Технології та Інструменти Розробки Програмного Забезпечення

ЛР №1 "Базова робота з git"

Постановка задачі

1. Зклонувати будь-який невеликий проєкт open-source з github:
    1. Зклонувати двічі: повний репозиторій, а також частковий, що міститиме лише один коміт однієї гілки за вашим вибором.
    2. Показати різницю в розмірі баз даних двох клонів.
2. Зробити не менше трьох локальних комітів:
    1. Продемонструвати різні способи додавання файлів до індексу.
    2. Продемонструвати різні способи вказання повідомлення коміту.
3. Продемонструвати уміння вносити зміни до останнього коміту за допомогою опції `--amend`.
4. Продемонструвати уміння об'єднати кілька останніх комітів в один за допомогою `git reset`.
5. Видалити файл(и) одним способом на вибір.
6. Перемістити файл(и) одним способом на вибір.
7. Гілкування:
    1. Створити три гілки, принаймні з одним унікальним комітом кожна
    2. Показати уміння переключатися між гілками
8. Продемонструвати уміння знайти в історії комітів набір комітів, в яких була зміна по конкретному шаблону в конкретному файлі (див. методичні вказівки).

Контрольні питання

1. Що таке локальна база даних гіта, а що таке робоче дерево?
2. В яких трьох станах може перебувати файл в робочому дереві?
3. Що таке часткове клонування, і чому при ньому локальна база даних займає менше місця?
4. Що таке коміт?
5. Що необхідно зробити із файлом після його зміни, щоби зберегти ці зміни в гіті?
6. Що означає тильда при вказанні після коміту чи бранча, наприклад `HEAD~`, `my-branch~3`?
7. Що робить команда `git rm`?
8. Що робить команда `git mv`?
9. Які два способи створення нової гілки пропонується в методичних вказівках? Чим вони відрізняються між собою?
10. Що відбувається з робочим деревом при зміні активної гілки?
11. Чим відрізняється локальна гілка від віддаленої?
12. Які способи зміни останнього коміта ви знаєте?

Методичні вказівки

- Клонування
- Коміти
- Зміна комітів
- Видалення та переміщання файлів
- Базове гілкування
- Перегляд історії та змін

Клонування

Для початку створимо папку для виконання ЛР:

```
mkdir ~/lab1
cd ~/lab1
```

Наступні команди передбачатимуть виконання з середини директорії `lab1`.

Для отримання локального репозиторію з віддаленого сервера використовується команда `git clone`. В найпростішому вигляді вона приймає лише один параметр: URL на віддалений репозиторій.

Зклонуємо віддалений репозиторій (для прикладу використовуватимемо утиліту `OpenRGB`):

```
$ git clone https://gitlab.com/CalcProgrammer1/OpenRGB.git
```

Після успішного виконання даної команди з'явиться папка з оригінальною назвою репозиторія -- `OpenRGB`. Оскільки `git` є розподіленою системою контролю версій, кожна локальна копія зазвичай містить базу даних об'єктів гіта. При виконанні звичайної команди `git clone` буде локально отримано повну історію всіх файлів, комітів, тегів, гілок.

Втім, не завжди потрібно мати всю повну історію, адже вона може займати значне місце на диску, вимагаючи багато часу на отримання з віддаленого сервера та навантажуючи мережеві ресурси. Наприклад на сервері, що запускає тести, важливо отримати лише один єдиний, останній, стан якоїсь однієї конкретної гілки чи тегу.

Для ілюстрації зклонуємо той же репозиторій в режимі "для тестів":

```
$ git clone https://gitlab.com/CalcProgrammer1/OpenRGB --depth=1 --single-
branch --branch=qmk_sonix openrgb-shallow
```

набір опцій означає наступне:

- `--depth=1` -- задає кількість комітів історії, що нам потрібна для скачування
- `--single-branch` -- задає гіту отримувати лише одну гілку
- `--branch=qmk_sonix` -- задає гіту ім'я гілки, робоче дерево якої буде створене після клонування. В поєднанні із `--single-branch` задає ім'я єдиної гілки, яку треба отримати
- `openrgb-shallow` -- останній опційний параметр вказує бажане ім'я локальної папки, замість оригінального імені репозиторію, в яку буде зклоновано проєкт.

Варто зауважити, що локальні папки з гітовими репозиторіями можна переіменовувати за потребою, це жодним чином не впливає на його роботу.

Тепер порівняємо отримані репозиторії:

```
$ du -sh ./*/.git
24M ./openrgb-shallow/.git
38M ./OpenRGB/.git
```

Як бачимо, розмір бази даних гіта є різною, оскільки скачане дерево історії у `openrgb-shallow` містить лише об'єкти, необхідні для відтворення одного останнього стану однієї гілки `qmk_sonix`

Коміти

Користуючись існуючим репозиторієм, створимо дві зміни:

- змінити текст будь-якого існуючого в репозиторії файлу
- сворити новий файл з будь-яким текстом

Скористаємося командою `git status` для отримання інформації про стан репозиторію:

```
$ git status
On branch qmk_sonix
Your branch is up to date with 'origin/qmk_sonix'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   OpenRGB.h

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    dummy.h

no changes added to commit (use "git add" and/or "git commit -a")
```

Часто буває так, що робота з проєктом (збірка, запуск тестів) створює десятки, а то і сотні нових файлів, що не повинні бути включеними до репозиторію. В такому разі, якщо нас цікавлять лише зміни до існуючих файлів репозиторію, можна скористатися наступною опцією команді `git status`

```
$ git status -uno
On branch qmk_sonix
Your branch is up to date with 'origin/qmk_sonix'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   OpenRGB.h

no changes added to commit (use "git add" and/or "git commit -a")
```

в даному випадку ми використали одну опцію `-u`, передавши їй аргумент `no`. Це зручний спосіб перевірки статусу на повсякдень, головне пам'ятати про ті корисні файли, які ви створювали -- в такому виконанні команди статусу ви їх не побачите.

Для створення коміту необхідно або додати до індексу файли для подальшого збереження в коміті, або скористатися опцією `git commit -a`

Спочатку скористаємося опцією `-a`:

```
$ git commit -am "some changes to OpenRGB.h"
[qmk_sonix 1e62374] some changes to OpenRGB.h
 1 file changed, 3 insertions(+), 1 deletion(-)
```

Як бачимо, в коміт автоматично додався файл `OpenRGB.h` -- той, що уже існував в репозиторії. Але той файл, що ми створили, але про який гіт ще нічого не знає -- `dummy.h` -- так і залишився в незмінному стані `unstaged`:

```
$ git status
On branch qmk_sonix
Your branch is ahead of 'origin/qmk_sonix' by 1 commit.
  (use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    dummy.h

nothing added to commit but untracked files present (use "git add" to track)
```

Для додавання нового файлу необхідно викликати команду `git add`: або явно на цьому файлі, або на будь-якій його батьківській папці любого рівня вкладеності. Якщо `git add` викликається на папку, будуть рекурсивно додані зміни будь-якого типу в ній та усіх підпапках.

```
$ git add dummy.h

$ git status
On branch qmk_sonix
Your branch is ahead of 'origin/qmk_sonix' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   dummy.h

$ git commit -m "create dummy.h file"
[qmk_sonix 25c084e] create dummy.h file
 1 file changed, 1 insertion(+)
 create mode 100644 dummy.h
```

Зміна комітів

Уявімо, що для вирішення задачі вам дозволено мати лише один коміт.

А також виявилося, що для цього вам необхідно додати ще зміни: наприклад, ще один новий файл `dummy2.h`.

Для вирішення такої ситуації є кілька способів, розглянемо два найбільш розповсюджені:

Спосіб 1: git commit --amend

Перший спосіб -- скористатися опцією `--amend`, що створена для зміни останнього коміту. Вона працює наступним чином: ви робите всі необхідні зміни, що необхідно додати до попереднього коміту, а далі просто використовуєте `git commit --amend` . Замість створення нового коміту, нові зміни додатуться в попередній. Якщо необхідно також оновити повідомлення коміту (commit message) -- можна скористатися опцією `-m` в комбінації з `--amend`. Наприклад:

```
$ git status
On branch qmk_sonix
Your branch is ahead of 'origin/qmk_sonix' by 2 commits.
  (use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    dummy2.h

nothing added to commit but untracked files present (use "git add" to track)

$ git add dummy2.h

$ git commit --amend -m "create dummy.h and dummy2.h files"
[qmk_sonix 09f3ddb] create dummy.h and dummy2.h files
 Date: Sat Sep 16 17:38:25 2023 +0300
 2 files changed, 2 insertions(+)
 create mode 100644 dummy.h
 create mode 100644 dummy2.h

$ git status
On branch qmk_sonix
Your branch is ahead of 'origin/qmk_sonix' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Як бачимо кількість нових комітів не змінилася, але змінилося наповнення та опис останнього коміту.

Спосіб 2: git reset HEAD~N

Для подальшої команди необхідно виконати сценарій зі способу 1, щоби у вас було два власні коміти: один зі зміною існуючого файла, і один з додаванням двох нових файлів.

Якщо вам необхідно зробити зміни в декількох останніх комітах, чи наприклад об'єднати їх в один, можна попросити гіт відкотитися на N комітів назад, але при цьому залишивши фізично зміни в робочому дереві. Наприклад:

```
$ git reset HEAD~2
Unstaged changes after reset:
M   OpenRGB.h

$ git status
On branch qmk_sonix
Your branch is up to date with 'origin/qmk_sonix'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   OpenRGB.h

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    dummy.h
    dummy2.h

no changes added to commit (use "git add" and/or "git commit -a")
```

Бачимо, що гіт скасував два останні коміти, але залишив на диску збережені в них зміни. Якби ми додали до команди `reset` опцію `--hard` -- гіт би просто скасував коміти, повністю привівши стан репозиторію до того коміта, який ми вказали.

`HEAD~N` -- адресує N-ний коміт назад від поточної голови. Тобто `HEAD~1` (або скорочено для одного коміту `HEAD~`) адресує попередній коміт, `HEAD~3` адресує третій коміт назад від поточного.

Таким чином, тепер залишається доробити необхідні зміни, створити новий коміт з усіма необхідними змінами.

Видалення та переміщання файлів

Видалення спосіб 1

Якщо ми просто видалимо файл, що гіт відслідковує, з файлової системи -- з точки зору гіта це буде "зміна по видаленню файла". Цю "зміну" необхідно явно додати до індексу для коміта, після чого закомітити видалення файлу:

```
$ rm dummy2.h

$ git status -uno
On branch qmk_sonix
Your branch is ahead of 'origin/qmk_sonix' by 2 commits.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    deleted:    dummy2.h

no changes added to commit (use "git add" and/or "git commit -a")

$ git add dummy2.h  # додаємо до індексу факт видалення файлу

$ git status -uno
On branch qmk_sonix
Your branch is ahead of 'origin/qmk_sonix' by 2 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    deleted:    dummy2.h

Untracked files not listed (use -u option to show untracked files)

$ git commit -m "delete dummy2.h"
[qmk_sonix bacbad0] delete dummy2.h
 1 file changed, 1 deletion(-)
 delete mode 100644 dummy2.h
```

Видалення спосіб 2

Альтернативно, можна скоротити кількість дій скориставшись `git rm`, що одразу а) фізично видалить файл з файлової системи та б) додасть факт видалення файлу до індексу. Залишиться лише закомітити.

```
$ git rm dummy2.h
rm 'dummy2.h'

$ git status -uno
On branch qmk_sonix
Your branch is ahead of 'origin/qmk_sonix' by 2 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    deleted:    dummy2.h

Untracked files not listed (use -u option to show untracked files)

$ git commit -m "delete dummy2.h"
[qmk_sonix 22735dc] delete dummy2.h
 1 file changed, 1 deletion(-)
 delete mode 100644 dummy2.h
```

Переміщення спосіб 1

Точно такою ж логікою, як для `git rm`, можна користуватися для виконання переміщення файлу. З тією відмінністю що переміщення як такого з токи зору гіта не існує, є факт видалення файлу та факт додавання файлу в новому місці.

Втім, якщо ці дві зміни зробити в одному коміті, гіт зрозуміє що це фактично переіменування файлу, про що вам і напише.

Також не забуваємо факт, що з точки зору файлової системи переміщення це фактично переіменування файлу, з іншим до нього повним шляхом.

Для прикладу переіменуємо файл `dummy.h` в `important.h`:

```
$ mv dummy.h important.h

$ git status -uno
On branch qmk_sonix
Your branch is ahead of 'origin/qmk_sonix' by 3 commits.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    deleted:    dummy.h

no changes added to commit (use "git add" and/or "git commit -a")

$ git add important.h

$ git status -uno
On branch qmk_sonix
Your branch is ahead of 'origin/qmk_sonix' by 3 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   important.h

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    deleted:    dummy.h

Untracked files not listed (use -u option to show untracked files)

$ git add dummy.h

$ git status -uno
On branch qmk_sonix
Your branch is ahead of 'origin/qmk_sonix' by 3 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    renamed:    dummy.h -> important.h

Untracked files not listed (use -u option to show untracked files)

$ git commit -m "rename dummy.h to important.h"
[qmk_sonix 076debf] rename dummy.h to important.h
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename dummy.h => important.h (100%)
```

Переміщення спосіб 2

Альтернативно, можна скоротити кількість дій, скориставшись `git mv`, що одразу зробить фізичне переміщення, а також додавання обох змін -- видалення старого файлу та додавання нового -- до індексу. Залишається лише закомітити.

```
$ git mv dummy.h important.h

$ git status -uno
On branch qmk_sonix
Your branch is ahead of 'origin/qmk_sonix' by 3 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    renamed:    dummy.h -> important.h

Untracked files not listed (use -u option to show untracked files)

$ git commit -m "rename dummy.h to important.h"
[qmk_sonix 94644ad] rename dummy.h to important.h
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename dummy.h => important.h (100%)
```

Базове гілкування

Перегляд гілок

Гілка -- це вказівник на якийсь коміт.

Віддалена гілка -- гілка (вказівник), який існує на віддаленому сервері, звідки клонували репозиторій. При клонуванні локально створюється лише одна гілка.

Локальна гілка -- гілка (вказівник), який існує у вас локально, але не на віддаленому сервері. Назва локальної гілки може бути довільною, прив'язка до назв віддалених гілок м'яка (тобто, за бажання можна "прив'язати" локальну гілку з будь-якою назвою до будь-якої віддаленої гілки, причому прив'язка є лише указанням в яку віддалену гілку класти коміти з локальної при виконанні `git push`)

Для перегляду списку локальних гілок використовується команда `git branch`:

```
$ git branch
  my-local-branch-1
  my-local-branch-2
  my-local-branch-3
* qmk_sonix
```

Біля тієї гілки, стан коміту якої наразі відображений в робочому дереві, тобто активної гілки, буде стояти зірочка.

Для перегляду віддалених гілок використовується опція `-a`:

```
$ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/acermonitor
  remotes/origin/alienware_aw510k_merge
```

`origin` -- назва віддаленого серверу, за замовчуванням `origin`, але може бути довільне. `remotes` -- необхідний префікс

Створення гілок

Створити гілку можна кількома способами, найбільш розповсюджені це:

```
$ git branch my-local-branch-1
$ git checkout -b my-local-branch-2
```

Різниця полягатиме лише в тому, що при виконанні `git checkout -b` одразу ж буде здійснено зміну активної гілки на новостворену. Стан робочого дерева при цьому не зміниться, оскільки обидві гілки -- стара та новостворена -- вказуватимуть на один і той же коміт.

Команда `git branch <some-name>` при передачі їй аргументу трактує аргумент як ім'я гілки, яку необхідно створити, і створює її з посиланням на поточний коміт, але активною залишається стара гілка.

Переключення гілок

Для зміни гілки використовується команда `git checkout`. При цьому відбудеться зміна активної гілки, над якою тепер будуть виконуватися команди, а також робоче дерево приведеться до вигляду того стану, на який вказує гілка, на яку ми переключилися. Тобто, файли які існують лише в гілці, на яку ми перемкнулися, будуть створені у файловій системі, а натомість файли, які існували лише в попередній гілці, але не існують в поточній -- будуть видалені з файлової системи. Проте вони, звісно ж, продовжать знаходитися в базі даних гіта, і для їх перегляду треба переключитися назад на ту гілку, в стані якої вони існують.

Перегляд історії та змін

Базове використання

Для демонстрації роботи з переглядом історії та змін перейдемо до репозиторію з повною історією проєкту.

Для перегляду історії комітів використовується команда `git log`.

```
$ git log -n 5
commit 10e53074b2b0428fb0490101331e684ba2b7b0d6 (HEAD -> master,
origin/master, origin/HEAD)
Author: Pedro Martello <martello@gmail.com>
Date:   Fri Sep 8 23:03:25 2023 +0000

    Support for ASUS TUF RTX 4070 12G Gaming graphics card

commit 96dd52a5e9c4c7464abc02843932f45eb491f43e
Author: Peter Repukat <orly.alia5@gmail.com>
Date:   Thu Sep 7 01:08:21 2023 +0200

    Fix k95_plat iso key mapping

commit a42b55f391061128f1d96aefd49ee2ee87046c0d
Author: thombo <thomas.boos@bluewin.ch>
Date:   Tue Sep 5 21:03:37 2023 +0200

    Support for MSI board 7D40 added

commit 7471dad28fd2b2204fdd5e4686b8470074755add
Author: c10l <gitlab@c10l.cc>
Date:   Tue Sep 5 01:08:46 2023 +0000

    Add missing build dependency to README

commit 824cd7e3fa471e500c06a4b5955a744fc957bc98
Author: MmAaXx500 <viktor.balogh45@gmail.com>
Date:   Mon Sep 4 17:05:07 2023 +0000

    Fix Corsair Hydro udev rules
```

Аргументом `-n` можна обмежити кількість комітів для перегляду.

При форматуванні release notes, зручно використовувати наступний набір опцій:

```
$ git log 21092ef7..HEAD --format="%h %as '%s'"
10e53074 2023-09-08 'Support for ASUS TUF RTX 4070 12G Gaming graphics card'
96dd52a5 2023-09-07 'Fix k95_plat iso key mapping'
a42b55f3 2023-09-05 'Support for MSI board 7D40 added'
7471dad2 2023-09-05 'Add missing build dependency to README'
824cd7e3 2023-09-04 'Fix Corsair Hydro udev rules'
b452110b 2023-08-23 'Fix serial numbers from buggy ASUS keyboard firmwares'
```

Де `21092ef7..HEAD` задає діапазон комітів (можна використовувати короткі чи довгі хеші, назви гілок, тагів), а також опція `--format` в якій ви пишете шаблон формування виводу під необхідний вам формат. Зручно форматувати так щоби одразу можна було вставити весь вивід в електронну таблицю, розпарсивши як CSV.

Пошук комітів, в яких зробили конкретну зміну в конкретний файл

Часто трапляється типова задача: є файл, в якому є рядок, по якому треба довідатися історію його зміни. Для цього використовується комбінація `git log` та `git diff`.

Спочатку за допомогою `git log` виведемо лише ті коміти, в яких змінювався текс заданий шаблоном через опцію `-G` та обмежимо вивід лише файлом, що нас цікавить `README.md`:

```
$ git log -G 'View device information' README.md
commit 942df264324e96903524ffcbee46ee23988db582
Author: Adam Honse <calcprogrammer1@gmail.com>
Date:   Mon Jun 29 14:20:48 2020 -0500

    Update Readme
```

Тепер за допомогою `git diff` переглянемо зміни, які були внесені даним комітом у файл `README.md`:

```
$ git diff
942df264324e96903524ffcbee46ee23988db582~..942df264324e96903524ffcbee46ee239
88db582 README.md
```

Нагадаємо, що за допомогою тильди `~` адресується попередній коміт до вказаного.

Альтернативно можна переглянути зміни коміту за допомогою `git show`:

```
git show 942df264324e96903524ffcbee46ee23988db582 README.md
```