0. Register on the GitHub and create a remote git repo
```
$ mkdir hometask
$ cd hometask
$ git init
$ git remote add origin ssh://myserver/my_repo
```

1. Stashing in git (git stash)
a) $ git stash
b) $ git status
c) $ git stash pop
d) $ git status

2. Understand merging strategies (fast-forward, recursive etc)
a) Fast forward merge can be performed when there is a direct linear path from the source branch to the target branch. In fast-forward merge, git simply moves the source branch pointer to the target branch pointer without creating an extra merge commit.

b) In Recursive merge, after you branch and make some commits, there are some new original commits on the ‘master‘. So, when it’s time to merge, git recurses over the branch and creates a new merge commit. The merge commit continues to have two parents.

c) Octopus Merge strategy resolves cases with more than two heads but refuses to do a complex merge that needs manual resolution. It is primarily meant to be used for bundling topic branch heads together. This is the default merge strategy when pulling or merging more than one branch.

3. Read about configuring git (local and global)
a)
```
$ mkdir repo1
$ mkdir repo2
$ cd repo1
$ git init
$ cd repo2
$ git init
```

b)
```
$ cd repo1
$ git config user.name "localname"
$ git config user.email "localemail"
```

c)
```
$ git config --global user.name "globalname"
$ git config --global user.email "globalemail"
```

d)
```
$ cd repo1
$ git config user.name
> localname

$ git config user.email
> localemail

$ git config --global user.name
> globalname

$ git config --global user.email
> globalemail
```

4. Read about how to connect to a remote repository on GitHub by https and ssh
a)
```
$ mkdir testrepo
$ cd testrepo
```

   $ git init
b) $ git add . (Adds the files in the local repository and stages the for commit.)
   $ git commit -m "First commit" (Commits the tracked changes and prepares them to be pushed to a remote repository.)
c) $ git remote add origin <REMOTE_URL> (Setting up a new remote.)
   $ git remote -v (Verifying the new remote URL.)
d) $ git push -u origin main (Pushing the changes in your local repository up to the remote repository you specified as the origin.)
   Also we can use $ git pull to get the latest updates from our remote repository.

5. Read about popular git-based workflows
In the Git flow development model, you have one main development branch with strict access to it. It's often called the develop branch.
Developers create feature branches from this main branch and work on them. Once they are done, they create pull requests. In pull requests, other developers comment on changes and may have discussions, often quite lengthy ones.
It's a good idea to use Git flow in the following cases:
- If you're working with an open-source project.
- If you have a lot of junior developers in your team.
- If your product is already established.

In the trunk-based development model, all developers work in a single branch with open access to it. Often it's simply the master branch. They commit code to it and run it. It's super simple.

In some cases, they create short-lived feature branches. Once code on their branch compiles and passes all tests, they merge it straight to master. It ensures that development is truly continuous and prevents developers from creating merge conflicts that are difficult to resolve.

It's a good idea to use trunk-based development model in the following cases:
- If your porject isn't too big and complicated.
- If it's necessary to work quickly.
- If your team mainly consists of senior developers.

6. Understand difference between merging and rebasing branches
a) Let's look at some cases when rebase would be reasonable and effective:
- If you're developing locally. You have not shared your work with anyone else. At this point , you should prefer rebasing over merging to keep history tidy. If you've got your personal fork of the repository and that is not shared with other developers, you're safe to rebase even after you've pushed to your fork.
- Your code is ready for review. You create a pull request, others are reviewing your work and are potentially fetching it into their fork for local review. At this point you should not rebase your work. You should create 'rework' commits and update your feature branch. This helps with traceability in the pull request, and prevents the accidental history breakage.

Talking about the advantages of using this method we can list the following:
- Your code history will remain flat and readable. Clean, clear commit messages are as much part of the documentatiton of your code base as code comments, comments on your issue tracker etc.
- Manipulating a single commit is easy (e.g. reverting them).

b) Now let's talk about the tradeoff rebasing has against merging. To start with I would like to mention pull requests, because you can't see what minor changes someone made if they rebased. Also rebasing can be dangerous! Rewriting history of shared branches is prone to team work breakage. Also using rebase means more work. Using rebase to keep your feature branch updated requires that you resolve similar conflicts again and again. Also sometimes you will have to force push the changes.

7. Understand when conflicts occur and what is conflicts resolution in git
a) Done

b) To illustrate how conflicts are working we will do the following. Let's say that we have README.md file created with the following text "This is my second Github repository!" in our remote repository.And creating our first commit "Changed first to second". For the second set of code, we are changing the same README.md file on our local repository. Instead of "second Github repository", we're going to say "third Github repository". We're going to commit this file and set the commit message to "Change first to third". We need to pull our changes onto our local branch to consolidate the changes. When you pull the changes, you'll see an error message "Merging btanch 'origin/master' into 'master'. Fix 1 conflict and then continue."

c, d) The easiest way to resolve a conflict is to change the file on your computer. If you open README.md now, you'll see lines that say this':
<<<<<< HEAD This is my third Github repository====== This is my second Github repository >>>>>> The code between <<<;<<<; HEAD and ======= is the code in our local repository (our code). The code between ===== and >>>>>> is the code from the remote repository.
These two lines of code are conflicting. We need to choose between the "second Github repository" or the "third Github repository". To fix the conflict, you choose the correct line of code. Then you delete everything else. In this case, let's say "third" is the correct version. What you'll do is delete everything else that's incorrect. After fixing all the conflicts we just have to commit our changes.

8. Pull request reviewing and merging
a) $ cd testrepo
   $ git branch newbranch (creating a new branch)
b) $ git checkout newbranch (switching to a new branch)
c) $ git add newfile.txt
   $ git commit -m "added newfile.txt"
d) $ git pull (synchronizing our local repository with remote one)
e, f, g) $ git push origin newbranch (pushing our changes to GitHub).
After that we have to open GitHub and find button "Compare & pull request" and click it, provide necessary details on what you've done.
h) - first, check which branch we are on.
   $ git branch
   - switching to the master branch.
   $ git checkout master
   - add the original repository as an upstream repository
   git remote add upstream [HTTPS]
   - fetch the repository
   $ git fetch upstream
   - merge it
   $ git merge upstream/master
   - push changes to GitHub
   $ git push origin master
   - delete the unnecessary branch
   $ git branch -d test newbranch