Лабораторная работа GIT

**ЗАСТОСУВАННЯ СИСТЕМИ КОНТРОЛЮ ВЕРСІЙ GIT ТА РЕПОЗИТОРІЯ GITHUB**

**Задания для выполнения.**

**Part 1. Install and configurate git.**

1. Install GIT on your workstation.

Установить Git, для чегонеобходимо загрузить exe-файл инсталлятора со страницы <https://git-scm.com/downloads> и запустить его. После установки в первую очередь интересует консольная версия. Использование Git из командной оболочки обеспечивает доступ ко всем командам git.

Source: <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

2. Run git-Bash and Configurate git: change your global configs:

- your name and email, wich should be add in all commit describe;

- setup core text editor and editor to find diffs.

<https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup>

3. Put in report list of commands or screenshots of your actions in git-Bash.

//=============================================//

**Part 2. Commiting changes in local repository**

**Main aim:** committing process of project develop (creating website) in local repository.

1. Create folder named **“LR\_FIO\_GIT1”** for your future project.

2. Open git-Bash console in root directory of your project and initiate git-control for it. Check status.

<https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository>

<https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository>

3. Create empty *readme.txt* file. Check status.

4. Add files for committing. Make init commit. Check status.

5. Show log. Try to get log by set different variants of attributes.

<https://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History>

6. Look material one more and learn more about the state of project files during version control.

<https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository>

7. Create .gitignore file (for example *readme.xxx*). Include .gitignore in its own list. Try to use different patterns in .gitignore file to make invisible for git some local files that should not be commited. Check that files mentioned in .gitignore file shouldn’t be visible via executing git status command.

8.

//=============================================//

**Part 3. Branching and merging**

0. Study material by refs

<https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell>

<https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging>

<https://git-scm.com/book/en/v2/Git-Branching-Branch-Management>

<https://git-scm.com/book/en/v2/Git-Branching-Branching-Workflows>

1. Create *develop* branch and checkout on it.

2. Create *index.html* emptyfile. Commit.

3. Create branch with name “*images*”. Checkout on it. Add images folder with some images inside it. Commit.

4. Change your *index.html*. Add images source inside it. Commit.

5. Go back to *develop* branch.

6. Create branch with name “*styles*”. Checkout on it. Add styles folder with style source inside it. Commit.

7. Change your *index.html*. Commit.

8. Goto *develop* branch.

9. Merge two new branches into *develop* using **$ git merge** command. Resolve conflict if it appeared. Do it in next sequence:

• merge *images* into *develop*.

• merge *styles* into *develop*.

**Attention!!! Do not delete any branches!**

10. Merge develop into master.

11. Try to inspect your repository with git log command. Use different options with this command **($git log --help)**.

<https://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History>

Put in report log of your activity taking with 3-4 different commands.

//=============================================//

**Part 4. Rebasing**

0. Study material by refs

<https://git-scm.com/book/en/v2/Git-Branching-Rebasing>

1. Check out to *develop* branch and repeat 3.1-3.8 items once more (use names *images2*, *styles2* for new branches).

2. Use **$** **git rebase** command instead of **$ git merge**. Do rebase in next sequence:

• rebase *images2* onto *develop* ;

• rebase (fast-forward) *develop* onto *images2* ;

• rebase *styles2* onto *develop* ;

• rebase (fast-forward) *develop* onto *styles2*.

3. Merge *develop* into *master*.

NOTE: As a result *master/develop* branches should contain all commits that were done in process.

**4. Execute command “git reflog“ and save it content some where (not in repository) with filename “LR\_GIT\_REZ1.txt”.**

//=============================================//

**Part 5. Remote repository**

1. Create account on GitHub

<https://git-scm.com/book/en/v2/GitHub-Account-Setup-and-Configuration>

2. Create remote repository in your account with name “MyRD\_FIO”.

3. Generate ssh-key and integrate it with GitHub.

4. Linking your local and remote repository.

<https://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes>

5. Push all your changes **with all** your branches to remote repository **($ git push <remote repository name> --all**).

4. Check loading your working repository to remote repository by used browser.

5. Create folder for adding new feature in project. Named it **“LR\_FIO\_GIT2”** and clone project in this folder.

6. Checkout on *develop*.

7. Create any file in new folder **“LR\_FIO\_GIT2”** and commit it.

8. Create new branch *first*.

9. Create another new branch *second*.

10. Do one (or more) commits to *develop* - add to repository some files, change them.

11. Checkout on the *first* branch and do several commits (about 5).

12. Checkout on the *second* branch and do several commits (about 3). Make sure that you have at least one commit, which contains changes in more than one file.

13. Do interactive rebasing. Your commits from the *second* branch should appear on the top of *develop* branch. During interactive rebasing:

• Divide one of your commits (so you should have two commits instead of one);

• Change content of other commit and change commit message for it.

14. Merge *second* branch into *develop* (fast-forward). *develop* and *second* branches should point to the same commit.

15. Checkout to the *first* branch.

16. Do interactive rebasing. Your commits from the *first* branch should appear on the top of *develop* branch. During interactive rebasing:

• Squash three commits into one;

• Drop one commit.

17. Merge *first* branch into *develop* (fast-forward). Both branches should point to the same commit.

18. Merge the *develop* branch into *master* branch.

19. Push all your local branches to origin.

**20. Execute command “git reflog” and save it content somewhere (not in repository) with filename “LR\_GIT\_REZ2.txt”.**

**Questions to control**

1. Дайте визначення системи контролю версій.
2. Класифікація систем контролю версій (СКВ). Переваги, недоліки різних СКВ.
3. Особливості СКВ Git. Стан файлів, що зберігаються у Git. Каталоги Git.
4. Створення Git-репозиторію.
5. Запис змін до репозиторію.
6. Життєвий цикл станів файлів.
7. Перегляд історії коммітів. Скасування змін. Робота з віддаленими репозиторіями.
8. Основні об’єкти системи Git. Їх характеристика.
9. Розгалуження в Git. Основи розгалуження і зливання, конфлікти при зливанні.
10. Віддалені гілки. Синхронізація, відстеження віддалених гілок. Відправлення змін на віддалену гілку.
11. Git на сервері. Протоколи обміну даними з Git, їх переваги та недоліки.
12. Віддалений репозиторий. Git-хостінг.
13. Розподілений Git. Розподілені робочі процеси (централізований, з менеджером по інтеграції, з диктатором та його помічниками).
14. Сприяння проекту. Організація процесу проектування (окрема маленька команда, окрема команда з керівником, невеликий відкритий проект, великий відкритий проект).
15. Вибір ревізії. Виділення коммітів за допомогою тегів.
16. Комміти – посилання на предків. Діапазони коммітів.
17. Відлагодження проекту за допомогою Git (анотація файлів, бінарний пошук).
18. Основи налаштування і конфігурування Git.