Project work

«Git, GitHub»

The work was done:

Fazylov Vladislav

Kolpakov Artem

Parfenov Nikita

Teacher:

Mingazheva Elena

2022

**Content**

1. About Git ……………………………………………………………..…………
2. About GitHub …………………………………………………………………...
3. What is a Git Repository…...................................................................................
4. What is a Git commit ...........................................................................................
5. What is a Git Remote ...........................................................................................
6. How to create Git Branch ......................................................................................
7. How to delete Git Branch......................................................................................
8. How to rename Git Branch …...............................................................................
9. How Git SSH works ............................................................................................
10. How Git Diff works .............................................................................................
11. What is a Git Checkout ........................................................................................
12. What is a Git Pull ................................................................................................
13. How to Git Merge …............................................................................................
14. How to Git Stash .................................................................................................
15. How to create Git Hooks ......................................................................................
16. Git Squash ...........................................................................................................
17. What is a Pull Request in Git …………..............................................................
18. What is a Git Rebase ...........................................................................................
19. What is Git LFS ...................................................................................................
20. What is a Git submodule …..................................................................................
21. 30 Git commands needed to master the Git command line interface…………..

**About Git**

The Git feature that really makes it stand apart from nearly every other SCM out there is its branching model.

Git allows and encourages you to have multiple local branches that can be entirely independent of each other. The creation, merging, and deletion of those lines of development takes seconds.

This means that you can do things like:

Frictionless Context Switching. Create a branch to try out an idea, commit a few times, switch back to where you branched from, apply a patch, switch back to where you are experimenting, and merge it in.

Role-Based Codelines. Have a branch that always contains only what goes to production, another that you merge work into for testing, and several smaller ones for day to day work.

Feature Based Workflow. Create new branches for each new feature you're working on so you can seamlessly switch back and forth between them, then delete each branch when that feature gets merged into your main line.

Disposable Experimentation. Create a branch to experiment in, realize it's not going to work, and just delete it - abandoning the work—with nobody else ever seeing it (even if you've pushed other branches in the meantime).

Notably, when you push to a remote repository, you do not have to push all of your branches. You can choose to share just one of your branches, a few of them, or all of them. This tends to free people to try new ideas without worrying about having to plan how and when they are going to merge it in or share it with others.

There are ways to accomplish some of this with other systems, but the work involved is much more difficult and error-prone. Git makes this process incredibly easy and it changes the way most developers work when they learn it.

**About GitHub**

**GitHub** is an [Internet hosting service](https://en.wikipedia.org/wiki/Internet_hosting_service) for [software development](https://en.wikipedia.org/wiki/Software_development) and [version control](https://en.wikipedia.org/wiki/Version_control) using [Git](https://en.wikipedia.org/wiki/Git). It provides the [distributed version control](https://en.wikipedia.org/wiki/Distributed_version_control) of Git plus [access control](https://en.wikipedia.org/wiki/Access_control), [bug tracking](https://en.wikipedia.org/wiki/Bug_tracking_system), [software feature](https://en.wikipedia.org/wiki/Software_feature), [task management](https://en.wikipedia.org/wiki/Task_management), [continuous integration](https://en.wikipedia.org/wiki/Continuous_integration), and [wikis](https://en.wikipedia.org/wiki/Wiki) for every project. Headquartered in [California](https://en.wikipedia.org/wiki/California), it has been a subsidiary of [Microsoft](https://en.wikipedia.org/wiki/Microsoft).

It is commonly used to host [open source](https://en.wikipedia.org/wiki/Open_source) software development projects. As of June 2022, GitHub reported having over 83 million developers and more than 200 million [repositories](https://en.wikipedia.org/wiki/Repository_(version_control)) including at least 28 million public repositories. It is the largest [source code](https://en.wikipedia.org/wiki/Source_code) .

The main purpose of GitHub.com is to facilitate the [version control](https://en.wikipedia.org/wiki/Version_control) and [issue tracking](https://en.wikipedia.org/wiki/Issue_tracking_system) aspects of software development. Labels, milestones, responsibility assignment, and a search engine are available for issue tracking. For version control, Git (and by extension GitHub.com) allows [pull requests](https://en.wikipedia.org/wiki/Pull_request) to propose changes to the source code. Users with the ability to review the proposed changes can see a diff of the requested changes and approve them. In Git terminology, this action is called "committing" and one instance of it is a "commit." A history of all commits is kept and can be viewed at a later time.

In addition, GitHub supports the following formats and features:

* Documentation, including automatically rendered [README](https://en.wikipedia.org/wiki/README) files in a variety of [Markdown](https://en.wikipedia.org/wiki/Markdown)-like file formats
* [Wikis](https://en.wikipedia.org/wiki/Wikis)
* GitHub Actions, which allows building [continuous integration](https://en.wikipedia.org/wiki/Continuous_integration) and [continuous deployment](https://en.wikipedia.org/wiki/Continuous_deployment) pipelines for testing, releasing and deploying software without the use of third-party websites/platforms
* Graphs: pulse, contributors, commits, code frequency, punch card, network, members
* Integrations Directory
* Email notifications
* Discussions
* Option to subscribe someone to notifications by [@ mentioning](https://en.wikipedia.org/wiki/Mention_(blogging)) them.
* [Emojis](https://en.wikipedia.org/wiki/Emoji)
* Nested [task-lists](https://en.wikipedia.org/wiki/To_do_list) within files
* Visualization of [geospatial](https://en.wikipedia.org/wiki/Geospatial_analysis) data
* 3D render files that can be previewed using a new integrated STL file viewer that displays the files on a "3D canvas."[[76]](blob:https://en.wikipedia.org/90505ad3-11d7-4b36-911e-eb1ee86a94ec#cite_note-3d-77) The viewer is powered by [WebGL](https://en.wikipedia.org/wiki/WebGL) and [Three.js](https://en.wikipedia.org/wiki/Three.js).
* Photoshop's native PSD format can be previewed and compared to previous versions of the same file.
* PDF document viewer
* Security Alerts of known [Common Vulnerabilities and Exposures](https://en.wikipedia.org/wiki/Common_Vulnerabilities_and_Exposures) in different packages

GitHub's Terms of Service do not require public software projects hosted on GitHub to meet the [Open Source Definition](https://en.wikipedia.org/wiki/Open_Source_Definition). The [terms of service](https://en.wikipedia.org/wiki/Terms_of_service) state, "By setting your repositories to be viewed publicly, you agree to allow others to view and fork your repositories."[[77]](blob:https://en.wikipedia.org/90505ad3-11d7-4b36-911e-eb1ee86a94ec#cite_note-78)

**What is a Git Repository**

A Git repository is the **.git/**  folder inside a project. This repository tracks all changes made to files in your project, building a history over time. Meaning, if you delete the **.git/**  folder, then you delete your project’s history.

**What is a Git commit**

Commits are the main constructive elements of the Git project timeline. They can be viewed as snapshots or checkpoints on the timeline of a Git project. Commits are created using the git commit command, which takes a snapshot of the current state of the project. Commits of Git state snapshots are always executed to the local repository.

**What is a Git remote**

In fact, the git remote command is an interface for managing a list of records about remote connections that are stored in the /.git/config file of the repository. To view the current status of the list of remote connections, use the following commands.

**Git Branch**

Git-branch `git branch` is a Git command for managing branches. Use this label to indicate all issues related to branches, their creation, structure, management and deletion. General information. git branch is a command for managing branches in the Git repository.

**How to create Git Branch**

To create a new branch in Git, you need to run the command (instead of mybranch, specify the name of your new branch): git branch mybranch. After executing the command, a new branch will be created with the name you specified. Note that you are still in the current branch. Switching to a new branch. To switch to a new branch, run the command: git checkout mybranch.

**How to delete a Git Branch**

You can delete a branch using the branch parameter with the addition of the -d flag and specifying the branch name. If you have completed work on a branch and merged it with the main one, you can delete it without losing history. However, if you run the delete command before merging, an error message will appear as a result.

**How to rename a Git Branch**

In order to rename the current branch (the one you are currently in), you need to run the command:

**git branch -m "New name of the current branch".**

If you are in the master branch, and you need to rename the test branch, then run the command:

**git branch -m "test" "head\_branch".**

**How Git SSH works**

**Installation**

If not already installed, then Git can be taken here. Along with it will be a unix-like Git Bash console.

https://github.com/git-for-windows/git/releases/

**Cloning SSH**

Example of a command for cloning via SSH.

**git clone git@github.com:brockgr/csshx.git**

In general, the command for cloning over SSH looks like this:

**git clone git@server.domain:user/reponame.git**

Do not confuse it with HTTPS, which will require authorization via a login password:

**git clone https://github.com/brockgr/csshx.git**

**Creating an SSH key.**

On Windows, you can use both cmd and Git Bash, on \*nix — just in the console. But I don't understand cmd, so I give instructions only for Git Bash & \*nix:

**ssh-keygen -t rsa -C "user.name@mail.domain "**

You can choose passphrase, which increases reliability, but it will need to be entered every time you use it. If you forget, the key is useless for further use.

After executing the command, the public key appears respectively in

**C:\Users\%username%\.ssh\id\_rsa.pub**

**~/.ssh/id\_rsa.pub**

It is the public key that needs to be transferred to the specialist on the other side. (For sure you did, but it's still worth saying about it)

If everything is done correctly, then when trying to connect via ssh, the key will be used automatically.

**If the key is already there**

Then it should be put in c:\Users\%username%\.ssh . If the key name is different from id\_rsa, then you need to create a file c:\Users\%username%\.ssh\config with the following content:

**Host: server.domain**

**IdentityFile path\_and\_key name**

**How Git Diff works**

**Comparison with the last commit**

To output changes in files compared to the last commit, git diff without parameters is used:

***git diff***

The command outputs changes to files that have not yet been added to the index. The comparison takes place with the last commit.

**Comparison with the last commit, including files in the index**

If you have changed any files in your working directory and added one or more of them to the index (using git add), then the git diff command will not show changes in these files. To show changes in files, including files added to the index, the --cached key is used:

***git diff --cached***

**Comparison of commits**

The git diff command allows you to compare two different commits. First you need to determine the hash (ID) of commits that you want to compare. You can use the git log command to list commits and their IDs:

***git log --oneline***

Now let's compare the two commits. To do this, the hash of the first commit is specified as the first argument to the git diff command, and the hash of the second commit is specified as the second argument, for example:

***git diff 4612297 5e356cf***

**Comparison of two branches**

To output all changes between the ends of two branches, it is necessary to specify the names of the branches for git diff:

***git diff branch1 branch2***

**Comparing files between two branches**

To compare specific files between two branches, use the command:

***git diff branch1 branch2 ./myfile.cpp***

Instead of branch1, equal to, branch2, you need to specify the name of the branches, and instead of myfile.cpp the path to the file being compared. Separated by a space, you can add more files for comparison.

**Exclude some files from comparison**

Sometimes you need to execute difference in scoundrels, but exclude one or more files so that the difference in scoundrels command ignores them. To do this, a record of the form ’:(exclude)file\_name’ short or record’ is used:!file\_name’

**Example:**

***git diff -- . ':(exclude)file1.abc' ':(exclude)file2.abc'***

**Or a shorter entry:**

***git diff -- . ':!file1.abc' ':!file2.abc'***

Compare only changes in files (ignore new and deleted files)

To exclude new and deleted files from the comparison, you can use the --differential filter option, which allows you to choose which changes (files) need to be compared.

To compare only changes inside files (ignoring new, deleted, renamed files), the Modified (M) key is used - --differential filter=M:

***git diff --diff-filter=M***

**What is Git Checkout**

In Git, the term checkout means switching between different versions of the target object. The git checkout command works with three different objects: files, commits, and branches. Switching is also usually understood as an action associated with the execution of the git checkout command. As part of the "Undo Changes" topic, we looked at how the git checkout command can be used to view old commits.

**What is Git Pull**

The git pull command is used to extract and download content from a remote repository and immediately update the local repository with this content. Merging remote upstream changes into a local repository is a common workflow task that occurs when working together based on the Git system.

**How to Git Merge**

To merge two or more recent commits into one, use the git rebase command with the -i key (interactive mode).

For example, let's combine the last 2 commits into one. Execute the command:

***git rebase -i HEAD~2***

A text editor opens in which the first two lines correspond to the last two commits:

*pick ab37583 Added feature 1.*

*pick 3ab2b83 Added feature 2.*

*# Rebase e46d230..3ab2b83 onto e46d230 (2 commands)*

*#*

*# Commands:*

*# p, pick = use commit*

*# r, reword = use commit, but edit the commit message*

*# e, edit = use commit, but stop for amending*

*# s, squash = use commit, but meld into previous commit*

At the beginning of each line is the word pick. You need to change the word pick to squash or just to the letter s on the second line. This means that this commit will be merged with the previous commit. So, replace pick with s, you should get something like:

***pick ab37583 Added feature 1.***

***s 3ab2b83 Added feature 2.***

Save the changes and close the editor.

The editor opens again, in which you are prompted to enter a message to the commit, which is the union of two commits:

***# This is a combination of 2 commits.***

***# This is the 1st commit message:***

Added feature 1.

***# This is the commit message #2:***

Added feature 2.

***# Please enter the commit message for your changes. Lines starting***

***# with '#' will be ignored, and an empty message aborts the commit.***

Edit the commit message, save the changes, and close the editor.

As a result, you will get a commit that was merged from the last two commits.

**How to Git Stash**

*What is git stash for?*

Let's give an example. For example, you have made some changes to the files and want to switch to another branch, but so that your current changes are not there. Using the git stash command, you can hide these changes. Your changesare placedin a separate storage — in the stack, and you can safely switch to another branch.

Everything you hide with git stash gets into a separate list. Then you can extract from there what you have hidden there — your "hiding places" (this word will be used later in the text).

Let's look at how to use the git stash command

To hide the changes, just run the command:

***git stash***

**How to Create Git Hooks**

Hooks are stored in the hooks subdirectory relative to the main Git directory. For most projects it is .git/hooks. When you initialize a new repository with the git init command, Git fills the hooks directory with sample scripts, most of which are ready for use, and each of them contains documentation on the input data used. All examples are presented in the form of shell scripts containing Perl code, but you can use any language to write scripts — the main thing is to name the executable files correctly. If you decide to use any of the pre-installed scripts, it is enough to simply rename it by removing the .sample suffix.

**Git Squash**

Git squash is a technique that helps to take a series of commits and condense it. For example, suppose: you have a series of N commits and you can compress it into a single commit. Compression via git stash is mainly used to turn a large number of insignificant commits into a small number of significant ones. This makes it easier to keep track of Git history.

This technique is also used when merging branches. Most often, you will be advised to always compress commits and perform rebasing with the parent branch (for example, master or develop). In this case, the history of the main branch will contain only significant commits, without unnecessary detail.

**What is a Pull Request in Git**

A pull request is a request to the owner of another repository to accept the changes you made to the branch on your fork of their repository and integrate them into them as if you were working on them directly.

**What is a Git Rebase**

Rebase (rebasing) is one of the ways in git that allows you to merge the changes of two branches. This method has an advantage over merge — it allows you to rewrite the history of a branch, giving that history the look we need.

**What is Git LFS**

Git LFS is an extension for Git that captures data describing large files in a repository and stores the contents of a binary file in a separate remote repository. When cloning and switching branches in the Git repository, LFS downloads the correct version from this remote repository.

**What is a Git Submodule**

Git submodules allow you to save a git repository as a subdirectory of another git repository. A Git submodule is just a reference to another repository at a certain point in time. Thanks to the submodules, you can include the version history of external code in the Git repository and track it.

1. **it commands needed to master the Git command line interface**

**1.How to set a username and email address**

*git config --global user.name "Tara Routray"*

git config --global user.email "dev@tararoutray.com"

2. Caching credentials

git config --global credential.helper cache

3. Initializing the repository

git init

4. Adding individual files or all files to the prepared files area

git add somefile.js

git add .

5. Checking the repository status

git status

6. Making changes with a single-line message or through the editor

git commit -m "Your short summary about the commit"

git commit

7. View the history of commits with changes

git log -p

8. Viewing a given commit

git show 1af17e73721dbe0c40011b82ed4bb1a7dbe3ce29

git show 1af17e

9. Viewing changes before commit

git diff

git diff –staged

git diff somefile.js

10. Deleting tracked files from the current working tree

git rm dirname/somefile.js

git rm dirname/\*.html

11. Renaming files

git mv dir1/somefile.js dir2

12. Cancellation of prepared and unprepared changes

git checkout somefile.js

git reset HEAD somefile.js

git reset HEAD

13. Changing the last commit

git commit --amend -m "Updated message for the previous commit"

14. Rollback of the last commit

git revert HEAD

15. Rollback of a given commit

git revert 1af17e

16. Creating a new branch and switching to it

git branch new\_branch\_name

git checkout -b new\_branch\_name

17. Viewing the list of branches

git branch

git branch -a

18. Deleting a branch

git branch -d existing\_branch\_name

git branch -D existing\_branch\_name

git push origin --delete existing\_branch\_name

19. Merging of two branches

git merge existing\_branch\_name

git merge --no-ff existing\_branch\_name

20. Displaying the commit log as a graph for the current or all branches

git log --graph --oneline –decorate

git log --all --graph --oneline –decorate

21.Termination of merger in case of conflict

git merge –abort

22. Adding a remote Repository

git remote add awesomeapp https://github.com/someurl..

23. Viewing deleted URLs

git remote -v

24. Getting additional information about the remote repository

git remote show origin

25. Sending changes to a remote repository

git push origin main

26. Getting changes from a remote repository

git pull

git pull --verbose

27. Merging a remote repository with a local one

git merge origin

28. Sending a new branch to a remote repository

git push -u origin new\_branch

29. Deleting a deleted branch

git push --delete origin existing\_branch

30. Using rebasing

git rebase branch\_name