سلام به همگی.

امیدوارم که همگی تندرست و شادمان باشین.

من دو ویدیو برای کار با Git و GitHub آماده کردم که اصول کار را یاد می‌دهد. هدف این است که با کمک Git فایل‌هایمان را ردیابی کنیم و بتوانیم تغییراتش را ثبت و رصد کنیم. با کمک GitHub هم فایل‌هایمان را با همکارانمان یا با همه‌ی مردم به اشتراک می‌گذاریم.

نیاز به نصب نرم‌افزار Git دارید که از اینجا انجام میشه؛ Git برای ویندوز، مک و لینوکس در دسترس هست. همانطور که در ویدیو هم گفتم، باید در GitHub هم ثبت‌نام کنید. پس از ثبت‌نام کردن (که نیاز به VPN دارد)، آیدی‌تان را به من بدهید.

پس از انجام این‌دوکار می‌توانید سراغ فیلم‌ها بروید که در لینک‌های زیر هستند:

1. آشنایی مقدماتی با Git و GitHub
2. دستورالعمل کارکردن مشترک با GitHub

در نهایت این خاطرجمعی را بدهم که شما تنها نیاز به یاد گرفتن ۴ دستور ساده در Git Bash دارید و اصلا روند کار پیچیده نیست. مشکلی هم بود می‌توانید با من در تماس باشید.

در صفحه‌ی بعدی یادداشت‌هایی که برای نوشتن Git برداشته‌ام، آورده‌ام.

Git Cheat Sheet!
https://education.github.com/git-cheat-sheet-education.pdf

Git book, free and full of details.
https://git-scm.com/book/en/v2

This video was great and give a concise and comprehensive tutorial to start using Git and GitHub.

Let's learn git a little bit!

Git is a Source Control Management (SCM), to manage the progress of a project. One of its remarking features is that it logs update of content, including its time, name of the modifier, changes done on the content. So, it makes collaboration efficient. For me, who is working alone, it helps me keep track of my stuff (like website, etc. ...).

When you installed git, first configure with identifying yourself to it:

```
>> git config --global user.name "Hossein"
```

```
>> git config --global user.email "Email Address"
```

You can change the name of the branch that you're going to create by:

```
>> git config --global init.default branch "New branch name"
```

After that you are ready to initialize a git repository.

For getting help about git, there are several methods:

```
>> git *** -h
```

Instead of *** write the command which you want to know about more.

```
>> git help ***
```

This gives a detailed manual on this command, installed with git software.

At any stage, use

```
>> clear
```

to clean the space, note that it's not git command, but a cmd (bash) command.

Also, you can use

```
>> cd "directory address"
```

to go to specific directory, or run git in the same window that you are currently with *:run git bash:* option.

To install your repository:

```
>> git init
```

It creates a hidden file named ".git" in the same directory to keep your things tracked.

To look at the current status of your repo:

```
>> git status
```

It gives a detailed status of untracked files and staged files.

To track files, first stage them by:

```
>> git add "file.exe"

>> file add "/dir/"      # Adds the whole directory.

>> git add --all

>> git add .

>> git add -A
```

The last three codes add all the file to staging process.

To remove files from the staging,

```
>> git rm --cahced "file.txt"
```

How to ignore specific files from github? Make a file in the initial directory of github and name it ".gitignore". Codes written in this file dictates git to ignore what files.

```
>> /dir1/dir2/file.exe      # ignores a specific file

>> /dir1/      #ignores the whole subdirectory

>> *.docx      #ignores all docx files in all folders in any depth.

>> /dir1/**/*.txt      # ignores all txt files in dir1, in any depth.

>> !/dir1/temp/*.txt      # exempts temp subdirectory of above rule,
so all txt files in this folder will be considered by git.
```

You can find other useful code in ChatGPT.

Commit: this act means to take a snapshot of the condition of your project, and write it in a history book (log) of git. You can only restore to previous conditions by tracking them like this.

```
>> git commit -m "message."
```

Actually, git has three main environments:

1- working environment --> 2. Staging --> 3- Committing (new working env.)

And to record the changes it git, we have to go through all this stages.

To move files into staging,

```
>> git add
```

commands will be used that explained earlier.

To see the changes happened recently:

```
>> git diff
```

which gives a detailed view of the changes occurred.

To unstage the staged files, and moving them back into working environment,

```
>> git restore --staged "file.txt"
```

You can bypass stage and commit process by this snippet:

```
>> git commit -a -m "message."
```

-a means ALL files must be unstaged.

You can remove files from the directory by:

```
>> git rm "file.txt"
```

And restore them:

```
>> git restore "file.txt"
```

Pretty like the same way you delete and restore files in windows Recycle bin.

And rename by

```
>> git mv "oldname.txt" "newname.txt"
```

you have to commit after changing file name of deleting, since they are serious acts.

History of git is in:

```
>> git log
```

Which contains names, ids, dates, commit messages, etc. The order is from the last commitment to the first one.

One line log:

```
>> git log --oneline
```

To change the last commit message:

```
>> git commit -m "new message" --amend
```

Changing older messages is more dangerous and needs to work with base:

1. `>> git rebase -I HEAD~n`

moves the head of the writer into n-th commitment message, it goes into a interactive environment to edit it.

2. Change pick to edit, then change your message.
3. Quit by Esc+ type :wq + Enter

A very very detailed log in:

```
>> git log -p
```

To restore a specific commit:

```
>> git reset --hard #id_of_log
```

To enter the base environment:

```
>>git rebase -i --root
```

It's a great idea to work with branches. Suppose you work in GUI of software, colleagues work on bugs, etc. Each one of you makes a branch and modifies the source code, and then merges them to have a final version. That is the natural flow and essence of coding!

To make a new branch:

```
>> git branch name
```

To look at all branches' name:

```
>> git branch
```

To change the active branch:

```
>> git switch name
```

Make a branch and switch to it rapidly:

```
>>git switch -c BranchName
```

It's pretty remarkable that by changing the branch, file content of windows also changes and you can no longer access other branches' files, unless you switch to them.

Any change in any branch have to be committed, and branches work independently, and changing one of them doesn't modify others.

How to merge them? Suppose you are in the main branch.

```
>> git merge -m "Message" BranchName      # BranchName is the name of branch you want to merge with main branch.
```

Or simply (without messages)

```
>> git merge BranchName
```

To delete branches:

```
>> git branch -d BranchName
```

Conflicts may occur, suppose that you and your friend have change same place in two parallel branches, which have to be considered? When you merge, the git will show you

conflict message, and the name of the file which must be modified. In the file, there are things like:

<<<<<<<<<< HEAD

Your friends source code

==========

Your source code

==========

You choose one of these codes and delete unnecessary part. Then commit in the main branch. You'll see that there's no conflict and merging goes on completely.

To connect to remote host:

```
>> git remote add [alias] "Address"
```

alias is a name that we use later to address that host.

To push the changes to host:

```
>> git push -u alias branch
```

Decide which branch push to the host at alias address.

To force push use:

```
>> git push --force alias branch
```

To push all

```
>> git push --all
```

To pull from remote:

```
>> git fetch alias
>> git pull alias branch
```

This snippet pulls from alias repository into your local branch.

To clone a project from web:

```
>> git clone "address"
```

To hard restore your local repository from remote:

```
>> git reset --hard alias/branch
```

To see difference of your local files with a remote file:

```
>> git fetch alias
>> git diff alias/branch
```

Tips:

1. To see all files forked in git: `>> git ls-files`
2. In a specific directory: `>> git ls-files "/dir/../"`
3. To cancel pushing (for bulky ones) use Ctrl+C when pushing.
4. To see all remote connections in a git session `>> git remote -v`
5. To unstage all files in rep: `>> git rm -e --cached .`
