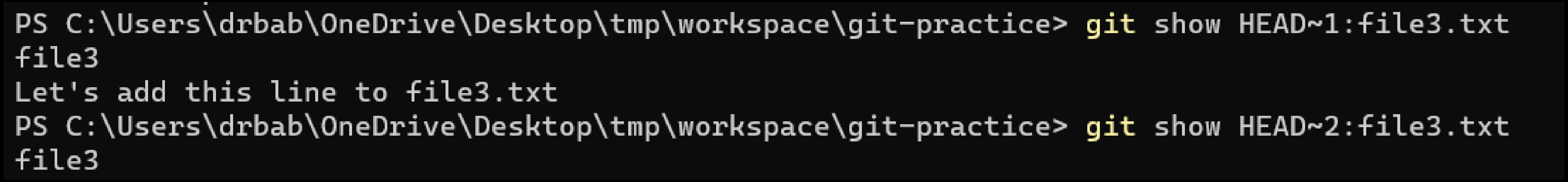
We need to decide what version of the file3.txt we would like to restore!

If you don't remember what it contains, ….



```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git log
commit e9e96b0d60507d56dcbec3dde05781b338bdbc32 (HEAD -> master)
Author: Majid Babaei <16mb64@gmail.com>
Date:   Sun Sep 10 13:51:29 2023 -0400

    file3 deleted!

commit 50e403925544ba793011f393b825c6b71b31d6b3
Author: Majid Babaei <16mb64@gmail.com>
Date:   Sun Sep 10 13:31:19 2023 -0400

    file3.txt updated

commit 55d7fd07131bf06fa6a30ad19f638990f6a2b42b
Author: Majid Babaei <16mb64@gmail.com>
Date:   Sun Sep 10 13:30:18 2023 -0400

    file3.txt added

commit 58b7f73f8705a22c003c7cab44bed77f5056c18a
Author: Majid Babaei <16mb64@gmail.com>
Date:   Sun Sep 10 13:28:13 2023 -0400

    file2.txt converted to utf-8 format!

commit bbbde312a1444b4ed08dc8555e98fcd0050f13dd
Author: Majid Babaei <16mb64@gmail.com>
Date:   Sun Sep 10 13:23:54 2023 -0400

    new line added to file2.txt

commit 75a3dedd5277111bb29477d99bf27c998e119cf1
Author: Majid Babaei <16mb64@gmail.com>
Date:   Sun Sep 10 13:22:28 2023 -0400

    file2 updated!
```

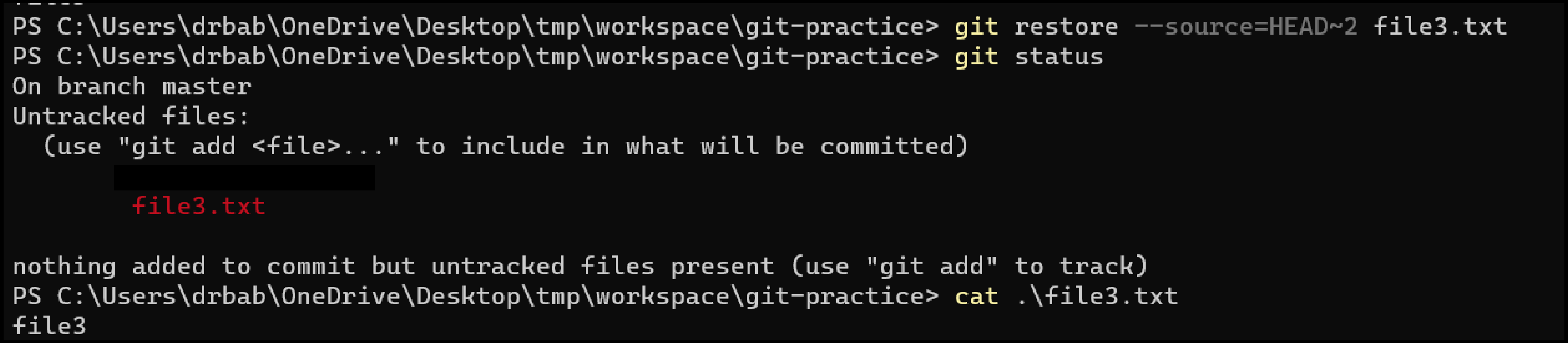
Let’s check the content of the file in some commits



```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git show HEAD~1:file3.txt
file3
Let's add this line to file3.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git show HEAD~2:file3.txt
file3
```

Okey! The version of the file in HEAD~2 looks good!

Let's define the source in the *git restore* command



```
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git restore --source=HEAD~2 file3.txt
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        [redacted]
        file3.txt

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\drbab\OneDrive\Desktop\tmp\workspace\git-practice> cat .\file3.txt
file3
```