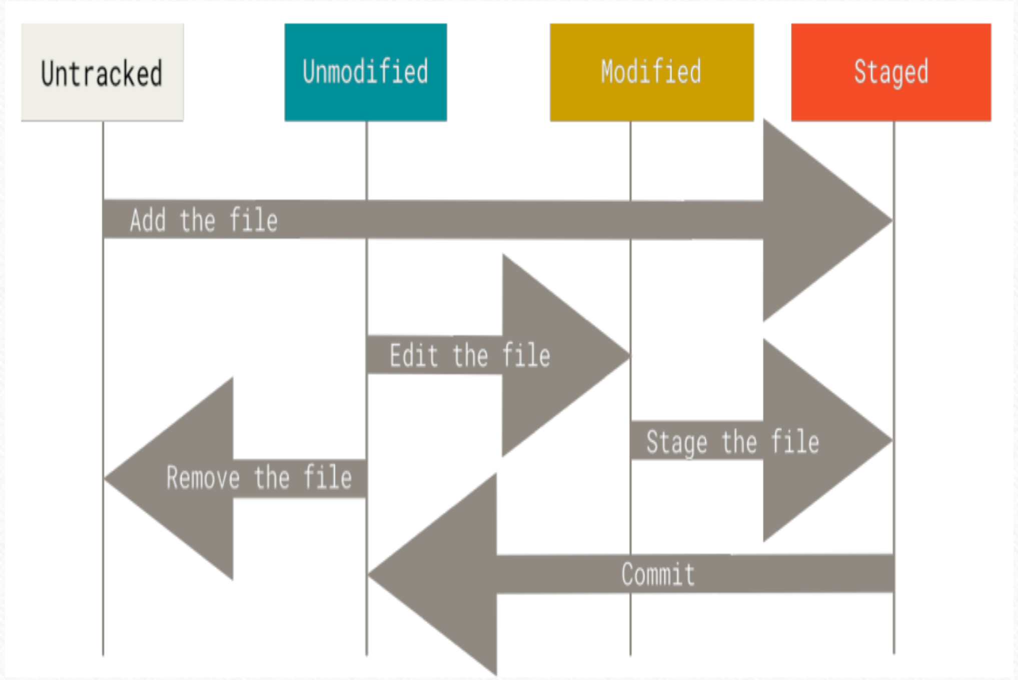
Git commands

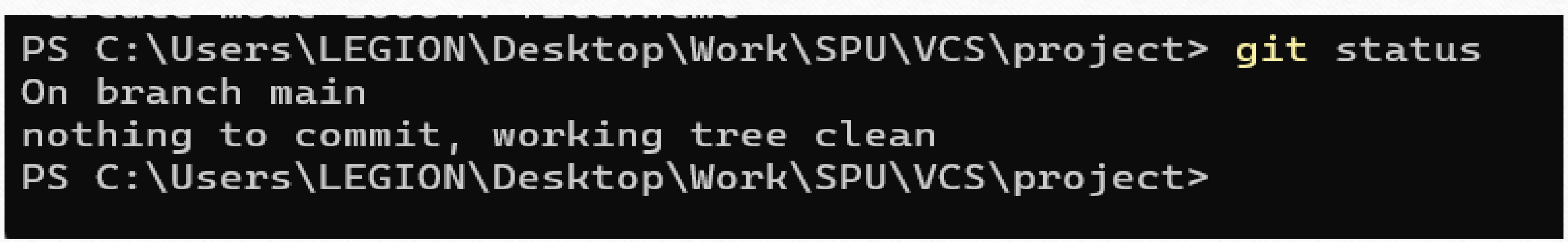
Recording changes to the repo

Each file in your working directory can be in one of two states: tracked or untracked. Tracked files are files that were in the last snapshot, as well as any newly staged files; they can be unmodified, modified, or staged. In short, tracked files are files that Git knows about.
Untracked files are everything else — any files in your working directory that were not in your last snapshot and are not in your staging area. When you first clone a repository, all of your files will be tracked and unmodified because Git just checked them out and you haven't edited anything.



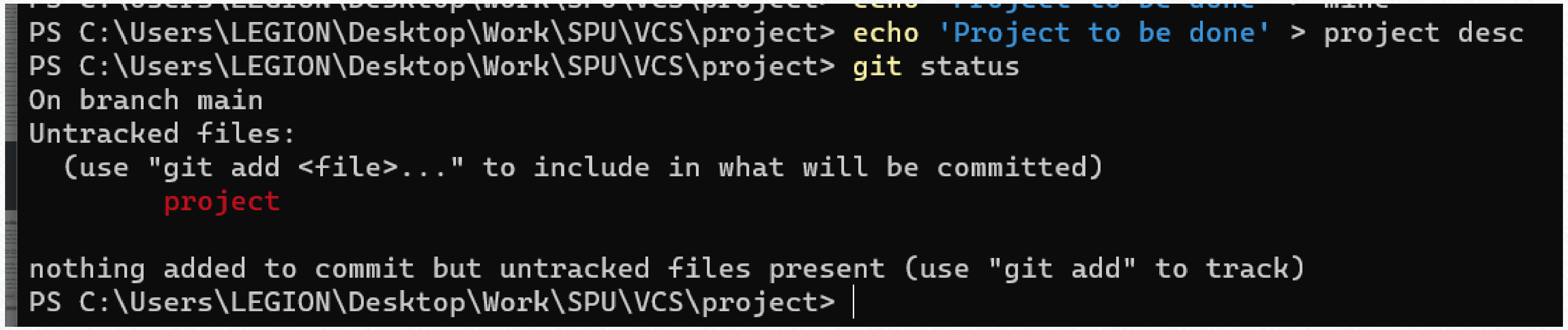
Checking the Status of Your Files

- The main tool you use to determine which files are in which state is the git status command. If you run this command directly after a clone, you should see something like this:



```
PS C:\Users\LEGION\Desktop\Work\SPU\VCS\project> git status
On branch main
nothing to commit, working tree clean
PS C:\Users\LEGION\Desktop\Work\SPU\VCS\project>
```

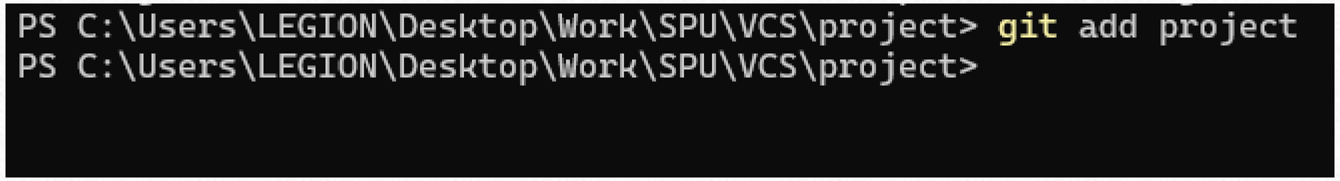
Let's say you add a new file to your project, a simple README file. If the file didn't exist before, and you run git status, you see your untracked file like so:



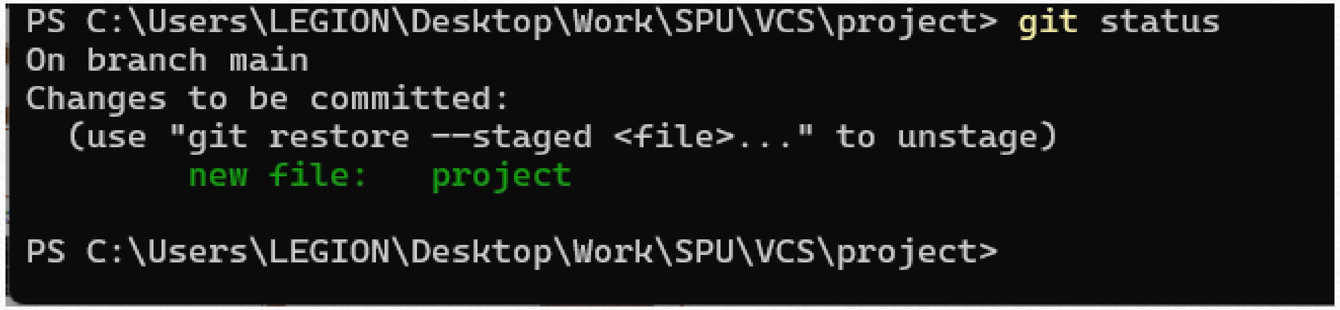
```
PS C:\Users\LEGION\Desktop\Work\SPU\VCS\project> echo 'Project to be done' > mine
PS C:\Users\LEGION\Desktop\Work\SPU\VCS\project> echo 'Project to be done' > project desc
PS C:\Users\LEGION\Desktop\Work\SPU\VCS\project> git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        project

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\LEGION\Desktop\Work\SPU\VCS\project>
```

Tracking New Files

- In order to begin tracking a new file, you use the command git add. To begin tracking the Project file, you can run this:



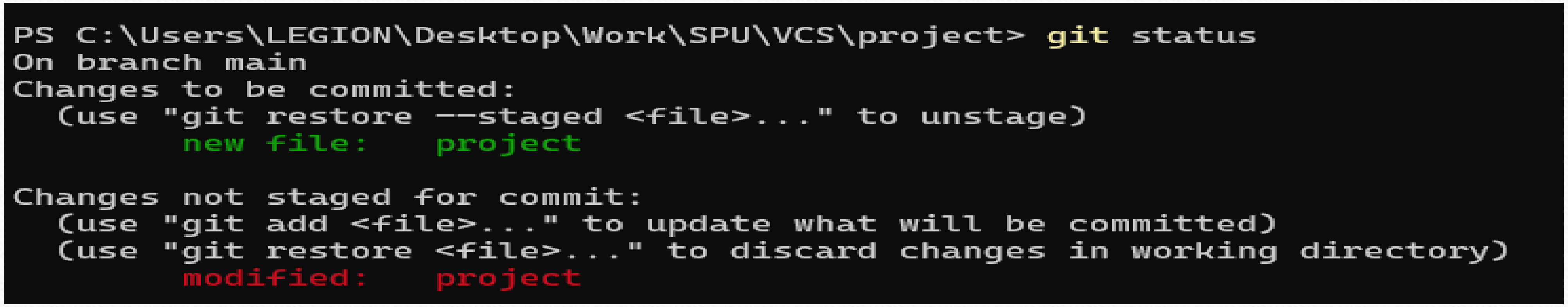
```
PS C:\Users\LEGION\Desktop\Work\SPU\VCS\project> git add project
PS C:\Users\LEGION\Desktop\Work\SPU\VCS\project>
```



```
PS C:\Users\LEGION\Desktop\Work\SPU\VCS\project> git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   project

PS C:\Users\LEGION\Desktop\Work\SPU\VCS\project>
```

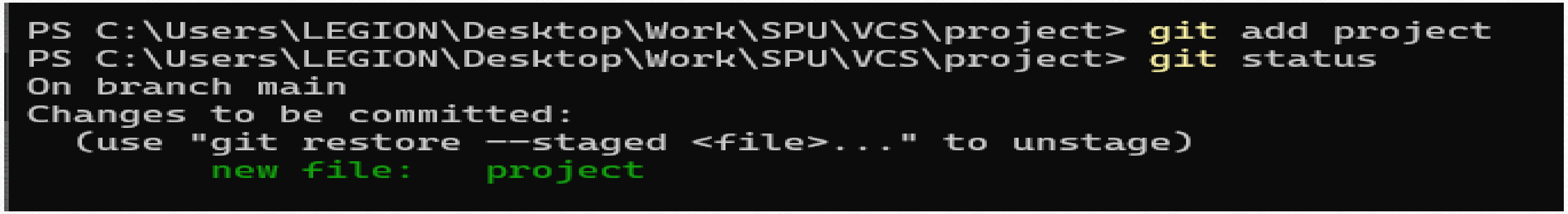
Staging Modified Files

- Let's change a file that was already tracked. If you change a previously tracked file called project.txt and then run your git status command again, you get something that looks like this:



```
PS C:\Users\LEGION\Desktop\Work\SPU\VCS\project> git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   project

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:   project
```

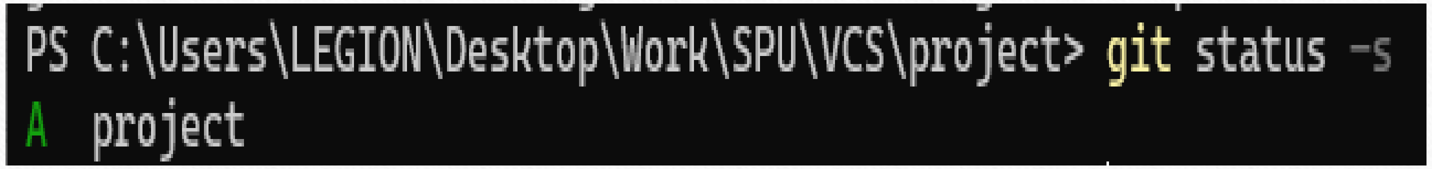
- The project.txt file appears under a section named "Changes not staged for commit" — which means that a file that is tracked has been modified in the working directory but not yet staged.
- To stage it, you run the git add command.
- git add is a multipurpose command — you use it to begin tracking new files, to stage files, and to do other things.



```
PS C:\Users\LEGION\Desktop\Work\SPU\VCS\project> git add project
PS C:\Users\LEGION\Desktop\Work\SPU\VCS\project> git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   project
```

Short Status

- While the git status output is pretty comprehensive, it’s also quite wordy.
- Git also has a short status flag so you can see your changes in a more compact way.
- If you run git status -s or git status --short you get a far more simplified output from the command:

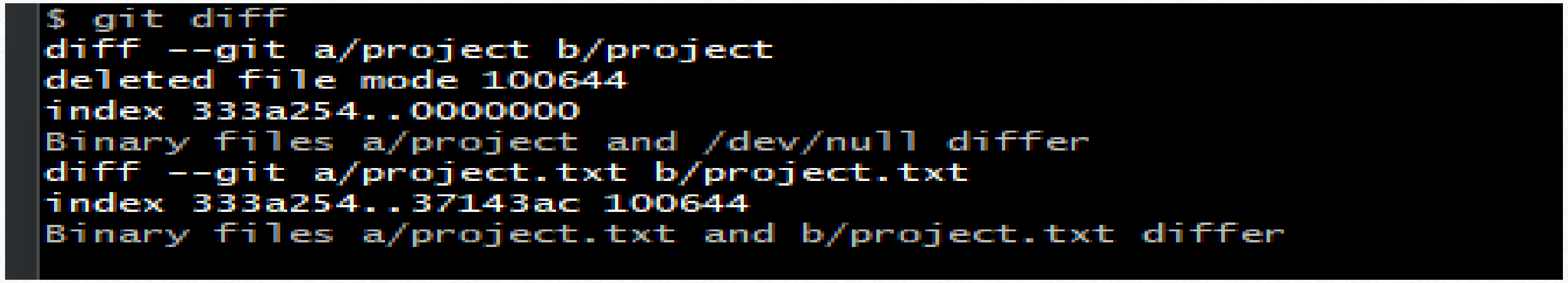


```
PS C:\Users\LEGION\Desktop\Work\SPU\VCS\project> git status -s
A  project
```

- New files that aren’t tracked have a ?? next to them
- new files that have been added to the staging area have an A
- modified files have an M and so on.
- There are two columns to the output — the lefthand column indicates the status of the staging area and the right-hand column indicates the status of the working tree.

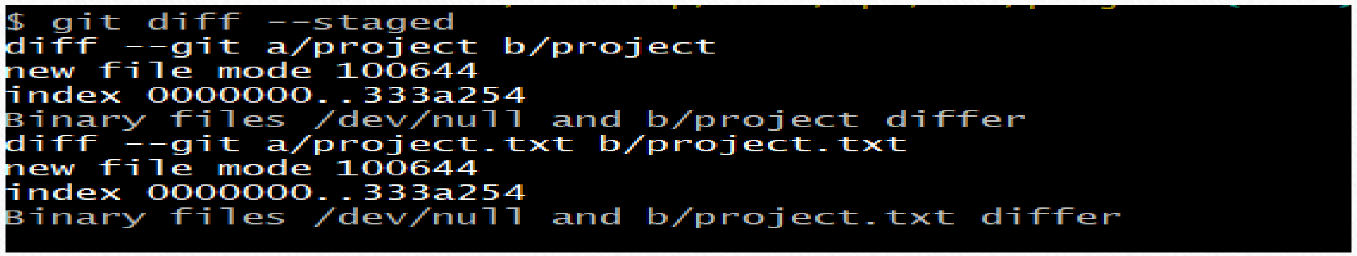
Viewing Your Staged and Unstaged Changes

- If the git status command is too unclear for you — you want to know exactly what you changed, not just which files were changed — you can use the git diff command.



```
$ git diff
diff --git a/project b/project
deleted file mode 100644
index 333a254..0000000
Binary files a/project and /dev/null differ
diff --git a/project.txt b/project.txt
index 333a254..37143ac 100644
Binary files a/project.txt and b/project.txt differ
```

- If you want to see what you've staged that will go into your next commit, you can use git diff --staged. This command compares your staged changes to your last commit:

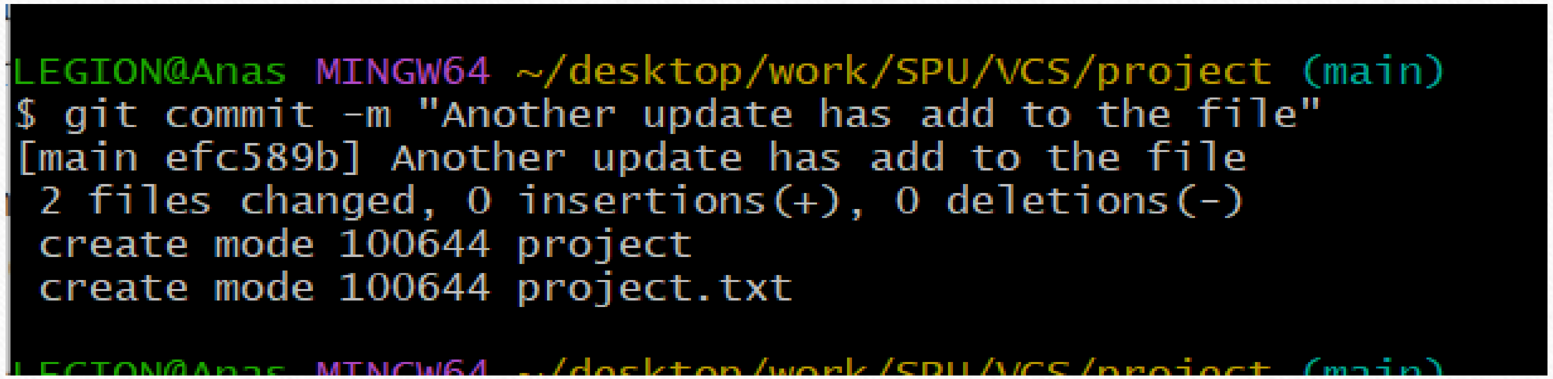


```
$ git diff --staged
diff --git a/project b/project
new file mode 100644
index 0000000..333a254
Binary files /dev/null and b/project differ
diff --git a/project.txt b/project.txt
new file mode 100644
index 0000000..333a254
Binary files /dev/null and b/project.txt differ
```

- It's important to note that git diff by itself doesn't show all changes made since your last commit — only changes that are still unstaged. **If you've staged all of your changes, git diff will give you no output**

Committing Your Changes

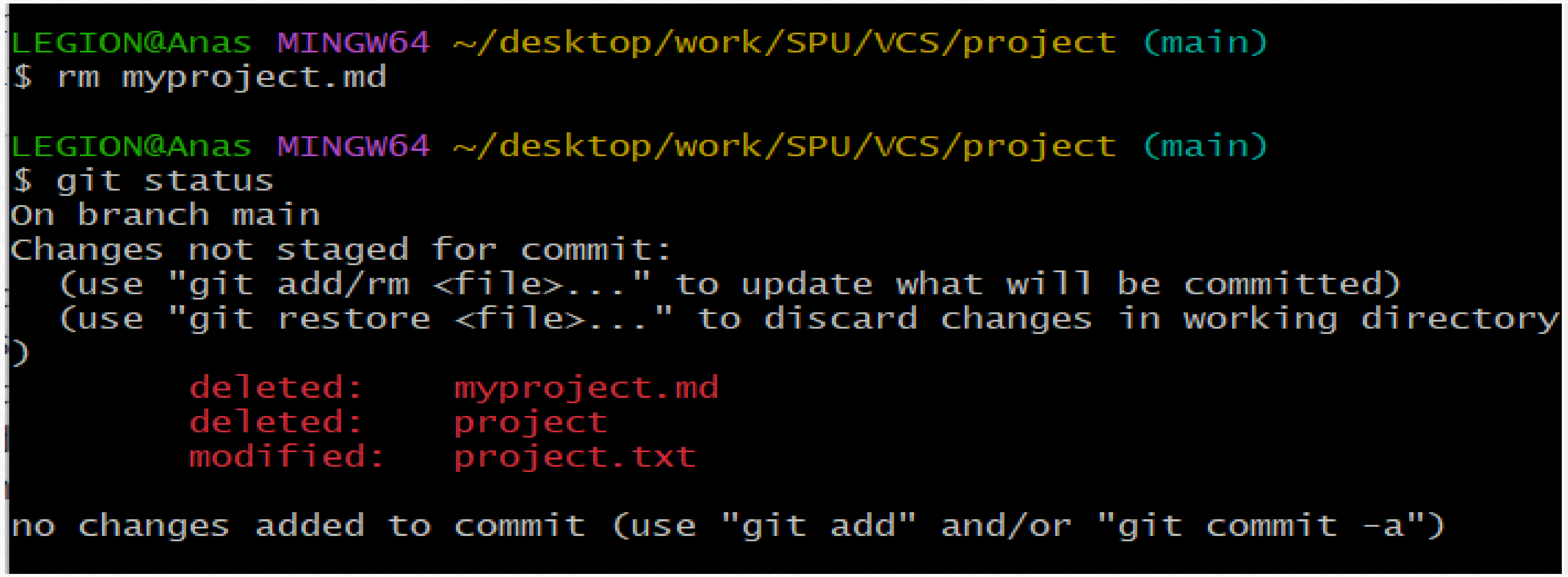
- Now that your staging area is set up the way you want it, you can commit your changes.
- Remember that anything that is still unstaged — any files you have created or modified that you haven’t run git add on since you edited them — won’t go into this commit.
- They will stay as modified files on your disk. In this case, let’s say that the last time you ran git status, you saw that everything was staged, so you’re ready to commit your changes.
- The simplest way to commit is to type git commit



```
LEGION@Anas MINGW64 ~/desktop/work/SPU/VCS/project (main)
$ git commit -m "Another update has add to the file"
[main efc589b] Another update has add to the file
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 project
 create mode 100644 project.txt

LEGION@Anas MINGW64 ~/desktop/work/SPU/VCS/project (main)
```

Removing Files

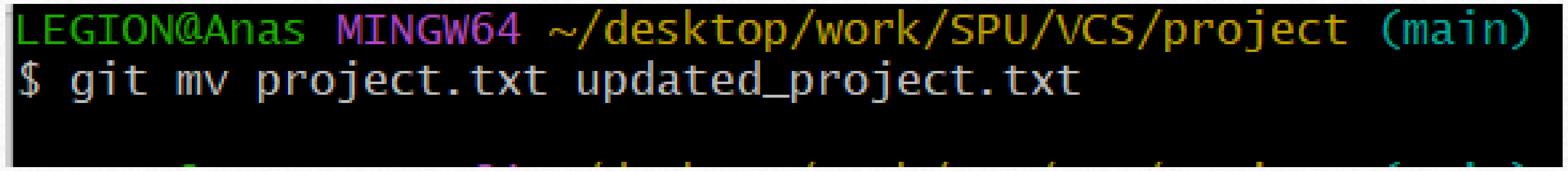
- To remove a file from Git, you have to remove it from your tracked files (more accurately, remove it from your staging area) and then commit.
- The git rm command does that, and also removes the file from your working directory so you don't see it as an untracked file the next time around.
- If you simply remove the file from your working directory, it shows up under the "Changes not staged for commit" (that is, unstaged) area of your git status output



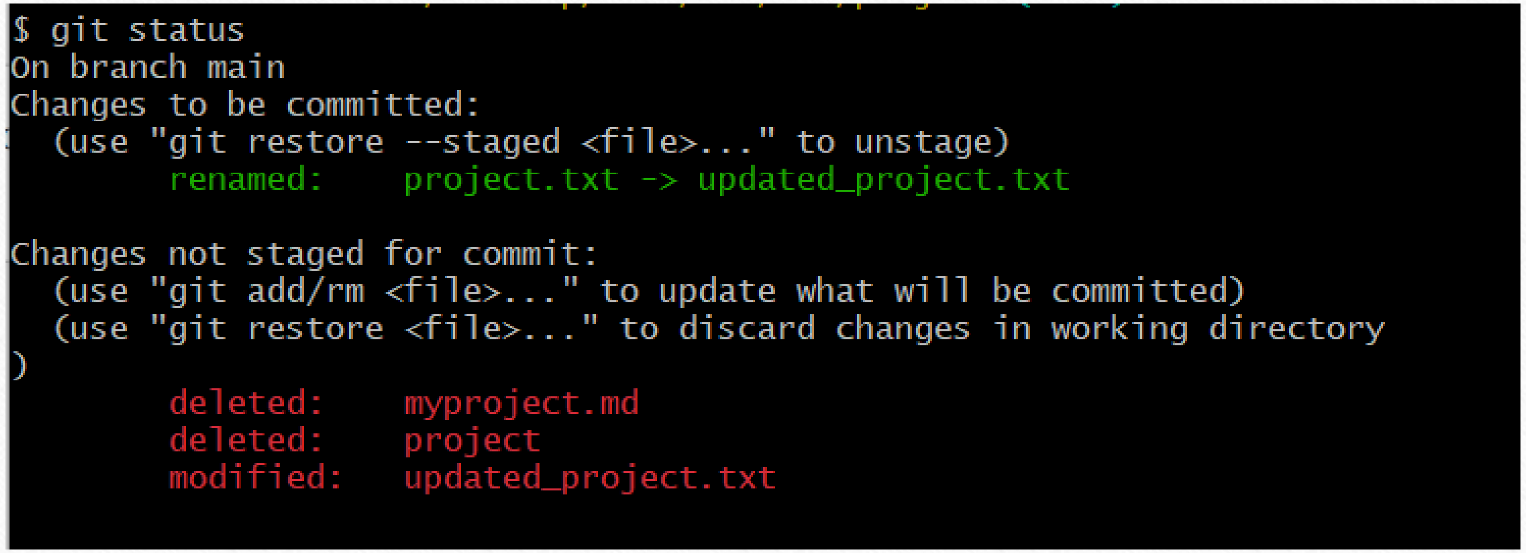
```
LEGION@Anas MINGW64 ~/desktop/work/SPU/VCS/project (main)
$ rm myproject.md

LEGION@Anas MINGW64 ~/desktop/work/SPU/VCS/project (main)
$ git status
On branch main
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:    myproject.md
        deleted:    project
        modified:   project.txt

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

Moving Files

- Unlike many other VCSs, Git doesn’t explicitly track file movement.
- If you rename a file in Git, no metadata is stored in Git that tells it you renamed the file.
- Thus it’s a bit confusing that Git has a mv command. If you want to rename a file in Git, you can run something like

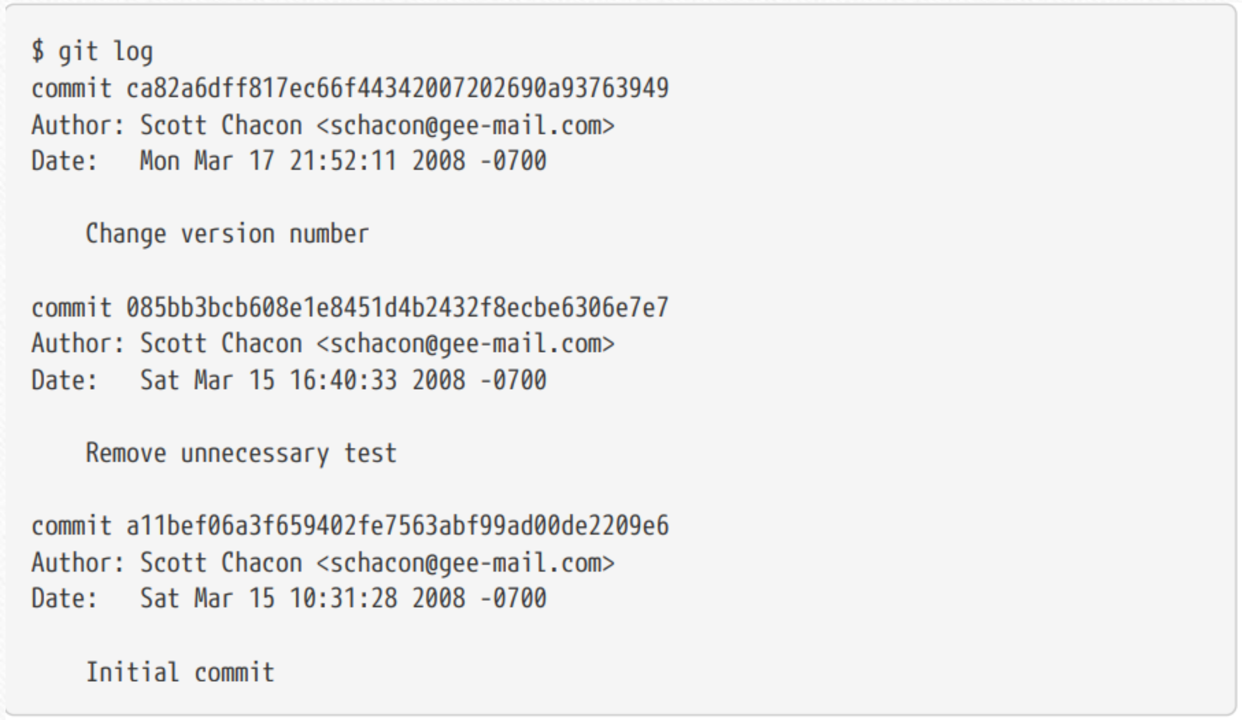


```
LEGION@Anas MINGW64 ~/desktop/work/SPU/VCS/project (main)
$ git mv project.txt updated_project.txt
```



```
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    project.txt -> updated_project.txt

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:    myproject.md
        deleted:    project
        modified:   updated_project.txt
```

Viewing the Commit History



```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    Change version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    Remove unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

    Initial commit
```