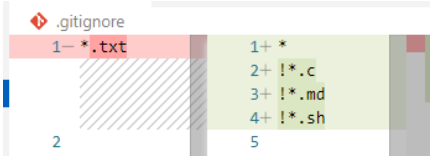
# Exercise 2

Use bash and git commands

1. Make a directory, ***exercise-2***, on the desktop of your computer and open it in visual studio code.
    a. mkdir exercise-2; cd exercise-2; code .
2. Open the terminal of visual studio code and create a repository and run **git status** and **git log**
    a. git init
    b. git status
    c. git log
3. Create **README.md** file of your repository and write ***# Exercise 2*** to it
    a. echo “# Exercise 2” > README.md
4. Create a **.gitignore** file and ignore all **.txt** files.
    a. echo “*.txt” > .gitignore
5. Add the changes to the **staging** area of the repository.
    a. git add .
6. Make a **commit** with message “**Initial commit**”
    a. git commit -m “Initial commit”
7. Run **git status**, **git log** and **git log --oneline**.

```
$ git log --oneline
02a0a7f (HEAD -> master) Initial commit
```

    a. git status
    b. git log
    c. git log --oneline
8. Remove **README.md** from the repository using **git rm**.
    a. git rm --force README.md
9. Run **git status** and then **unstage** the change using **git restore**.
    a. git status
    b. git restore --staged README.md
10. Run **git status** and **discard** changes using **git restore**.
    a. git status
    b. git restore README.md
11. Instead of all **.txt** files, ignore all files whose extensions are not **.c**, **.md** and **.sh**




    a. printf '*\n!*.c\n!*.md\n!*.sh\n' > .gitignore
12. Run **git status** and add **.gitignore** to the **staging** area.
    a. git status
    b. git add .gitignore
13. Create two files, **main.c** and **run.sh,** in the root of the repo.
    a. touch main.c run.sh

14. Write the below code using **printf** to **run.sh**

```
clear && gcc main.c -o main && ./main
```

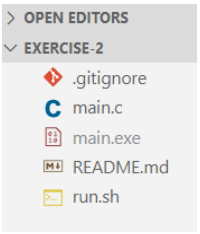
    a. printf “clear && gcc main.c -o main && ./main” > run.sh

15. Write the below code using **printf** to **main.c**

```
#include <stdio.h>\n\nint main(void) \n{\n\treturn 0;\n}
```

    a. printf “#include <stdio.h>\n\nint main(void) \n{\n\treturn 0;\n}” > main.c

16. Run **git status** and add the changes to the staging area
    a. git status
    b. git add .
17. **Commit** changes with message “**Created main.c and run.sh**”
    a. git commit -m “Created main.c and run.sh”
18. Run **sh run.sh** in the terminal. Has the executable file, **main.exe** or **main**, been ignored?




```
$ git log --oneline
e258890 (HEAD -> master) Created main.c and run.sh
cbda3a7 Initial commit
```

    a. sh run.sh
    b. Yes

19. Run **git log**. Change the message of the last commit to “**First commit**”
    a. git log
    b. git commit --amend -m “First commit”
20. Add a **note**, *The program and its compilation*, to the last commit.
    a. git notes add -m “The program and its compilation”
21. Create a **branch**, *feature-branch*, based on the *master branch*
    a. git branch feature-branch
22. Create another **branch**, *print-1-6-1*, based on the *master branch*
    a. git branch print-1-6-1
23. Get the list of branches using **git branch**
    a. git branch
24. **Rename** the *feature-branch* branch to ***print-1-3-1***
    a. git branch -m feature-branch print-1-3-1
25. Get the list of branches using **git branch**

```
$ git branch
* master
  print-1-3-1
  print-1-6-1
```

26. **Switch** to ***print-1-3-1*** branch and run **git log --oneline**
    a. git switch print-1-3-1
    b. git log --oneline

27. In **main.c** make a program using a **for loop** to **print from 1 to 3** to the **output**

```c
#include <stdio.h>

int main(void)
{
    for (int i = 1; i < 4; i++)
    {
        printf("%d  ", i);
    }
    printf("\n");

    return 0;
}
```

28. Run **sh run.sh** in the **terminal** and be sure your program works.
    a. sh run.sh
29. Run **git status** and **add** the changes to the **staging** area.
    a. git status
    b. git add .
30. **Commit** the changes with the message "**print from 1 to 3**". Run **git log --oneline**.

```
$ git log --oneline
e92faea (HEAD -> print-1-3-1) print from 1 to 3
dc6934a (print-1-6-1, master) First commit
02a0a7f Initial commit
```

    a. git commit -m "print from 1 to 3"
    b. git log --oneline
31. Now **switch to master** and run **git log --oneline**.
    a. git switch master
    b. git log --oneline
32. What is the **difference** between ***master*** and ***print-1-3-1***?
    a. On master we have 2 commits. But on print-1-3-1 there are 3 commits
33. **Switch** to ***print-1-3-1*** and run **git status**.
    a. git switch print-1-3-1 && git status
34. In **main.c** make a **for loop** after the previous loop to **print from 2 to 1** to the **output**

```c
#include <stdio.h>

int main(void)
{
    for (int i = 1; i < 4; i++)
    {
        printf("%d  ", i);
    }

    for (int i = 2; i > 0; i--)
    {
        printf("%d  ", i);
    }

    printf("\n");

    return 0;
}
```

35. Run **sh run.sh** in the **terminal** and be sure your program works.
    a. sh run.sh
36. **Commit** the changes with message "**print from 2 to 1**" and run **git log --oneline**

```
$ git log --oneline
147d7f1 (HEAD -> print-1-3-1) print from 2 to 1
e92faea print from 1 to 3
dc6934a (print-1-6-1, master) First commit
02a0a7f Initial commit
```

    a. git add .
    b. git commit -m "print form 2 to 1"
    c. git log --oneline
37. In the last loop change your code in order to print from **12 to 1** to the output.
38. Run **sh run.sh** in the **terminal** and be sure your program works.
    a. sh run.sh
39. **Commit** the changes with message "**print from 12 to 1**" and run **git log --oneline**
    a. git add . && git commit -m "print from 12 to 1" && git log --oneline
40. **Revert** the **last commit** with the message "**Revert print from 12 to 1**". Run **git log --oneline**

```
$ git log --oneline
38a8cf9 (HEAD -> print-1-3-1) Revert "print from 12 to 1"
db1ab4f print from 12 to 1
147d7f1 print from 2 to 1
e92faea print from 1 to 3
dc6934a (print-1-6-1, master) First commit
02a0a7f Initial commit
```

    a. git revert HEAD
    b. Edit the message in the editor
    c. git log --oneline
41. Then **hard reset** the branch to the commit with message "**print from 2 to 1**"
    a. git reset --hard 147d7f1
    b. 147d7f1 is the commit id of the commit. In your repo it is something else.
42. **Merge** *print-1-3-1* branch into master with the message "**print from 1 to 3 to 1**"
    a. git switch master
    b. git merge print-1-3-1 -m "print from 1 to 3 to 1"
43. Run **git log --oneline.** Delete branch *print-1-3-1* and run **git branch**

```
$ git log --oneline
147d7f1 (HEAD -> master) print from 2 to 1
e92faea print from 1 to 3
dc6934a (print-1-6-1) First commit
02a0a7f Initial commit
```

```
$ git branch
* master
  print-1-6-1
```

    a. git log --oneline
    b. git branch -d print-1-3-1
    c. git branch
44. **Switch** to *print-1-6-1* branch and run **git log --oneline**
    a. git switch print-1-6-1
    b. git log --oneline

45. Use **git cherry-pick** and add the commit with message “**print from 12 to 1**” to the branch

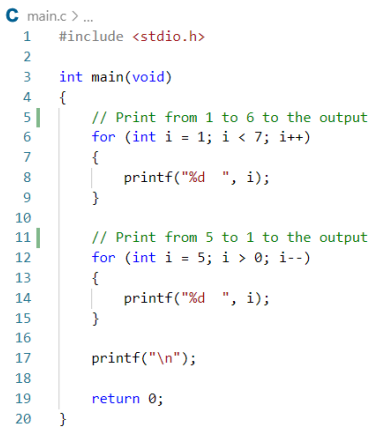

   a. git cherry-pick db1ab4f
   b. db1ab4f is the commit ID of the commit. In your repo it is something else.

46. Is there a conflict? solve it in a way that the program counts from 1 to 6 and then 5 to 1.
   a. Yes. Change the loops and save the changes

47. Run **git status** and **add** the changes to the **staging** area
   a. git status
   b. git add .

48. **Commit** changes with message “**print from 1 to 6 to 1**”
   a. git commit -m “print from 1 to 6 to 1” **or** git cherry-pick --continue

49. Add a comment, ***// Print from 1 to 6 to the output***, to the first loop in main.c
50. Add a comment, ***// Print from 5 to 1 to the output***, to the second loop in main.c




51. **Try** to switch to master. Is it possible? Use **git stash** to save changes and then switch to master.
   a. It is not possible to switch to master. Because there are some changes in the working directory
   b. git stash
   c. git switch master

52. Run **git log --oneline** and then **switch** to *print-1-6-1*
   a. git log --oneline
   b. git switch print-1-6-1

53. Use **git stash list** to get the list of stashes. Then restore the stash using **git stash pop**
   a. git stash list
   b. git stash pop

54. **Add** changes to the **staging** area and then **commit** changes with message "**Commented the code**"
    a. git add . && git commit -m "Commented the code"
55. **Merge** *print-1-6-1* into *master* with message "**count and print 1-6-1**".
    a. git switch master
    b. git merge print-1-6-1 -m "count and print 1-6-1"
56. Is there a conflict? solve it and use git merge --continue to complete the merge. Run **git log --oneline**.

```
$ git log --oneline
5c3f9ab (HEAD -> master) count and print 1-6-1
453d2c6 (print-1-6-1) commented the code
124c478 print from 1 to 6 to 1
147d7f1 print from 2 to 1
e92faea print from 1 to 3
dc6934a First commit
02a0a7f Initial commit
```

    a. Yes.
    b. git merge --continue
    c. git log --oneline
57. **Delete** *print-1-6-1* and run **git branch** and **git log --decorate --graph --oneline**

```
$  git log --decorate --graph --oneline
*   5c3f9ab (HEAD -> master) count and print 1-6-1
|\
| * 453d2c6 commented the code
| * 124c478 print from 1 to 6 to 1
* | 147d7f1 print from 2 to 1
* | e92faea print from 1 to 3
|/
* dc6934a First commit
* 02a0a7f Initial commit
```

    a. git branch -d print-1-6-1
    b. git branch
    c. git log --decorate --graph --oneline
58. Add a **tag**, **v1.0**, to the **last commit** and run **git tag** to list the tags
    a. git tag v1.0
    b. git tag
59. Run **git log**, **git log --oneline** and then add a message, ***The first release***, to the tag
    a. git log
    b. git log --oneline
    c. git tag -af v1.0 -m "The first release"
60. Run **git tag** and **git tag -n** to show the tag and then **delete** the tag.

```
$ git tag -n
v1.0            The first release
```

    a. git tag
    b. git tag -n
    c. git tag -d v1.0