Звіт (хід виконання та скріншоти) надсилати на адресу [bogdaniht@gmail.com](mailto:bogdaniht@gmail.com), бажано у форматі doc\*, щоб можна було редагувати та писати зауваження.

В полі Subject написати LAB\_SE\_BIO22

**Лабораторна робота №1.**

Distributed version-control system Git

***Мета роботи:*** ознайомитися системами керування версіями. Дослідити та отримати практичні навики щодо створення найпростішої програми та власного репозиторію.

**Завдання на лабораторну роботу:**

***Ознайомитись з теоретичними відомостями***, ретельно опрацювати матеріал. Вміти давати пояснення термінам та поняттям: система керування версіями; централізовані та розподілені системи контролю версіями; репозиторій; приватні та відкриті репозиторії; GitHub.

**1. Теоретичні відомості**

**Система керування версіями** (англ. source code management, SCM) — програмний інструмент для керування версіями одиниці інформації: вихідного коду програми, скрипту, веб-сторінки, веб-сайту, 3D моделі, текстового документу та ін.

Система керування версіями — це потужний інструмент, який дозволяє одночасно, без завад один одному, проводити роботу над груповими проектами.

Системи керування версіями зазвичай використовуються при розробці програмного забезпечення для відстеження, документування та контролю над поступовими змінами в електронних документах: у програмному коді додатків, кресленнях, електронних моделях та інших документах, над змінами яких одночасно працюють кілька людей.

Кожна версія позначається унікальною цифрою чи літерою, зміни документу занотовуються. Зазвичай також зберігається автор зробленої зміни та її час.

Інструменти для контролю версій входять до складу багатьох інтегрованих середовищ розробки.

Системи керування версіями існують двох основних типів: з централізованим сховищем та розподіленим (рис. 1).

Система збереження історії редагувань статей, що застосовується у Вікіпедії є прикладом системи керування версіями.

Система контролю дає можливість зберігати попередні версії файлів та завантажувати їх за необхідності. Вона зберігає повну інформацію про версію кожного з файлів, а також повну структуру проекту на всіх стадіях розробки. Місце зберігання даних файлів називають **репозиторієм**. В середині кожного з репозиторіїв можуть бути створені паралельні лінії розробки — **гілки**.

Гілки зазвичай використовують для зберігання експериментальних, незавершених(alpha, beta) та повністю робочих версій проекту(final). Більшість систем контролю версії дають можливість кожному з об'єктів присвоювати теги, за допомогою яких можна формувати нові гілки та репозиторії.

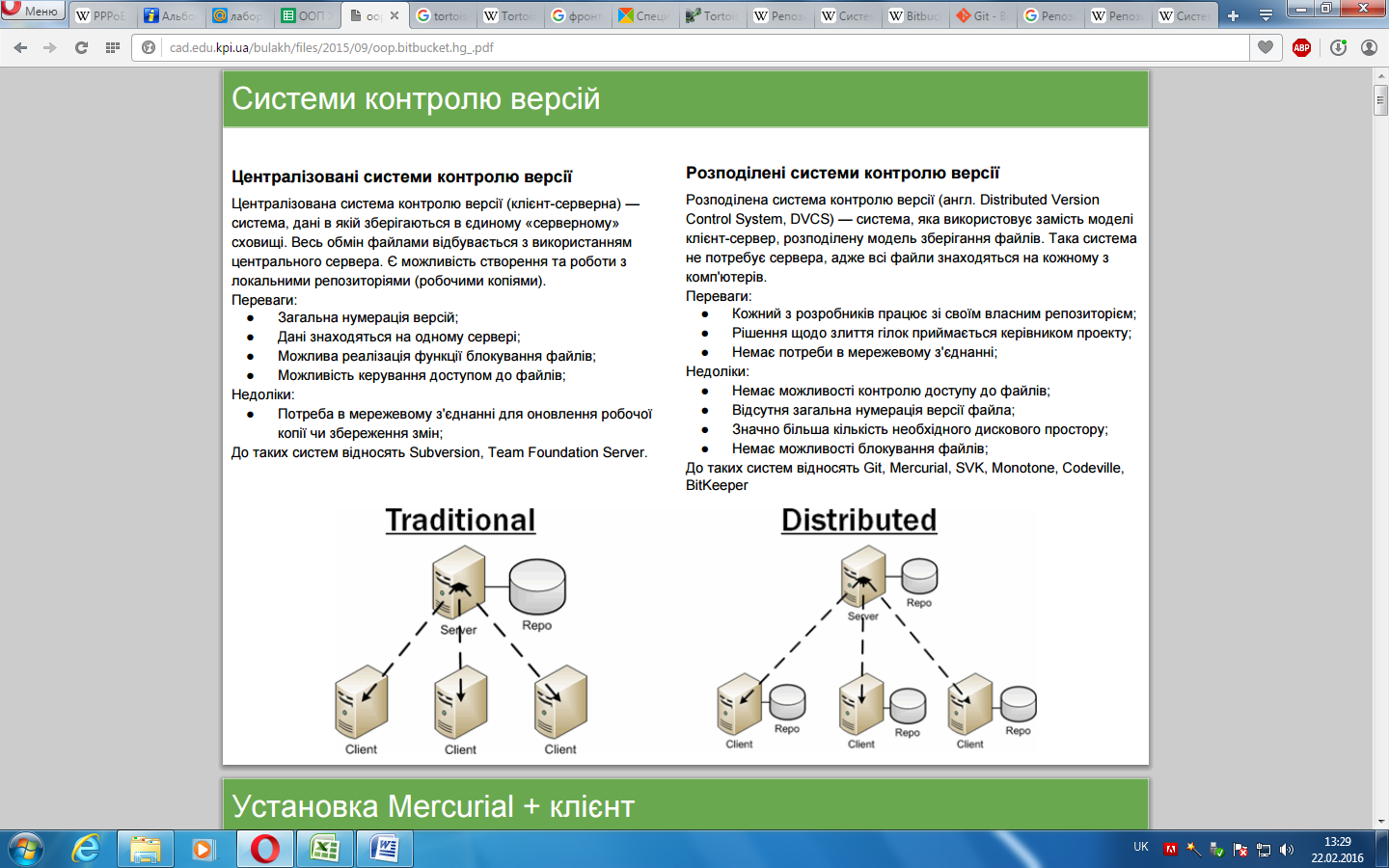


Рисунок 1 – Системи контролю версій

Використання системи контролю версії є необхідним для роботи над великими проектами, над якими одночасно працює велика кількість розробників. Системи контролю версії надають ряд додаткових можливостей:

* можливість створення різних варіантів одного документу;
* документування всіх змін (коли ким було змінено/додано, хто який рядок змінив);
* функція контролю доступу користувачів до файлів (є можливість його обмеження для різних користувачів);
* створення документації проекту з поетапним записом змін в залежності від версії;
* давання пояснення до змін та документування їх.

Найбільш відомими веб-сервісами для хостингу проектів на базі систем керування версіями є:

* GitHub (<https://github.com/>);
* BitBucket (<https://bitbucket.org/>);
* GitLab (<https://gitlab.com/>).

**GitHub —** один з найбільших веб-сервісів для спільної розробки програмного забезпечення. Існують безкоштовні та платні тарифні плани користування сайтом. Базується на системі керування версіями Git і розроблений на Ruby on Rails і Erlang компанією GitHub, Inc (раніше Logical Awesome).

Проект створив Лінус Торвальдс для управління розробкою ядра Linux. Git є однією з найефективніших, надійних і високопродуктивних систем керування версіями, що надає гнучкі засоби нелінійної розробки, що базуються на відгалуженні і злитті гілок. Розробники сайту називають GitHub «соціальною мережею для розробників».

Окрім розміщення коду, учасники можуть спілкуватись, коментувати редагування один одного, а також слідкувати за новинами знайомих. За допомогою широких можливостей Git програмісти можуть поєднувати свої репозиторії – GitHub має зручний інтерфейс для цього і може показувати внесок кожного учасника в вигляді дерева.

Для проектів є особисті сторінки, невеликі Вікі та система відстеження помилок. Прямо на сайті можна дивитись файли проектів з підсвічуванням синтаксису для більшості мов програмування.

Кількість приватних (закритих для перегляду користувачами Інтернету) репозиторіїв – 5. Для того, щоб мати можливість створювати більше приватних репозиторіїв потрібно переходити на платний тарифний план. Кількість відкритих репозиторіїв – необмежена.

Для забезпечення цілісності історії та стійкості до змін заднім числом використовуються криптографічні методи, також можлива прив'язка цифрових підписів розробників до тегів і комітів. Система керування версіями Git дає можливість контролювати зміни файлів у вибраних папках на своїх комп'ютерах і узгоджувати ці зміни зі змінами файлів на комп'ютерах команди розробників, які спільно працють над проектом. Є можливість узгоджання зміни зі сховищем файлових проектів (репозиторієм) на виділеному сервері, у тому числі на серверах загальнодоступних сервісів https://github.com та https://bitbucket.org.

Після завантаження програмного забезпечення Git з сервера <https://git-scm.com> і його встановлення на комп'ютері в списках встановлених програм з'явиться папка Git з посиланнями на додатки Git Bash, Git CMD та Git GUI. Git Bash - інтерпретатор командного рядка *bash* ОС UNIX / Linux перенесений у Windows та інтегрований з Git.

Інтеграція з Git дає можливіcть інтерпретатору виконувати команду Git з будь-якої поточної папки та зберігати всі настройки програми для використання при повторних викликах. Git CMD - інтерпретатор *cmd* командного рядка Windows, який також сприймає команди Git. Для виконання команд Git у вікні Git CMD необхідно командою cd зробити поточну папку з файлами які необхідно контролювати.

Git GUI - графічний інтерфейс користувача програми Git. З його допомогою можна виконати основні операції Git по керуванню файлами проекту в локальному та віддаленому (remote) репозиторіях без необхідності знань синтаксису команди Git. Проте, у деяких випадках його можливості виявляються недостатніми для більш складних випадків спільної роботи над проектами.

Після встановлення Git на комп’ютері в контекстному меню правої кнопки миші з'являться команди запуску *Git Bash Here* і *Git GUI Here*, за допомогою яких можна запустити інтерпретатор команд і графічну оболонку з прив'язкою до відкритої папки з контрольованими файлами проекту або папкою, яку необхідно зробити робочою папкою проекту.

**Практична частина.**

**Настроювання Git.**

1. Відкрийте папку, в якій знаходяться файли проекту. З контекстного меню виберіть *Git Bash Here* , і у вікні, що відкрилося введіть команду:  
*$ git init*

У результаті в папці з проектом з'являється папка з іменем «.git», що містить усі необхідні файли локального репозиторію.

2. Авторизуйтесь для внесення подальших змін, вкажіть своє ім’я та електронну пошту, наприклад:

$ *git config user.name "Bohdan Sus"*

$ *git config user.email* [*suse@univ.kiev.ua*](mailto:suse@univ.kiev.ua)

Якщо ви плануєте постійно працювати з проектами на даному комп'ютері, то додайте до команд авторизації ключ *global*:

$ *git config --global user.name "Bohdan Sus"*

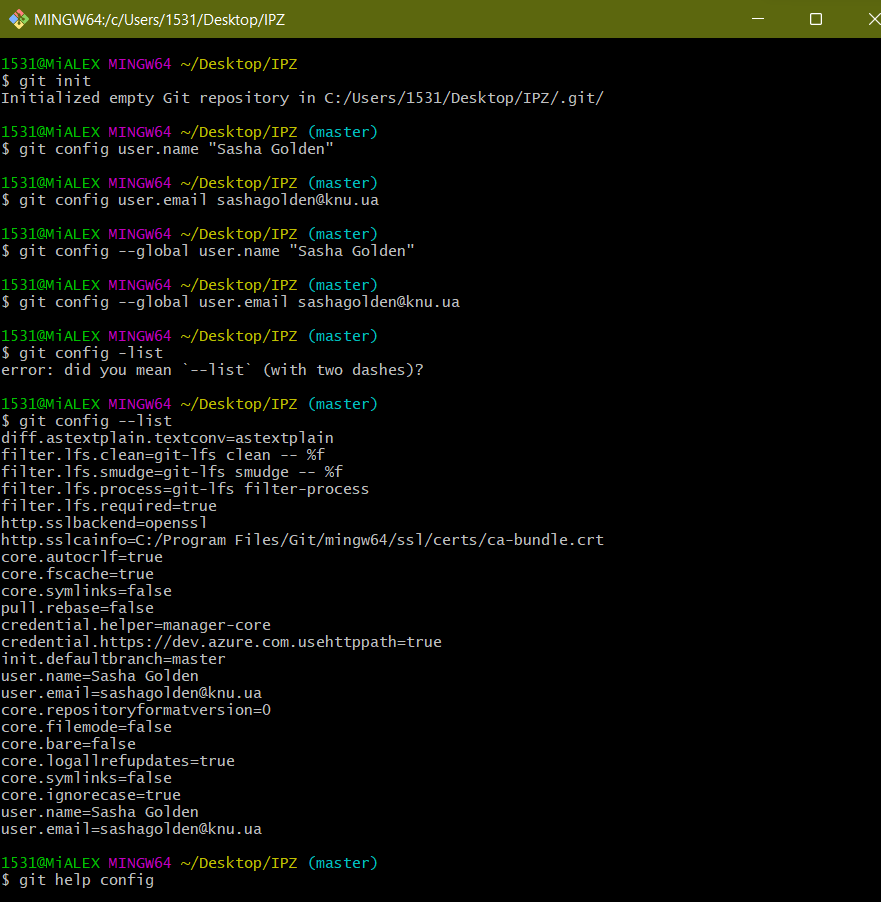
$ *git config --global user.email* [*suse@univ.kiev.ua*](mailto:suse@univ.kiev.ua)

*(Ці настройки достатньо зробити один раз).*

*Перевірити настройки можна за допомогою команди git config –list*

*Щоб отримати допомогу необхідно набрати*

*git help назва команди*

**

1. Створіть в папці з проектом файл *«.gitignore»* для постійного виключення з файлів зі змінами, що відстежуються, тимчасових файлів та файлів, що автоматично генеруються компіляторам та компоновщиком.

При створенні файлу з іменем «.gitignore» за допомогою контекстного меню можлива відмова системи Windows зберегти файл без вказування імені перед точкою. В такому разі збережіть його з будь-яким ім’ям, наприклад, «a.gitignore», а потім в командному рядку cmd або Git CMD перейменуйте його за допомогою команди ren, попередньо перейшовши до поточної папки:

>*ren a.gitignore .gitignore*

Перейменувати файл можна також за допомогою файлового менеджера.

Створіть у в файлі «.gitignore» список файлів та папок, які не потрібно відслідковувати, наприклад:

\*~

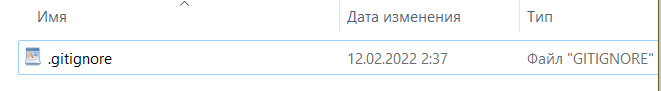
\*obj3

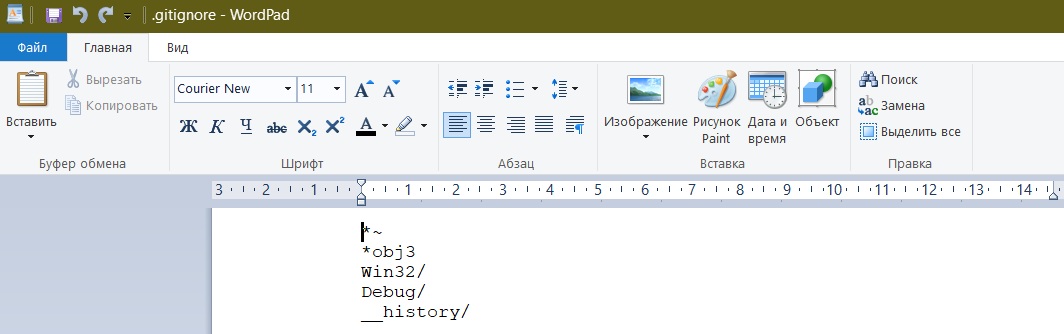
Win32/

Debug/

\_\_history/

Символ «\*» заменює будь яку кількість символів, а символ «/» після імені означає папку.





1. Імпортування папки в Git.

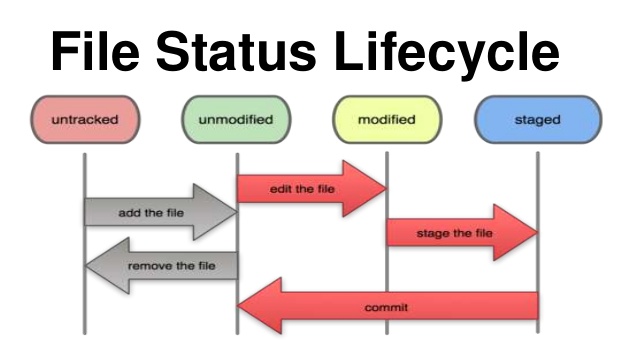
Необхідно зайти в поточну папку проекту і виконати команду

*git init*

В результаті виконання команди має з’явитись невидима папка [.git] зі структурою робочих папок git, або *повідомлення Reinitialized existing Git repository in Path/Folder/.git/*  ,якщо процес ініціалізації виконувався раніше.

В поточній папці проекту кожен файл може знаходитись в двох станах, під вірсійним контролем і без нього, тобто коли файл не відслідковується (untracked).

За допомогою команди *git status* можна подивитись в якому стані знаходиться репозиторій.



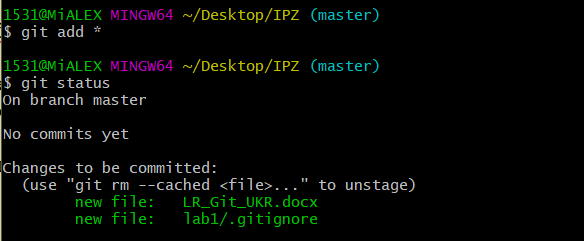
Для того щоб файл почав відслідковуватись необхідно виконати команду

*add <file>*

Додайте всі файли, що знаходяться в папці для відслідковування за допомогою команди

*add \**

Виконайте команду *git status* і перевірте стан файлів.



Відредагуйте якийсь файл проекту, збережітьь зміни і знову виконайте команду *git status*

Червоним кольором буде показао не проіндексовані файли. Для того щоб їх проіндексувати знову необхідно виконати команду

*add <file>*

*git status* показує стан файлу після останнього виконання команди *git add*

Далі необхідно виконати команду git commit –m “first modification”

Commit робить “знімок” проіндексованих файлів в робочому каталозі.

В результаті виконання має з’явитись подібне повідомлення

$ git commit -m "first commit"

[master (root-commit) 9d7386f] first commit

11 files changed, 438 insertions(+)

create mode 100644 App.config

create mode 100644 Form1.Designer.cs

create mode 100644 Form1.cs

create mode 100644 Program.cs

create mode 100644 Properties/AssemblyInfo.cs

create mode 100644 Properties/Resources.Designer.cs

create mode 100644 Properties/Resources.resx

create mode 100644 Properties/Settings.Designer.cs

create mode 100644 Properties/Settings.settings

create mode 100644 README.md

create mode 100644 WindowsFormsApplication1.csproj

1. Додавання імен віддалених репозиторіїв.

Одним з найпопулярніших репозиторіїв є github. На ньому необхідно зареєструватись і створити account.

Далі в консольному вікні необхідно виконати команду, наприклад:

git remote add origin <https://github.com/bohdansus/labs2020.git>

git push -u origin master

В результаті має з’явитись повідомлення типу:

Enumerating objects: 14, done.

Counting objects: 100% (14/14), done.

Delta compression using up to 8 threads

Compressing objects: 100% (13/13), done.

Writing objects: 100% (14/14), 6.44 KiB | 2.15 MiB/s, done.

Total 14 (delta 0), reused 0 (delta 0)

To https://github.com/bohdansus/labs2020.git

\* [new branch] master -> master

Branch 'master' set up to track remote branch 'master' from 'origin'.

**Клонування репозиторію з сервера.**

При спільній роботі над проектами кількох розробників в першу чергу потрібно отримати актуальні версії проекту з віддаленого репозиторію. Для цього необхідно знати коротке ім'я сервера, на якому знаходиться знаходиться віддалений репозиторій або його URL-адреса. Знання короткого імені достатньо, якщо файли в робочу папку вже копіювали з сервера за допомогою команди команди git clone, наприклад:

$ *git clone https://github.com/ім’я\_власника/ім’я\_папки\_проекта* *.*

*напр.*

*$ git clone https://github.com/bohdansus/labs2020*

В результаті виконання буде повідомлення:

*Cloning into 'labs2020'...*

*remote: Enumerating objects: 3, done.*

*remote: Counting objects: 100% (3/3), done.*

*remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0*

*Unpacking objects: 100% (3/3), done.*

При цьому за замовчуванням віддаленому серверу буде присвоєне ім’я «*origin*». Перевірити присвоєні імена серверів можна за допомогою команди *git remote* :

$ *git remote* *–v* .

Напр. $ git remote -v

origin https://github.com/bohdansus/labs2020.git (fetch)

origin https://github.com/bohdansus/labs2020.git (push)

Ключ *–v* дає можливість вивести повні URL-адреси серверів: *origin https://github.com/ ім’я\_власника /ім’я\_папки* (*fetch*) *origin https://github.com/ ім’я\_власника /ім’я\_папки* (*push*)

У дужках вказані команди скачування (fetch) і відправки змін (push) допустимі для цього сервера.

Додати ще одне ім'я сервера з віддаленим репозиторієм можна за допомогою команди *git remote add* :

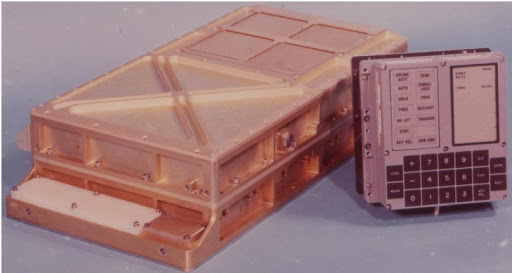
$ *git remote add* *коротке\_ім’я\_сервера* *URL-репозиторію* .

**Завдання 1.**

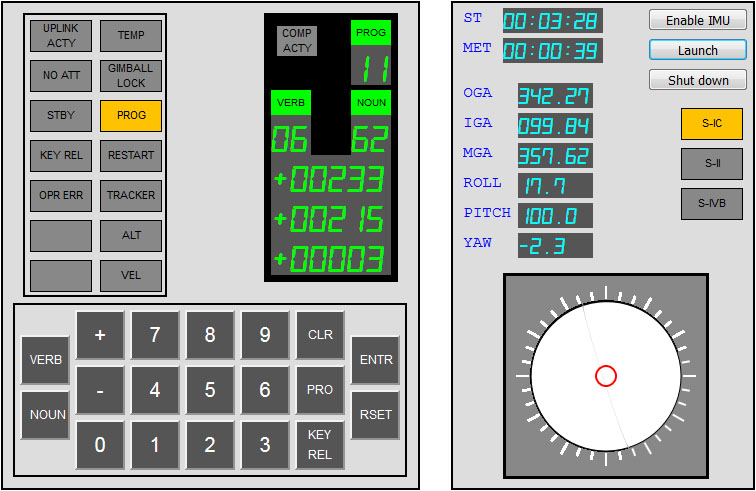
Зробити клонування репозиторію коду космічної місії польоту на місяць.

<https://github.com/chrislgarry/Apollo-11>

При бажанні проглянути та перевірити програмні коди для комп’ютера AGC **(Apollo Guidance Computer)** на симуляторі <http://svtsim.com/moonjs/agc.html>.







**Основні команди Git**

1. Перевірка стану робочої папки здійснюється за допомогою команди

$ *git status* .

Git Bash покаже назву поточної гілки «master», присвоєне за замовчуванням, і список усіх файлів в робочій папці, які не відслідковуються і не підпадають під шаблони файлу «.gitignore». Імена цих файлів будуть виділені червоним шрифтом.

6. Включення файлів і папок в перелік, що відслідковується виконується командою git add:

$ *git add* *імена\_файлів\_або\_шаблони\_імен* .

Наприклад, команда

$ *git add \*cpp \*dfm \*h* .

включає в перелік файлів, що відслідковуються файли вихідних текстів на мові C++ (\*cpp), опис графічної форми (\*dfm) і файли заголовків (\*h).

Внесіть зміни в один з текстових файлів, наприклад, додайте коментар до якого-небудь оператора вихідного тексту програми на мові C ++, і збережіть змінений файл. Знову виконайте команду *git status*. Перелік файлів робочої папки, що відслідковуються, які не були змінені і підготовлені до фіксації їх стану командою *commit* буде виведений зеленим шрифтом. Ім'я зміненого файлу і файлів, які не включені в ті, що відслідковуються, будуть показані червоним шрифтом.

1. Додайте змінений файл до файлів, що підготовлені до фіксації стану (staged files), за допомогою команди

$ *git add* *ім’я\_файла* .

Перевірте стан файлів командою git status і збережіть знімок поточного стану відслідковуються файлів командою git commit: $ git commit -m 'Ver 1.0'.

Перевірте стан файлів за допомогою команди *git status* і збережіть знімок поточного стану файлів, що відслідковуються за допомогою команди *git commit: $ git commit -m 'Ver 1.0'.*

Ключ «-m» дає можливість зберегти повідомлення (message) 'Ver 1.0' як коментар, що описує дану фіксацію (знімок стану). Повідомлення має містити простий текст, поміщений в апострофи або подвійні лапки. В цьому прикладі пропонується вказати, що цей знімок стану є першою версією файлів проекту. Будь-яка фіксація потребує обов'язкового зазначення рядка повідомлення.

Введення команди git commit без вказівки повідомлення запускає текстовий редактор, яким за замовчуванням є стандартний редактор тексту ОС UNIX / Linux Vim. Редактор Vim завжди запускається в командному режимі. Щоб перевести його в режим вставки, тобто в режим введення і зміни тексту, необхідно ввести з клавіатури символи «i» або «a».

В першому пустому рядку відкритого тексту-коментаря необхідно ввести повідомлення, яке описує дану фіксацію, a потім перейти в командний режим натисканням клавіші <Esc>. Вихід з редактора зі збереженням змін виконується за допомогою команди

**:wq** або ***ZZ*** .

При цьому програма git виконає фіксацію стану файлів проекту зі збереженням в якості коментаря рядка введеного повідомлення.

1. Перегляньте історію змін за допомогою команди *git log*:

$ *git log* або $ *git log --pretty=oneline*.

Ключ *«--oneline»* зменшує обсяг виведеної інформації. Ключ *«-p»* дає можливість вивести відмінність між кожним *commit*, а ключ *«число»* - обмежує виведення кількістю *commit*, заданих числом.

commit 9d7386f23952c938a9a9ae2fd1dc59f84dcfe6f8 (HEAD -> master, origin/master)

Author: bogdaniht@gmail.com <bogdaniht@gmail.com>

Date: Fri Sep 1 12:50:50 2020 +0300

Наприклад, команда

$ *git log -p -3*

виведе інформацію про 3 останні commits із зазначенням відмінностей в файлах, що відстежуються.

10. **Створення нової гілки**

Необхідність створення нової гілки в розробці проекту виникає, якщо передбачаються суттєві відхилення від початкового проекту або розробникам потрібне схвалення результатів роботи керівником проекту.

Створіть гілку *testing* за допомогою команди

$ *git branch testing* .

Дізнатись, яка гілка є поточною можна за допомого команди *git log*:

$ *git log* або $ *git log --pretty=oneline*

.

В першому рядку буде виведена інформація про наявність двох гілок *1bdf852 (HEAD -> master, testing) текст\_повідомлення\_commit,* на які вказує вказівник *HEAD.*

Перехід на нову гілку виконується за допомогою команди *git checkout*.

*$ git checkout testing .*

Повторне введення команди *git log --oneline* покаже, що відбувся перехід:

*1bdf852 (HEAD -> testing) текст\_повідомлення\_commit.*

11. Внесіть зміни в один-два текстові файли нової гілки, що відслідковуються і зафіксуйте зміни за допомогою команди

*$ git commit -a -m "текст\_комментаря\_commit"*

Ключ «-*a*» в команді *git commit* дає можливість пропустити команду *git add,* що додає змінені файли в область індексуваня, та готує їх до фіксації.

Перегляньте історію змін гілки *testing* за допомогою команди *git log --oneline* .

12. Об’єднування гілок.

Перейдіть на головну гілку *master* за допомогою команди

*$ git checkout master .*

Виконайте об’єднування гілки *testing* з гілкою *master* за допомогою команди *$ git merge testing.*

$ git merge testing

Updating 9d7386f..ed802bc

Fast-forward

Form1.Designer.cs | 2 +-

1 file changed, 1 insertion(+), 1 deletion(-)

Перегляньте історію змін за допомогою команди *$ git log --graph --oneline*

Ключ «*--graph*» дає можливість графічно показати історію розділення та об’єднання гілок *master* и *testing*:

$ git log --graph --oneline

\* ed802bc (HEAD -> master, testing) second commit

\* 9d7386f (origin/master) first commit

Перегляньте результат об’єднання виправлених файлів в текстовому редакторі.

**Завдання 2.**

Написати програму для розрахунку чисел Фібоначчі та завантажити на Github.

Питання для перевірки: (під час захисту)1. Що таке Git?  
2. Які види систем керування версіями Ви знаєте?.  
3. Які можливості надають системи контролю версій?  
4. Які системи контролю версій називають централізованими?  
5. Які системи контролю версій називають локальними?  
6. Які системи контролю версій називають розподіленими?  
7. Які переваги централізованих систем контролю версій?  
8. Які недоліки централізованих систем контролю версій?  
9. Які переваги локальних систем контролю версій?  
10. Які недоліки локальних систем контролю версій?  
11. Які переваги розподілених систем контролю версій?  
12. Які недоліки розподілених систем контролю версій?  
13. В якому статусі можуть перебувати файли при роботі з Git?  
14. Для чого використовується команда add?

15. Для чого використовується команда status?  
16. Для чого використовується команда commit?  
17. Для чого використовується команда init?  
18. Що необхідно писати в коментарях комітів?  
19. Як переглянути список комітів?  
20. Для чого використовується команда checkout?

21.Як створити нову гілку?  
22. Як перейти на нову гілку?  
23. Як перейти на основну гілку?  
24. Як виконати повне злиття гілок?