МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

Інститут Прикладного системного аналізу

Кафедра Системного проектування

**Лабораторна робота № 1.**

З дисципліни

«ПРОЕКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ»

На тему

***«***Системи контролю версій SVN, GIT.***»***

Виконала

студентка групи ДА-61

Шкульова Катерина

КИЇВ

2019

**Мета роботи:** за допомогою системи контролю версій завантажити коди програми у

репозіторій. Відтворити типовий цикл розробки програмного забезпечення з

використанням системи контролю версій.

**Завдання**

1. Обрати безкоштовну систему репозиторія для системи контролю версіями,

наприклад projectlocker, або інш.

2. Встановити клієнтське безкоштовне програмне забезпечення для роботи с системою

контролю версій (GIT, SVN clients).

3. Протягом роботи над лабораторними роботами 2-6 використовувати систему

контролю версіями.

4. Описати цикл розробки програмного забезпечення з використанням системи

контрою версій.

**Хід роботи**

Для системи контролю версіями було оброно систему репозиторія GitHub.

**Розподілена (децентралізована) система керування версіями** ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *Distributed revision control system*) — порівняно новий напрям [систем керування версіями](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BA%D0%B5%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D1%96%D1%8F%D0%BC%D0%B8). Від традиційних, централізованих систем керування версіями децентралізовані відрізняються:

* Кожний розробник працює з власним репозиторієм.
* Модель роботи схожа на «базарний» стиль в тому, що кожен може створити власні [гілки](https://uk.wikipedia.org/wiki/%D0%93%D1%96%D0%BB%D0%BA%D0%B0_(%D0%BA%D0%B5%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D1%96%D1%8F%D0%BC%D0%B8)).
* Репозиторії можуть клонуватись будь ким, і часто клонуються декілька раз.
* Може існувати декілька «центральних» репозиторіїв.
* Обмеження прав доступу не використовуються. Натомість, зміни додаються на основі мереж довіри, наприклад, на основі попереднього досвіду співпраці або якості змін.
* Важелі керування належать керівникам проектів, котрі вирішують які [гілки](https://uk.wikipedia.org/wiki/%D0%93%D1%96%D0%BB%D0%BA%D0%B0_(%D0%BA%D0%B5%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D1%96%D1%8F%D0%BC%D0%B8)) об'єднувати.
* Більшість операцій не потребує мережевого з'єднання.
* Існує перелік операцій для [синхронізації](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85) репозиторіїв — передачі або отримання змін з інших репозиторіїв.

Основною відмінністю GIT від інших систем (таких як Subversion та подібних їй) є те, як він сприймає дані. Концептуально, більшість СКВ зберігають інформацію як список файлових редагувань. Ці інші системи (CVS, Subversion, Perforce, Bazaar тощо) розглядають інформацію як список файлів та змін кожного з них протягом деякого часу (це зазвичай називають **оснований на дельтах** контроль версій).

Git не оброблює та не зберігає свої дані таким чином. Замість цього, Git сприймає свої дані радше як низку знімків мініатюрної файлової системи. У Git щоразу, як ви створюєте коміт, тобто зберігаєте стан вашого проекту, Git запам’ятовує як виглядають всі ваші файли в той момент і зберігає посилання на цей знімок. Для ефективності, якщо файли не змінилися, Git не зберігає файли знову, просто робить посилання на попередній ідентичний файл, котрий вже зберігається. Git вважає свої дані більш як **потік знімків**.

**Основні команди**

git init - створення git-репозитарія

git add <filename> - добавити файл в буфер

git add . - добавити усі файли в буфер

git commit -m "commit text" - коміт. відправляємо вміст буфера в репозиторій

git -am "commit text" - добавлення в буфер та виконуємл коміт однією командою (не працює для нових та видалених файлів, лише для файлів які були змінені)

git staus

git log - найчастіша команда, показує логи (коміти)

git log --oneline - стиснутий вигляд перегляду комітів

git diff <filename> - показати відмінність у  файлу в робочій директорії та репозиторієм  
git diff --storage <filename> - показати відмінність у редагуванні файлу із буферної зони та репозит

git checkout -- <filename> - взяти файл із репозиторія в робочу директорію (відміна внесених змін)

git checkout <hash> <filename> - чекаут файлу із попередньої рівізії <hash> (файл попадає в буферну зону!)

git reset HEAD <filename> - витягнути файл із буфера назад в робочу директорію

git commit --amend -m "commit text" - зміна останнього коміта

git revert <hash> - відміняє коміт (виконує повну протилежність, те що було знищено вертає, те що було добавлено знищує)

git reset --soft <hash>- переміщує HEAD

git reset --mixed <hash>- переміщує HEAD на конкретний коміт, та змінює буфер, щоб він відповідав репозиторію, при цьому робоча директорія не змінюється

git reset --hard <hash> - переміщує HEAD на конкретний коміт, усі зміни які були за цим комітом будуть знищені

gir clean -n тестово показує, які невідслідковані файли будуть знищені командою git clean -f

git clean -f знищує невідслідковані файли із робочої директорії (яких немає в буфері та репозиторії)

git ls-tree HEAD - покаже файли для HEAD

git branch [-r,-a]- покаже список бранчів (\* - активний бранч) (-r покажи віддалені бранчі, -a покаже локільні і віддалені бранчі)

git branch <name\_branch> - створення нового бранча

git branch --merged - покаже перелік бранчів в текучому бранчі

git branch -m <name\_branch> <new\_name\_branch> - переіменування бранча

git branch -u <branch\_name> <alias>/<branch\_name> - задаємо відслідковування бранча на віддаленому репозиторію

git checkout <name\_branch> - переключення на бранч

git checkout -b <name\_branch> - створення нового бранча та переключення на нього

git checkout -d <name\_branch> - знищення бранча

git checkout -D <name\_branch> - знищення бранча (в незалежності, злитий він з іншим бранчем чи ні)

git merge <name\_branch> - зливання бранча <name\_branch>із тикучим бранчом (в якому ми зараз знаходимя)

git merge <alias>/<name\_branch> - зливання бранча <name\_branch> із репозиторія <alias> із тикучим бранчом (в якому ми зараз знаходимя)

git merge --no-ff branch\_name - не робити швидке зливання, зробити  новий коміт із повідомленням (докуиентація процесу зливання)

git merge --ff-only branch\_name - зробити зливання, лише, якщо воно буде швидким

git merge --abort -    перервати(вийти з режиму) зливання

git stash list - перегляд записів у копілці

git stash save "comment" - збереження змін в "копілку"

git stash pop <element\_name> - переносить зміни із копілки в текучу робочу директорію (якщо не вказати елемент, то по змовчанню бере перший елемент)

git stash apply - копіює зміни із копілки в текучу робочу директорію

git stash drop <element\_name> - знищення елементу із копілки

**Віддалені репозиторії**

git remote [-v] - список усіх віддалених репозиторіїв, які відомі git'у

git remote add <alias> <url> - добавлення віддаленого серверу під назвою <alias>, який ми можемо знайти за адресою <url>

git remove rm <alias> - знищення запису про віддалений репозиторій

git push [-u] <alias> <branch\_name> - відправити код на віддалений репозиторій ( -u відслідковувати віддалений репозиторій)

git push - відправляє внесені зміни на віддалений репозиторій, який раніше був заданий як відслідковуваний

git push <alias> :<branch\_name> - знищення бранчу на віддаленому репозиторію

git push <alias> <local\_branch\_name>:<branch\_name> - відправлення локального бранчу <local\_branch\_name> у віддалений репозиторій <alias> у бранч <branch\_name>

git push <alias> --delete <branch\_name> - ще одна (більш зрозуміла команда) знищення бранчу

git clone <url> [<folder>] - клонування віддаленого репозиторія

git pull = git fetch +  git merge

