Module 3 Working with Remotes

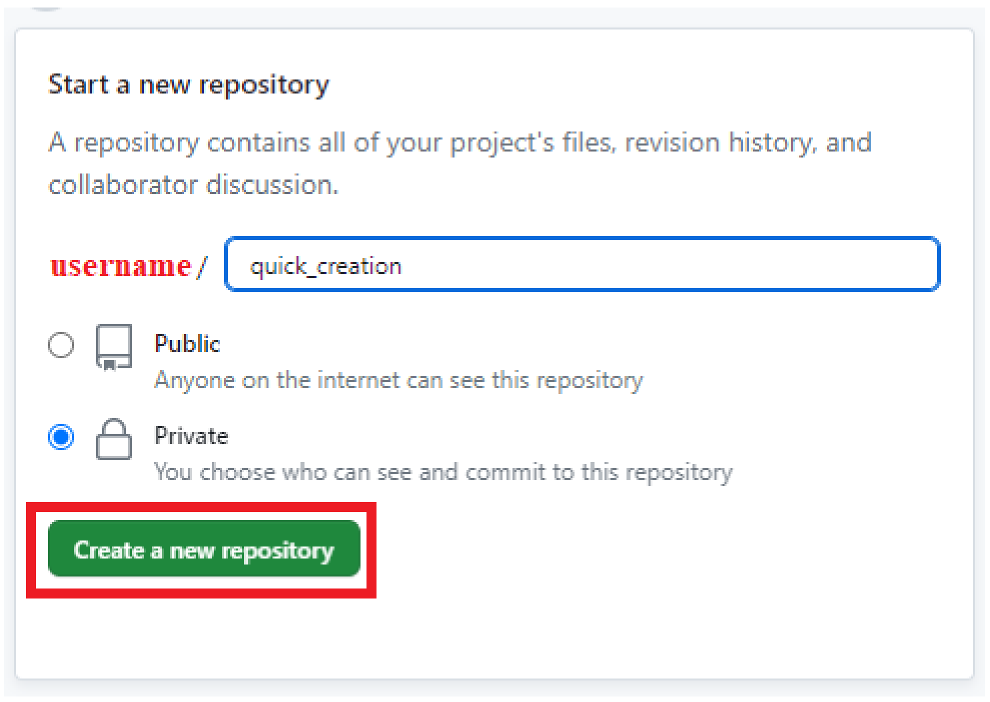
1. Introduction to GitHub

(1) GitHub

- Git is a **Distributed** VCS:
    - Each developer has a *copy* of the whole *repository* on their local machine
    - A *copy* can be used as a ***remote repository*** for all other copies
- **GitHub**
    - *Web-based Git repository* hosting service
    - **Basic function**: share and access *repositories* on the web, copy and clone *repositories* to the local computer
    - **Extra features**: bug tracking, wikis, and task management
    - **Availability**: free access to a *Git* server for *public* and *private* repositories
    - **Security note**: for real configuration and development work, a **secure and private** *Git* shall be used and the authorized people shall be limited.
    - **Sign-up**: GitHub sign-up

(2) Basic interaction with GitHub

- **Create a repository**
    - Quick creation:



Start a new repository

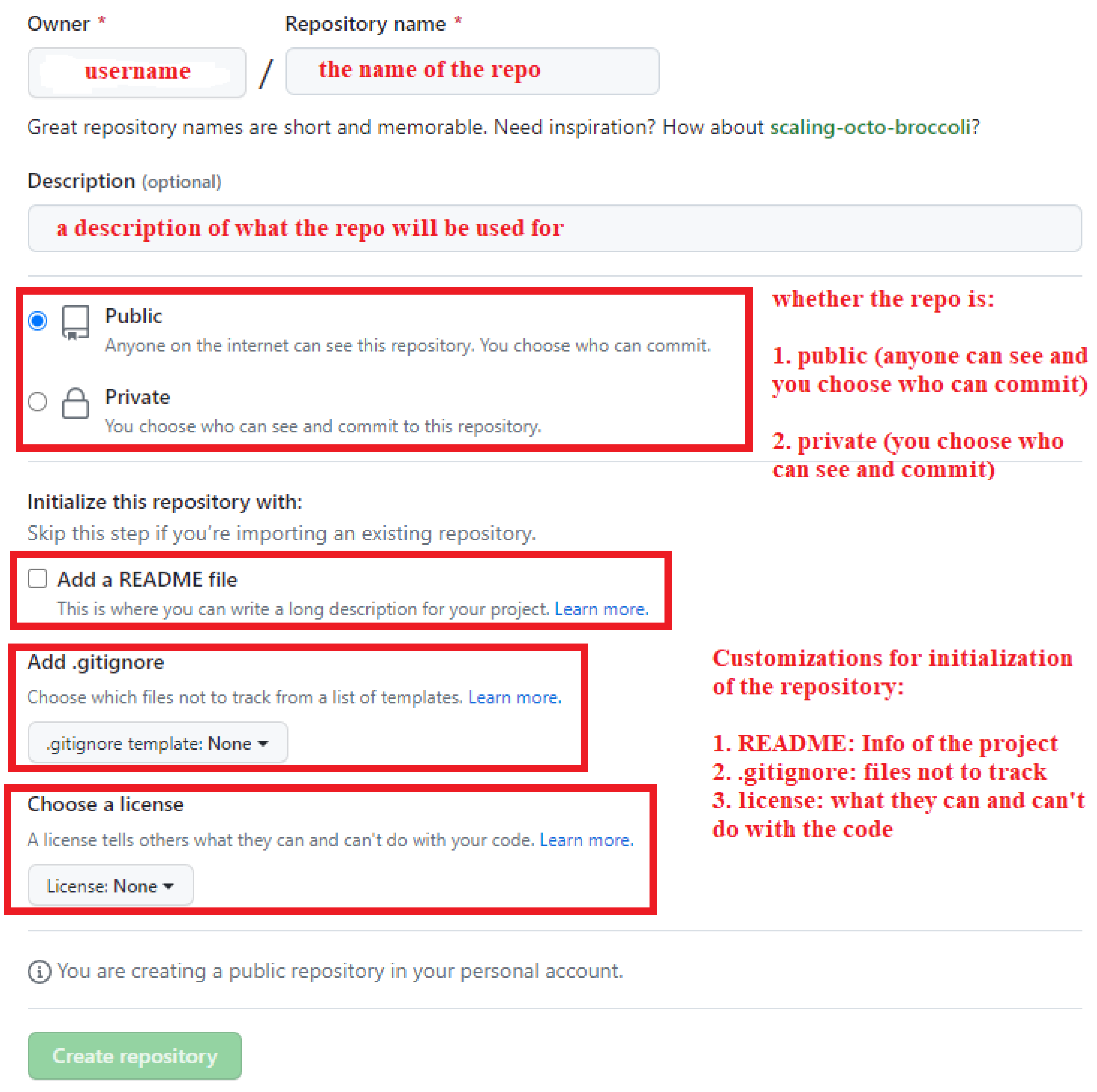
A repository contains all of your project's files, revision history, and collaborator discussion.

username / quick_creation

Public
Anyone on the internet can see this repository

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

    - Customized creation:



Owner * username / Repository name * the name of the repo

Great repository names are short and memorable. Need inspiration? How about scaling-octo-broccoli?

Description (optional)

a description of what the repo will be used for

[✓ 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.

whether the repo is:

1. public (anyone can see and you choose who can commit)

2. private (you choose who can see and commit)

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

Add a README file
This is where you can write a long description for your project. Learn more.

Add .gitignore
Choose which files not to track from a list of templates. Learn more.
.gitignore template: None

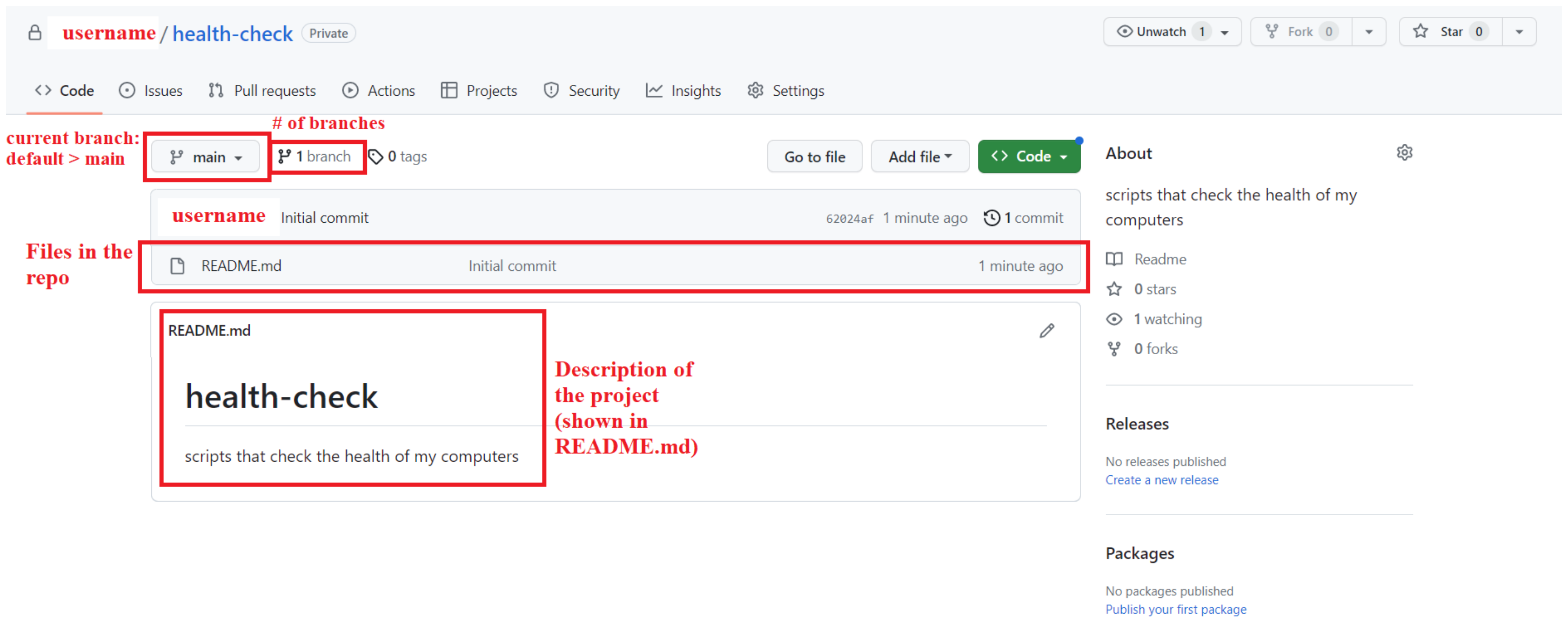
Choose a license
A license tells others what they can and can't do with your code. Learn more.
License: None

Customizations for initialization of the repository:

1. README: Info of the project
2. .gitignore: files not to track
3. license: what they can and can't do with the code

You are creating a public repository in your personal account.

- Created *GitHub Repository*:



username / health-check Private

current branch: default > main

main

# of branches

1 branch

0 tags

username Initial commit 62024af 1 minute ago 1 commit

Files in the repo

README.md Initial commit 1 minute ago

README.md

health-check

scripts that check the health of my computers

Description of the project (shown in README.md)

About

scripts that check the health of my computers

Readme
0 stars
1 watching
0 forks

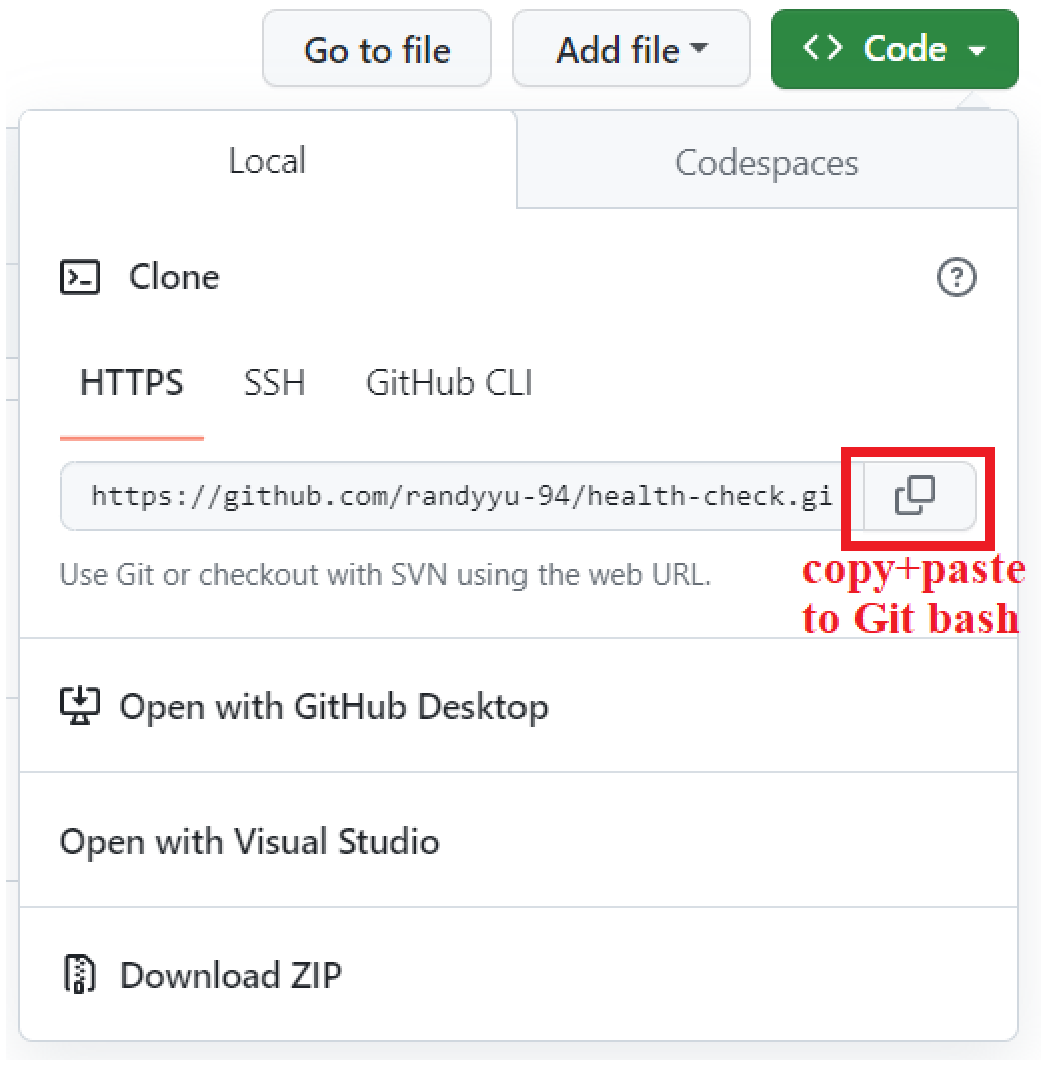
Releases
No releases published
Create a new release

Packages
No packages published
Publish your first package

- **Create a local copy of the repository**

```
git clone https://github.com/[user_name]/[repository_name].git
```

Function: downloads a copy of the *remote repository* from *GitHub* to the local machine



```
https://github.com/randyyu-94/health-check.gi
```

Use Git or checkout with SVN using the web URL.

- **Make modifications on *README.md* file and commit the changes**

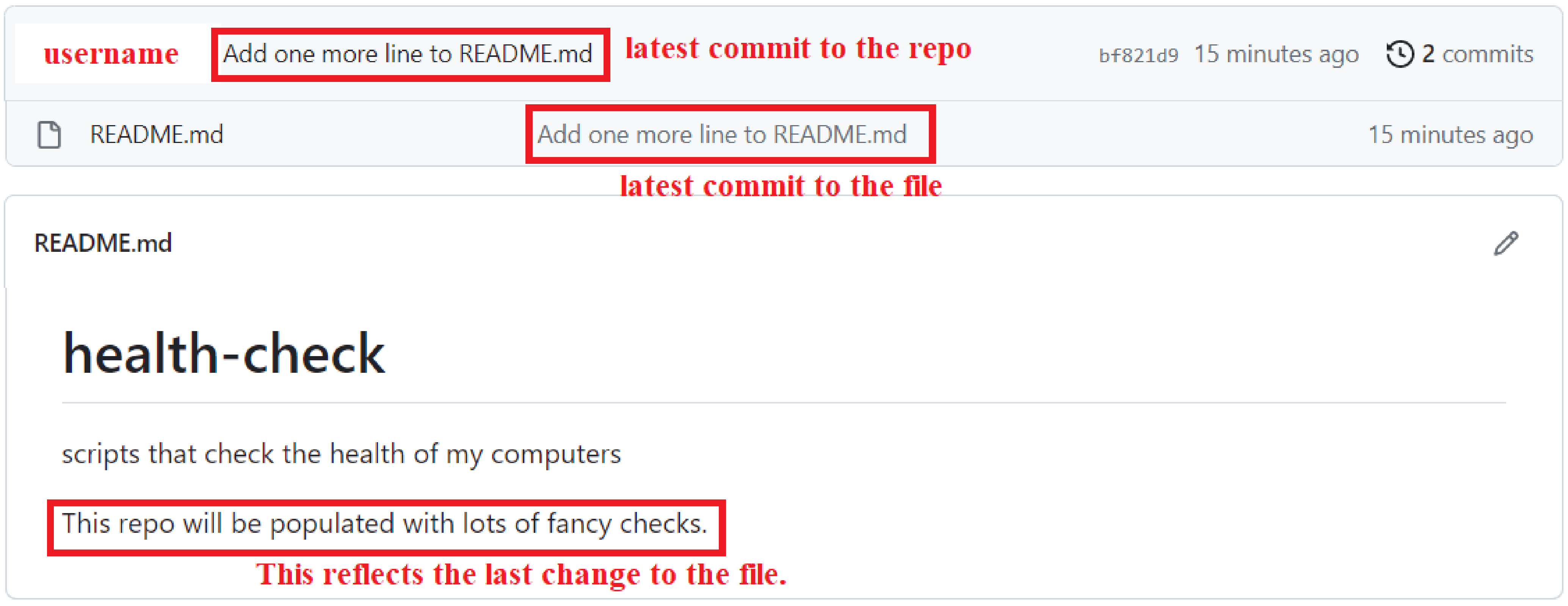
```
git commit -a -m '<commit message>'
```

Function: adds the *modified* file (only *modified*, no *untracked*) to the staging area, and commits the change with the

- **Update the change in GitHub repository**

```
git push
```

Function: gathers all the *snapshots* and sends them to the *remote repository*


username
Add one more line to README.md
latest commit to the repo
bf821d9
15 minutes ago
2 commits
README.md
Add one more line to README.md
15 minutes ago
latest commit to the file
README.md

health-check

scripts that check the health of my computers

This repo will be populated with lots of fancy checks.

This reflects the last change to the file.

- **Enable credential helper (for Unix computer only)**

```
git config --global credential.helper cache
```

Function: enables the *cache* for recording the password and avoiding entering password within 15 minutes

Note: This is not available on **Windows 10** computer. An error will raise if it is executed, like "credential-cache unavailable; no unix socket support".

- **Update the local repository from the remote repository**

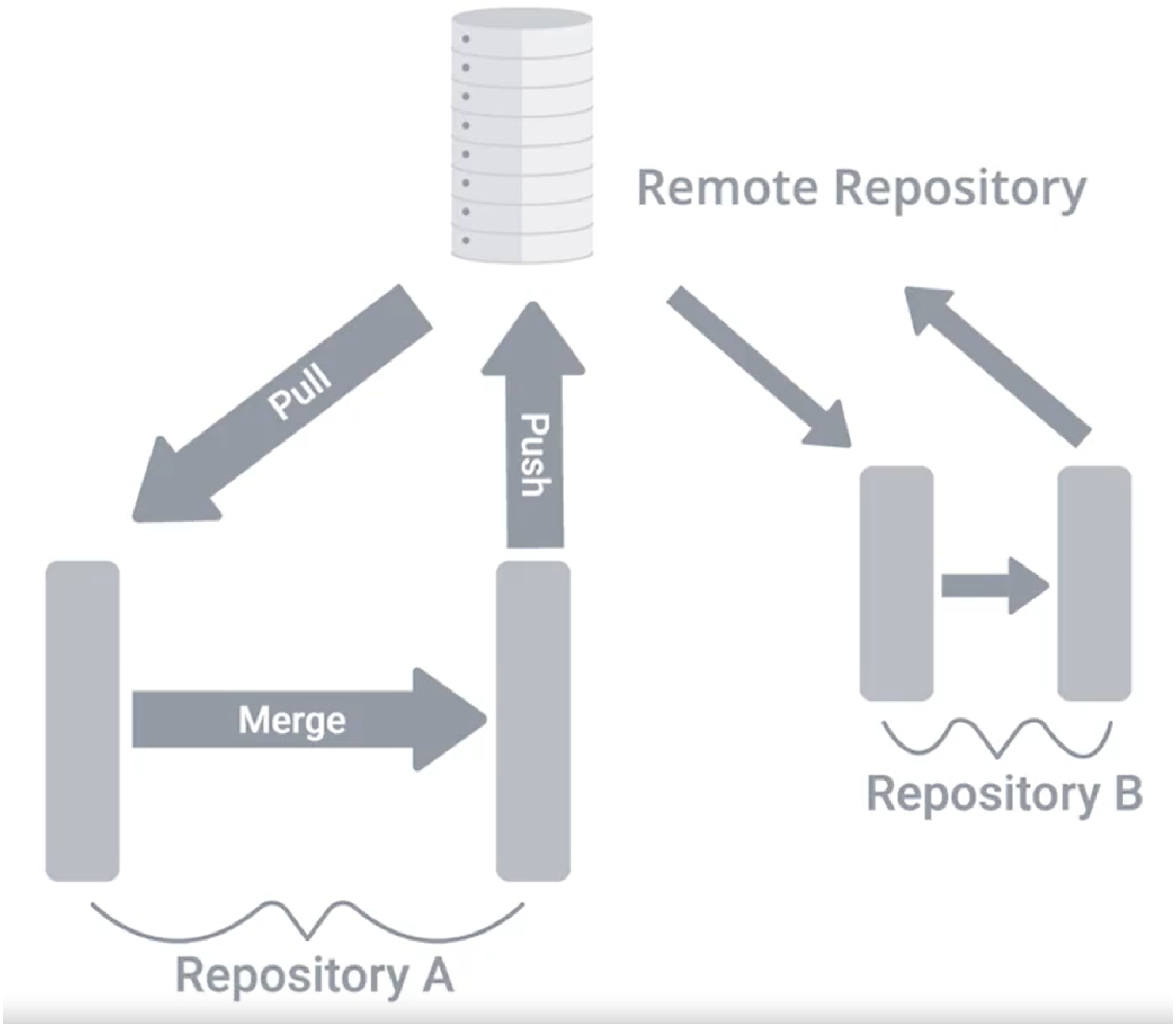
```
git pull
```

Function: retrieves the newest changes from the remote repository to the local repository

2. Using a Remote Repository

(1) Remote repository

- **Remote repository**
    - Allows many developers to contribute to a project from their own workstations making changes to their *local copies* independently
    - Developers can share their updates using *push* command or retrieve others' updates using *pull* command
    - **Common remote repository platforms**: GitHub, BitBucket, GitLab, private Git server
- **Multiple people working on the same remote repository**



- Git keeps different *commits* in *separate branches*
- If someone has updated a *repository* since the last sync: Git tells it's time to do an update
- If you have updated your *local repository*: you may need to fix *merge conflicts* before pulling from or pushing to the *remote repository*
- **Protocols to control remote repositories**: HTTP (pull only), HTTPS & SSH (pull and push)

(2) Working with remotes

```
git remote -v
```

Function: checks the configuration for the remote

- shows the URLs associated with the *origin* remote with both "*fetch*" and "*push*"
  - ***Origin***: the default name of the *remote repository*
- "*fetch*" URL can be HTTP while "*push*" URL can only be HTTPS or SSH

```
git remote show origin
```

Function: shows more information of the remote origin, including: *Fetch URL, Push URL, HEAD branch, Remote branch, Local branch used for "git pull", and local reference used for "git push"*

- **Remote branches**
    - Used for storing copies of the data in the *remote repository*

```
git branch -r
```

Function: check the remote branches (read only)

(3) Fetching new changes

- **When the remote repository is updated ahead of local repository (*commits* in remote repo is not reflected locally)** :
    - Check it by using "*git remote show origin*": it will show "(local out of date)"

```
git fetch
```

Function: copies the commits done in the remote repository to the remote branches

```
git log origin/<branch name>
```

Function: checks the commit history of the remote branch

```
git checkout <local branch>
git merge origin/<branch name>
```

Function: merges the remote branch into our local branch

(4) Updating the local repository

```
git pull
```

Function: fetches the remote copy of the current branch and tries to merge it into the current local branch.

Note: git pull = git fetch + git merge

```
git remote update
```

Function: fetches the contents of all remote branches

3. Solving Conflicts

(1) The pull-merge-push workflow

- When the *remote repository* is modified and not synchronized with the *local repository*:
    - We cannot directly use *git push* to update the *remote repository*
    - Instead, we need to use a *pull-merge-push* workflow to synchronize both *repositories*
- **Work flow on modifying a remote branch**

```
git pull #pull new changes from the remote repo and merge the remote repo with the local repo

#if a merge conflict raises:
git log --graph --oneline #check the log files in a graph shape with one line of each commit
git diff #check the difference in both files being merged
# solve the conflict using text editor
git add <file_name> #add the file to the staging area
git commit #commit the change

git push #push our changes to the remote repo
git status #At any stage, check whether the local repo is synchronized with the remote repo
git log --graph --oneline #re-check the log files to make sure that both commits are merged
```

-
    - **Conflict marker**: >>>

(2) Pushing remote branches

- **Advantages of having multiple branches**:
    - Allows people to work on different tasks (e.g., debugging, testing a new feature, and releasing a new version)
    - Allows releasing more versions out of the same tree for different purposes (e.g., stable version and beta version)
        - Disruptive changes can be tested on the beta version before they are fully released

```
git checkout -b <branch_name>
```

Function: creates a new *branch* with <branch_name> and switch to this *branch*

```
git push -u origin <branch_name>
```

Function: creates a *remote branch* in the *remote repo* with the same <branch_name> as the ***new*** *local branch*, and makes a *git push* from the *local branch* <branch_name> to the *remote branch* <branch_name>

(3) Rebasing changes

```
git checkout <branch1>
git rebase <branch2>
```

Function: rewinds the head of to the head of (an updated version of the ancestor) and replays the commits of that are after the ancestor on top of the rewound head.

- After rebasing, you can merge back to using *git checkout* and *git merge* .

```
git push --delete origin <branch_name>
```

Function: deletes the *remote branch* <branch_name>

```
git branch -d <branch_name>
```

Function: deletes the *local branch* <branch_name>

```
git fetch
git rebase origin/<branch_name>
```

Function: fetches the *commits* from *remote branch* and rebases the *local branch* onto the *remote branch*

- *git rebase* is an alternative command of *git merge*, making the history **linear**

(4) Best practices for collaboration

- **Notes**
  - Always *synchronize local branches* with *remote branches* before starting any local work.
    - i.e., pull before any work
    - Minimizes the chance of conflicts and the need for rebasing
  - Avoid having very large changes that modify a lot of different things (e.g., changing a variable + adding a new feature).
    - Instead: try to make small self-contained changes and push the changes often
  - Suggest using a separate *feature branch* when working on a big change.
    - Allows to make new changes and fix bugs in different branches
  - Regularly merge changes on the *master branch* onto the *feature branch*.
    - Reduces the chance of meeting many *merge conflicts* in the final *merge*
  - Have the latest version in the *main branch* and the stable version on a separate *branch*.
  - **Do not rebase changes** that have been pushed to *remote repos*.
  - Having good *commit messages* is important.