Mastering Version Control and Collaboration Tools

Git GitHub
GitLab SVN

Mednour.khl@gmail.com

(+216) 94 038 928

Hello everyone,

I recently completed a Udemy bootcamp on Git, GitHub, GitLab, and SVN.

In this presentation, I'll share key commands and scenarios, complete with comments and explanations, to help you master version control.

Git is essential for developers, providing powerful tools for managing code.

This course emphasized the command line, ensuring we can handle any situation with confidence.

Let's dive into the details!

Version Control System

1. Keeps **Versions** of every file/directory

2. **Documents changes** with descriptive message

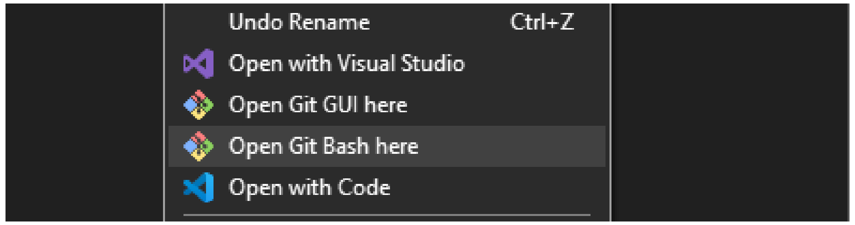
3. Displaying **differences**

Scenario 1: Git directory

1)Install git on your computer

2)Create new folder

2) In folder wright click and open git bash here



Scenario 2: configure Git user information

1) **$ git config --global user.name “user name” --** configure git with user name

2) **$ git config -- global user.email “email” --** configure git with email

3) **$ git config --list   --** Verify Git Configuration

Scenario 3: Initialize and Manage a Git Project

1) **Cd <directory> --**Navigate to the desired directory

2) **Git init --**initialize git project in this directory

3) **Git status --**get the the status of the files in directory

4) **Touch hello world.py --** Create a new file named hello_world.py

5) **Ls** -- list all file under this directory

6) Print some code in the file: (any line like print(“hello”))

7) **Git status**

8)**Git add helloworld.py --**Add hello_world.py to the staging area

9)**git add .** --you can add all changes: without specific file

10) **Git status --**Check Git Status Once More and Verify that the file is staged

11)**Git commit –m “first commit using git” --**Commit the staged changes with a message:This command takes a snapshot of the project at this point in time.

11) **Git log --** View the log of all commits

Scenario4: Show Differences Between Files

1) **touch greetings.py** -- create new file

2) add some code and **git add .** And **commit** like previous steps

3) add code in hello world file

4) **git status** **--**file modified: helloworld.py

5) **git add .** **--** add all untracked files

6) add some code in greetingsfile.py

7)**Git diff** **--** Show the differences between the working directory file and the tracked file

Scenario5:remove file completely from git repository

1)**Git rm <filename> <filename2**> -- Remove Files Completely from Git Repository: This command stages the removal of the specified files.

2)**Git rm --r <directory>**   -- This command stages the removal of the specified directory and all its files**.**

3)**Git ls files**   --This command displays all files currently tracked by Git.

Scenario 6: Tagging and Versioning

- **tagging**: you have history and you need to tag and to specify important points during this history.

- **Tag** given points in the history of the repository to tag different points, like very important point.

1)**Git tag version1** --tag specific point

2)**Git tag** -- get the list of tag in repository

3)**Git tag --list** --list all tags using a different command

4)**Git tag --l “v1*”** --get the list of tag which contain start with v1

5)**Git tag --a v2.2 –m “descriptive message”** -- create tag with descriptive message

6)**Git show <tag-name>** --get all details with tag

7)**Git tag –delete v1.1** --delete tag

Scenario7: Unstaging Files

1) **git restore --staged file2.py** -- taking file from the staging area and bring it back to the working directory: making it no longer part of the next commit

2)**Git status** -- the file is not under the staged area anymore

3) **git reset HEAD file1.py** -- This command also unstages the file and brings it back to the working directory

Scenario8 : Unmodify and Revert File Changes

- modify for example file1.py : add   comment for example

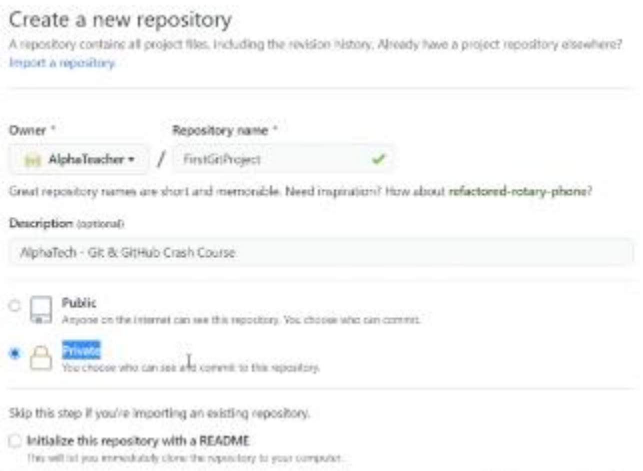
1)   Git status -- This command will show that file1.py has been modified.

2) **Git restore file1.py** -- This command undoes all changes made to file1.py and restores it to the most recent version from the last commit. Any local changes will be lost.

- **Note**: The git restore command is powerful and should be used with caution. When you revert a modified file in the working directory, any local changes you made to this file will be gone, and Git will replace the modified file with the most recent version from the last commit.

Scenario 9 : Remote version control system

Most used "Git Repository Hosting Service"
GitHub
GitLab
SourceForge
Bitbucket
So chances are high that you've heard about these guys, these guy, at least once.


Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * AlphaTeacher / Repository name * FirstGitProject

Great repository names are short and memorable. Need inspiration? How about refactored-rotary-phone?

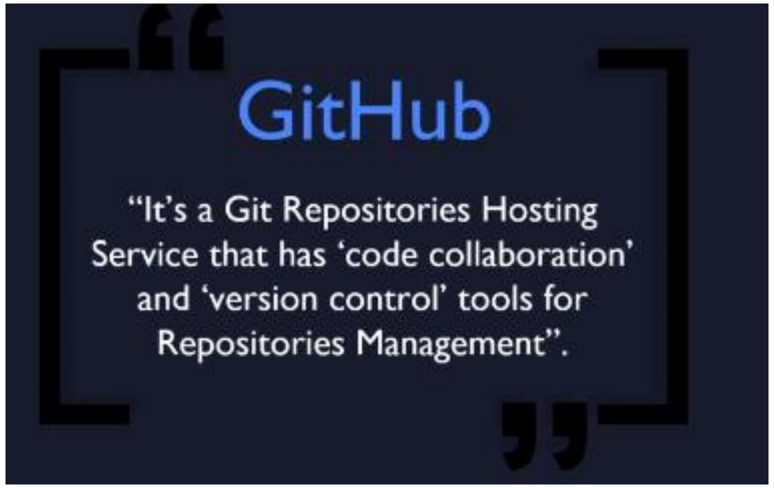
Description (optional) AlphaTech - Git & GitHub Crash Course

( ) Public
Anyone on the internet can see this repository. You choose who can commit.

[✓ Private]
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

[ ] Initialize this repository with a README
This will let you immediately clone the repository to your computer.


GitHub

"It's a Git Repositories Hosting Service that has 'code collaboration' and 'version control' tools for Repositories Management".

Create git hub account + create new repository

To push your local repository to a remote repository, you can use the git push command. Here is a step-by-step guide to do this:

1) **cd /path/to/your/local/repository** -- Navigate to your local repository

2) **Git remote add origin** https://github.com/your-username/your-repository.git --Add the remote repository

3) **git push -u origin main** --Push your local changes to the remote repository and Replace main with the name of your branch if it's different.

Scenario 10 : SSH process

1) Git remote –v   --lists all the remote repositories associated with your local repository

2) $ cd .ssh   --Navigate to the .ssh Directory : users/your local name/.ssh

3)$ ssh-keygen -t rsa -b 4096 -C "mail address" -- generate ssh key in this directory

-t rsa: Specifies the type of key to create, in this case, RSA.

-b 4096: Specifies the number of bits in the key, 4096 bits in this case.

-C “mailAdress": Adds a comment to the key, which is usually an email address.

4) enter file which to save key : testCourse   --choose name like you want

5) enter you preferred passphrase

6) **$ eval $(ssh-agent -s)** --This command starts the SSH agent in the background, which is necessary for managing your SSH keys

7) **$ ssh-add testCourse** --Add Your SSH Key to the Agent

8**) $ clip <testCourse.pub** --copy content of file in keyboard

9) Navigate to Github :account/settings/ssh and GPG keys

10) Click button new ssh key and add content here (specify title or anything)

11)**Cd our repository** -- return navigate to our local repository

12) **$ git push origin master** --take our repository and push him to git hub

13) Check git hub and refresh and it excepted my repository here

Scenario 11 : gitignore

1)Create a file named **.gitignore** at the root of your repository

2) **# .gitignore**

Build/

***.txt**

--the following script writting in **.gitignore file** is to ignore all files .txt extension in the build/ directory.

3) **git status** -- Check the status of untracked files

4) **Git add .gitignore** --Add changes to the staging area

5) **git commit -m “msg“** --Commit the changes

6) **git push origin master** --Push the changes to the remote repository

Scenario 12 : cloning a GitHub project

1) Copy the repository URL

2) **Cd /path…** --Open Git Bash on your local machine and navigate to the directory where you want to clone the repository.

3) **git clone <repository-url>** -- clone the repository to your local machine.

4)**ls** --List the files in the cloned project

5)Modify some code in some files

6) **Git status** --see the changes you have made

7)**Add .** -- Add changes to the staging area

8) **Git commit –m ”msg”** --Commit the changes

9) **Git push origin master** --Push the changes to the remote repository

10) Refresh your GitHub account

11) Create a pull request: Click the "Create pull request" button to propose your changes to the owner of the original repository

Scenario 13 : Creating and working with Branches

1) **Git branch “branche name”** --create new branch

2) **Git branch** --show all branches in repository

3) **git checkout “name of new branch”** -- Switch from the current branch to this new branch

4) Git checkout –b “bran” -- create and switch to branch same time

5)Modify and add in some files where I’m current in new branch

6)**Git add .**

7) **Git commit “msg”**

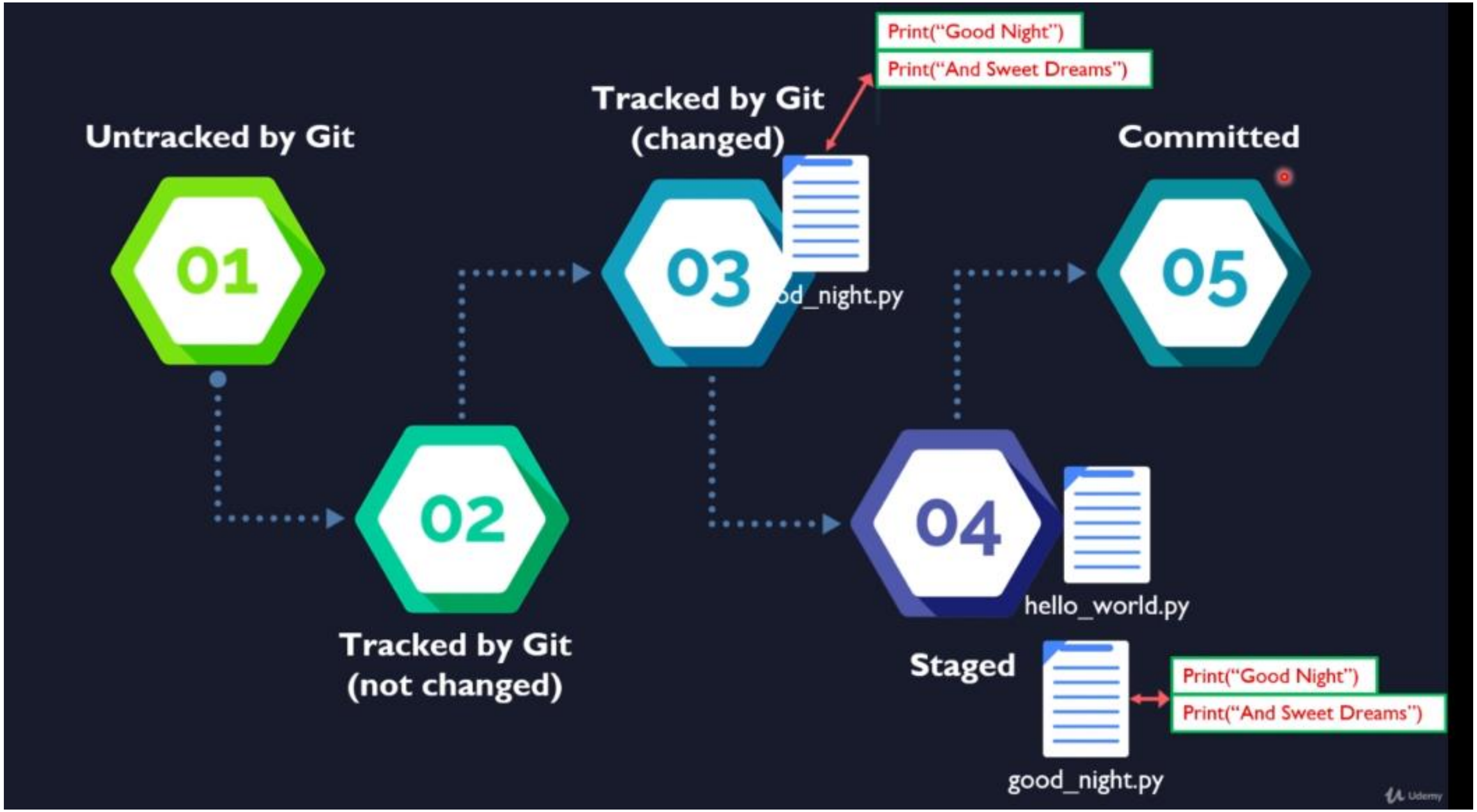
8) **Git push origin “branch name”** --Push the changes to the remote repository

9) In GitHub, check that the files added or modified in the new branch are not shown in the master branch.

10) **Git checkout master**

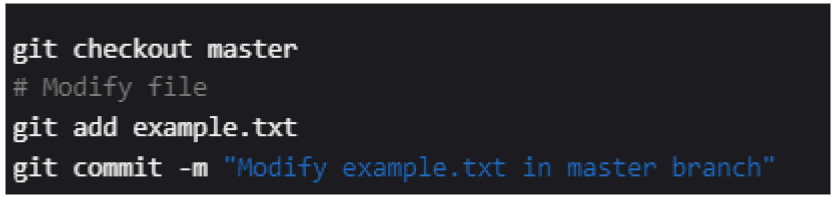
11) **Git merge <branchname2>** --merge master+new branch

12)**Git branch --d branch name** --remove branch


Print("Good Night")
Print("And Sweet Dreams")
Untracked by Git
Tracked by Git (changed)
Committed
01
03
d_night.py
05
02
04
hello_world.py
Tracked by Git (not changed)
Staged
Print("Good Night")
Print("And Sweet Dreams")
good_night.py
Udemy

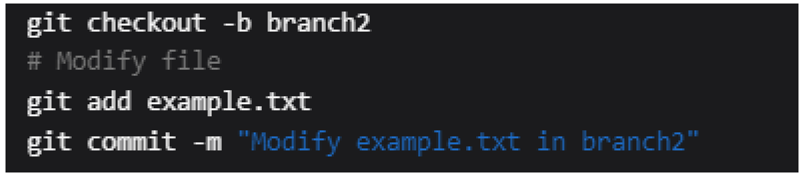
Scenario 14 : Handling a Merge conflict in Git

1)Modify and commit in master



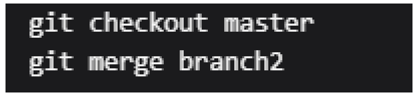
```
git checkout master
# Modify file
git add example.txt
git commit -m "Modify example.txt in master branch"
```

2)Modify and commit in branch2



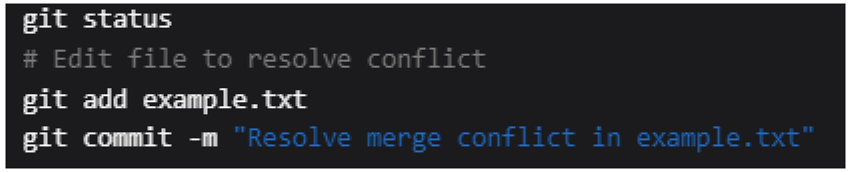
```
git checkout -b branch2
# Modify file
git add example.txt
git commit -m "Modify example.txt in branch2"
```

3)Merge branch2 into master



```
git checkout master
git merge branch2
```

4)Check status and resolve conflict



```
git status
# Edit file to resolve conflict
git add example.txt
git commit -m "Resolve merge conflict in example.txt"
```

Scenario 15 : Short Hand -Alias

1)Short Status

```
git status --short
# Displays the status of the working directory in a short format.
# M: file has been modified
# ?: file is not tracked by git
```

2)Git Documentation Path

```
# Path to Git documentation on Windows
C:\Program Files\Git\mingw64\share\doc\git-doc
```

3)Log with Patch

```
git log -p
# Shows the commit logs along with the differences introduced in each commit.
```

4)Last 3 Commits with Differences

```
git log -p -3
# Shows the last 3 commits along with the differences introduced in each commit.
```

5)Log Since a Specific Time

```
git log --since=3.months
git log --since=3.days
git log --since=3.weeks
# Shows what happened in the repository during the specified time period.
```

6)Log with Statistics

```
git log --stat
# Displays the commit logs along with the statistics of changes (insertions/deletions) for
each commit.
```

7)Alias for Status

```
git config --global alias.st status
# Creates an alias 'st' for the 'status' command, making it shorter and easier to use.
```

8)Alias for Unstage

```
git config --global alias.unstage 'reset HEAD --'
# Creates an alias 'unstage' to replace 'reset HEAD --', making it easier to unstage changes.
```

9)Last Commit

```
git log -1 HEAD
# Shows the details of the last commit.
```

10)Alias for Last Commit

```
git config --global alias.last 'log -1 HEAD'
# Creates an alias 'last' to show the details of the last commit.
```

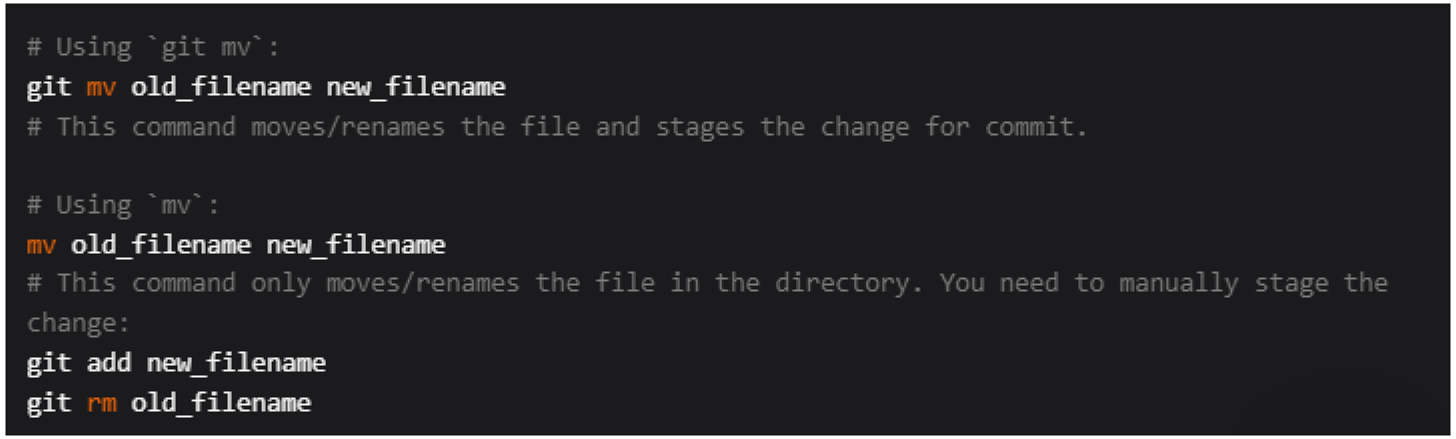
11)Using the Alias

```
git last
# Uses the alias 'last' to show the details of the last commit.
```

12)Remove File from Directory

```
rm bootcamp.py
# Removes the file 'bootcamp.py' from the directory but not from the Git index.
# To add the deletion to the staging area:
git rm bootcamp.py
```

13)Difference Between git mv and mv



```
# Using `git mv`:
git mv old_filename new_filename
# This command moves/renames the file and stages the change for commit.

# Using `mv`:
mv old_filename new_filename
# This command only moves/renames the file in the directory. You need to manually stage the
change:
git add new_filename
git rm old_filename
```