МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ДОПОВІДЬ НА ТЕМУ

" Системи управління версіями. Стратегія створення гілок. Long-living branches (GitLab-flow)"

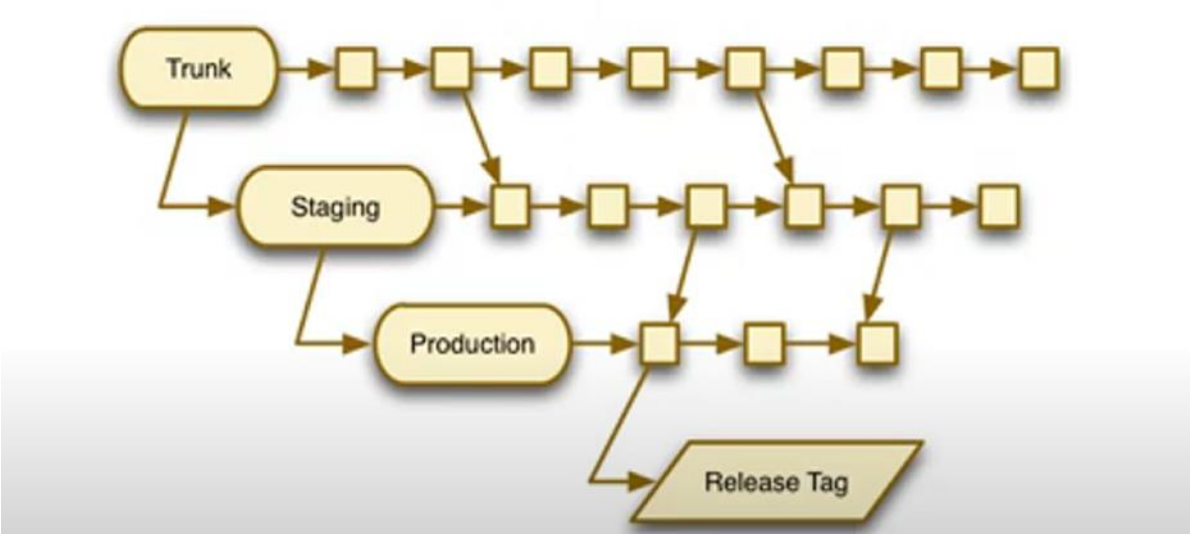
з дисципліни «Автоматизація тестування програмного забезпечення»

Виконала студентка: Наталія ЛУШНИЦЬКА
Група: ІН.мз-41с
Перевірив: Артем Геннадійович КОРОБОВ

Суми, 2025

Варіант 4

Long-living branches (GitLab-flow)



Long-living branches (довгоживучі гілки) - це гілки, які існують у репозиторії протягом тривалого часу і виконують роль стабільних або основних гілок для підтримки різних середовищ або етапів розробки. У GitLab Flow такі гілки зазвичай захищені, щоб уникнути випадкових змін без належного контролю.

Основні особливості long-living branches у GitLab Flow

Основна (main/master) гілка - містить стабільний, готовий до продакшену код. Вона захищена від прямих змін більшістю розробників, і оновлення в неї потрапляють через merge requests після перевірки та рев'ю.

Гілка production - додаткова довгоживуча гілка, яка використовується для безпосереднього розгортання в продакшен. Вона створюється від main і служить для управління релізами, коли час випуску обмежений або потребує додаткової перевірки (наприклад, в iOS додатках або при суворих часових рамках релізу).

Feature branches (фічеві гілки) - короткоживучі гілки, що створюються від main для розробки окремих функцій або виправлень. Після завершення роботи вони зливаються назад у main через merge requests.

Переваги використання довгоживучих гілок

- Захист стабільного коду від випадкових змін.

- Чітке розмежування середовищ розробки, тестування і продакшену.
- Можливість контролювати час релізу через окрему гілку production.
- Підвищення якості коду завдяки обов’язковим merge requests і рев’ю.
- Інтеграція з CI/CD для автоматичного тестування і розгортання.

Порівняння з іншими моделями

На відміну від GitHub Flow, де є лише main і feature branches, GitLab Flow додає довгоживучі гілки, такі як production, для кращого контролю релізів і середовищ. У порівнянні з Gitflow, де є develop і release гілки, GitLab Flow спрощує структуру, зосереджуючись на main і production як основних довгоживучих гілках.

Використання Git Flow у повсякденній розробці

Щоб почати роботу з моделлю Git Flow, вам спочатку необхідно правильно налаштувати структуру гілок:

Ініціалізуйте свій репозиторій Git:

Термінал

git ініціалізація

Додайте гілку develop: Хоча основна гілка створюється автоматично під час запуску, git ipitname також необхідно створити гілку «develop», яку ми будемо використовувати в якості основної для всієї нашої роботи з розробки.

Термінал

git checkout -b разрабатывать основной

git push -u origin разрабатывать

Це налаштування визначає develor гілку, в якій відбувається вся інтеграція, і main гілку, що відображає стан готовності до виробництва.

Робочий процес гілки функцій

Гілки функцій використовуються для незалежного розроблення нових поліпшень або функцій. Вони відгалужуються від гілки розробки і зливаються з нею. Робочий процес гілки функцій особливо корисний у

спільних середовищах, де кілька розробників одночасно працюють над різними аспектами проєкту.

Ізоляція кожної функції у власній гілці гарантує, що developгілка залишиться стабільною, що знижує ризик внесення помилок в основну лінію розробки. Кожна гілка функції має бути присвячена певній частині функціональності або поліпшення і незалежна від інших гілок функції, що дає змогу проводити цільове тестування та перевірку коду. Така ізоляція спрощує як розуміння, так і усунення неполадок функції під час розробки. Після завершення, тестування та перевірки коду функції її можна знову об'єднати з developверткой, зробивши її частиною наступного запланованого циклу випуску.

Створення гілки функцій:

Термінал

git checkout -b feature/ < ім'я_функції > розробляти

Интеграция готовой функции:

Терминал

git checkout разрабатывать

git merge --no-ff feature/ < ім'я_функції >

git branch -d feature/ < ім'я_функції >

git push origin разрабатывать

Опція --no-ffстворює коміт злиття, навіть якщо можливе швидке злиття, зберігаючи історію гілки функцій як окремої сутності. Це полегшує розуміння контексту змін і полегшує як відстеження конкретних розробок функцій, так і потенційні повернення цілих функцій за потреби.

Робочий процес гілки релізу

Гілки випуску допомагають підготувати новий виробничий реліз. Вони дають змогу вносити незначні виправлення помилок і готуватися до випуску, наприклад, оновлювати метадані та номери версій.

Створення гілки релізу:

Терміналgit checkout -b release/ < версия > разрабатывать

Завершення гілки релізу:

Після останніх доопрацювань і оновлень:

Термінал

git checkout основний

git merge --no-ff release/ < версія >

git tag -a < версія >

git checkout розробляти

git merge --no-ff release/ < версія >

git branch -d release/ < версия >

git push origin основний --теги

git push origin розробляти

Позначка релізу в основній гілці допомагає відстежувати версії і може полегшити процеси розгортання.

Робочий процес гілки виправлення

Гілки виправлення використовуються для негайного реагування на небажаний стан версії живого виробництва. Вони дають змогу швидко вносити виправлення у виробництво, не перериваючи поточну роботу з розробки.

Створення гілки виправлень:

Терміналgit checkout -b hotfix/ < версія > main

Завершення гілки виправлень:

Після застосування виправлення:

Термінал

git checkout основний

git merge --no-ff hotfix/ < версія >

git tag -a < версія >

git checkout розробляти

git merge --no-ff hotfix/ < версія >

git branch -d hotfix/ < версія >

git push origin основний --теги

git push origin розробляти

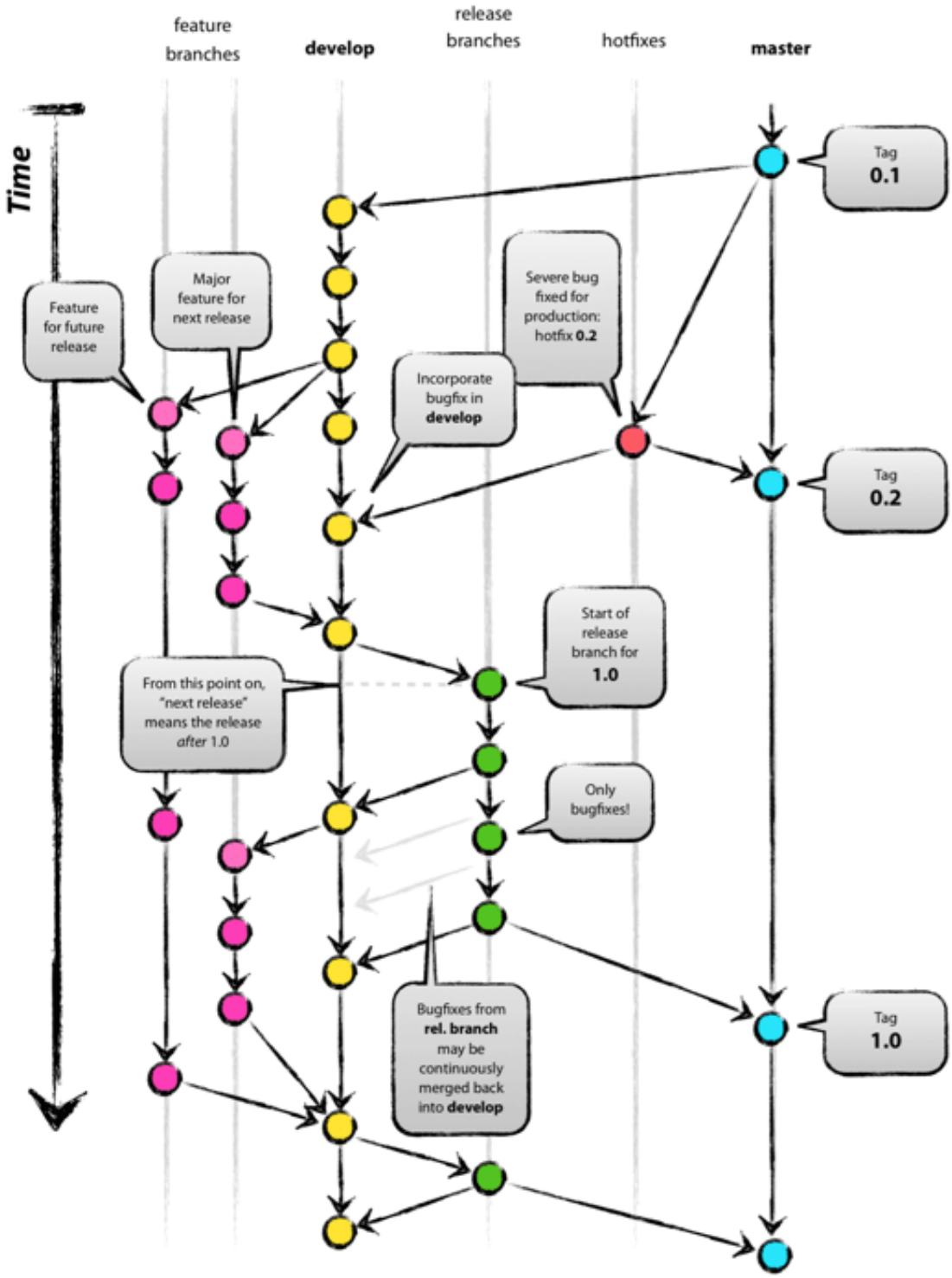
Проблеми з Git-потоком

Git-потік був однією з перших пропозицій щодо використання гілок Git'a, і він привернув до себе багато уваги. Він передбачає наявність основної гілки та окремої гілки для розробки, з допоміжними гілками для фіч, релізів та хотфіксів. Розробка відбувається на гілці develop, переходить на гілку release і, нарешті, об'єднується в основну гілку.

Git-флоу є чітко визначеним стандартом, але його складність створює дві проблеми. Перша проблема полягає в тому, що розробники повинні використовувати гілку develop, а не main. main зарезервована для коду, який випускається у виробництво. Умовно прийнято називати гілку за замовчуванням основною і здебільшого відгалужуватись від неї та зливатись з нею. Оскільки більшість інструментів автоматично використовують основну гілку за замовчуванням, дратує необхідність перемикатися на іншу гілку.

Другою проблемою Git-потоку є складність, яку вносять гілки гарячих виправлень та релізів. Ці гілки можуть бути гарною ідеєю для деяких організацій, але для переважної більшості з них вони є надмірністю. Сьогодні більшість організацій практикують безперервну доставку, що означає, що ваша гілка за замовчуванням може бути розгорнута. Безперервна доставка усуває необхідність у випуску гілок з виправленнями та випусків, включаючи всі пов'язані з ними церемонії. Прикладом такої церемонії є зворотне злиття гілок випуску. Хоча існують спеціалізовані інструменти для вирішення цієї проблеми, вони вимагають документації і

додають складності. Часто розробники припускаються помилок, таких як злиття змін лише до основної гілки, а не до гілки розробників. Причиною цих помилок є те, що потік Git'a занадто складний для більшості випадків використання. Наприклад, багато проектів роблять релізи, але не потребують виправлень.

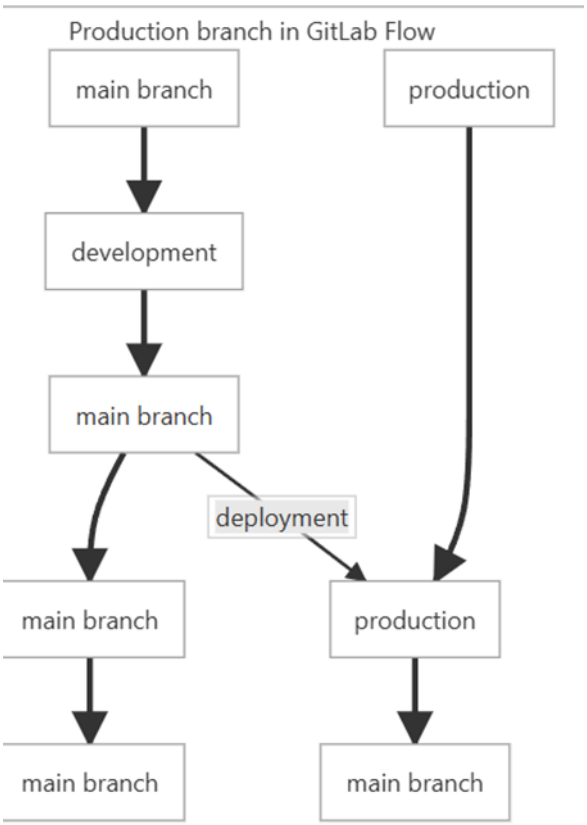


Виробнича гілка з потоком GitLab

Потік GitHub передбачає, що ви можете розгортати у виробництво кожного разу, коли ви об'єднуєте гілку функцій. Хоча в деяких випадках це можливо, наприклад, в SaaS-додатках, є випадки, коли це неможливо, наприклад:

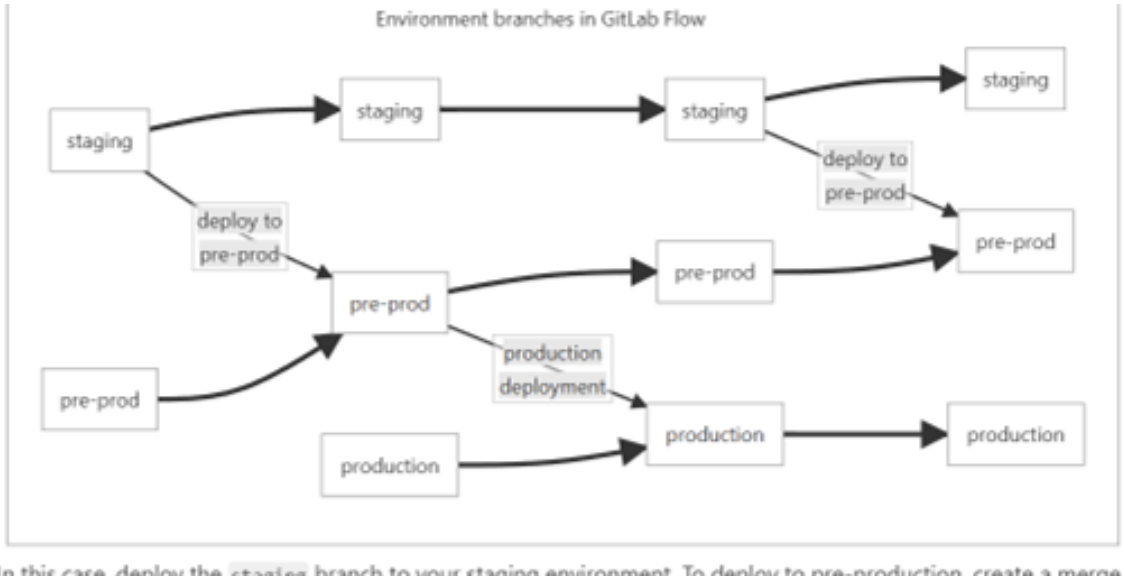
- Ви не контролюєте час випуску. Наприклад, додаток для iOS, який випускається, коли він проходить валідацію в App Store.
- У вас є вікна розгортання - наприклад, робочі дні з 10:00 до 16:00, коли операційна команда працює на повну потужність, - але ви також зливаєте код в інший час.

У цих випадках ви можете створити виробничу гілку, яка відображає розгорнутий код. Ви можете розгорнути нову версію, об'єднавши основну гілку з виробничою. Хоча це не показано на графіку нижче, робота над основною гілкою відбувається так само, як і в потоці GitHub, тобто функціональні гілки об'єднуються в основну.



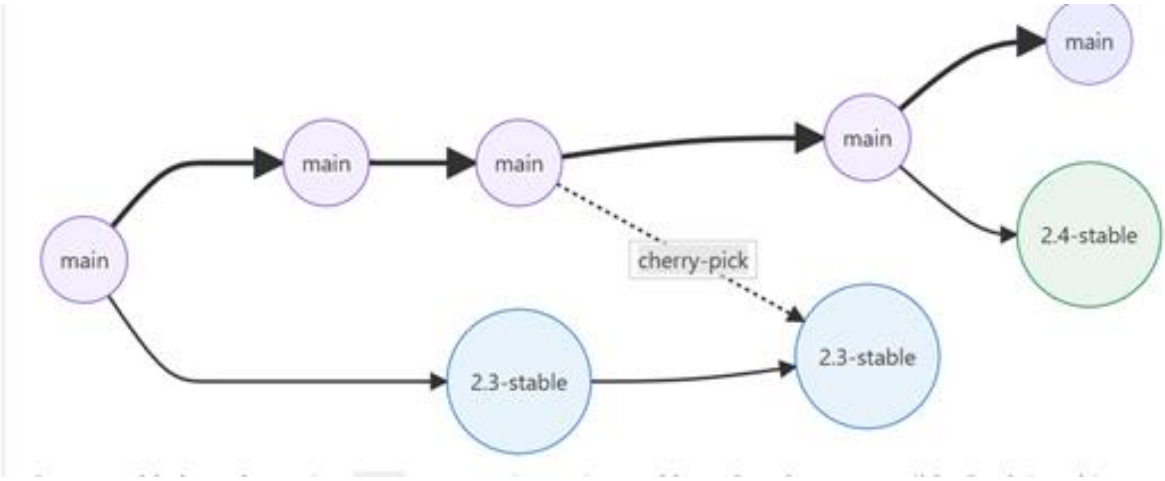
Гілки оточення з потоком GitLab

Було б непогано мати оточення, яке автоматично оновлюється до гілки, що перебуває на стадії розробки. Тільки в цьому випадку назва цього оточення може відрізнятися від назви гілки. Припустимо, у вас є середовище для розробки, препродакшн-середовище і продакшн-середовище:

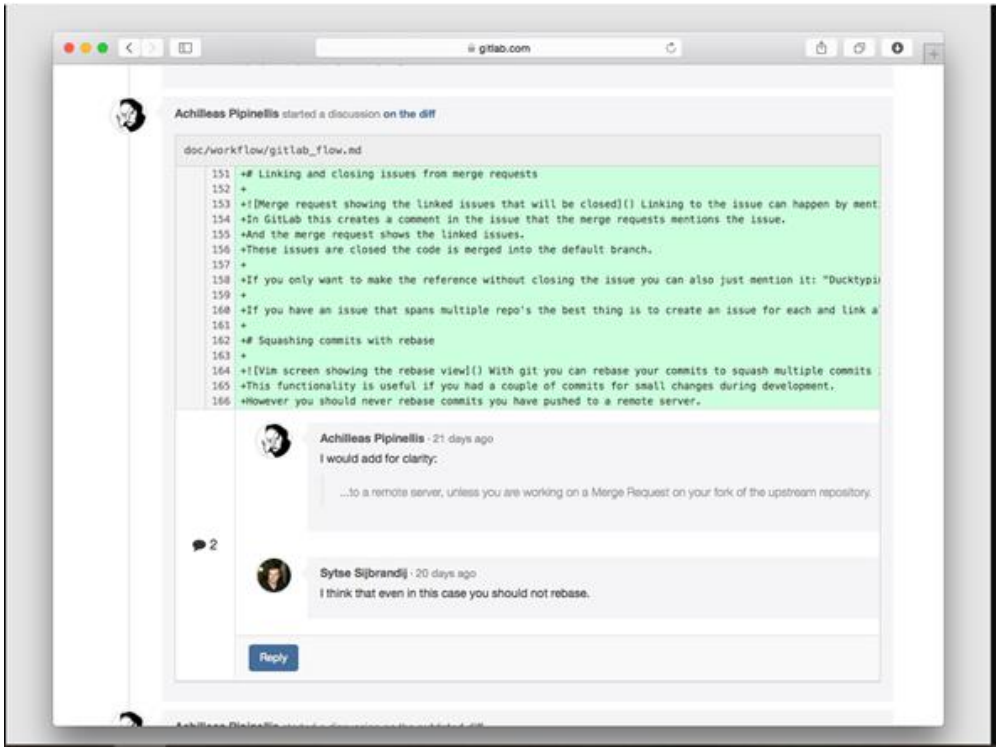


Реліз гілок за допомогою потоку GitLab

Працювати з релізними гілками слід лише тоді, коли вам потрібно випустити програмне забезпечення для зовнішнього світу. У цьому випадку кожна гілка містить мінорну версію, наприклад, 2.3-stable або 2.4-stable:



Об'єднання/витягування запитів з потоком GitLab



https://docs.gitlab.co.jp/ee/topics/img/gitlab_flow_mr_inline_comments.png

Як написати гарне повідомлення про запуск

Повідомлення про коммітування має відображати ваші наміри, а не лише вміст коміту. Ви можете бачити зміни в комміті, тому повідомлення про фіксацію повинно пояснювати, чому ви внесли ці зміни:

This commit message doesn't give enough information

git commit -m 'Improve XML generation'

These commit messages clearly state the intent of the commit

git commit -m 'Properly escape special characters in XML generation'

Прикладом хорошого повідомлення про фіксацію є: «Об'єднайте шаблони, щоб зменшити кількість дублюючого коду в користувацьких поданнях». Слова «змінити», «покращити», «виправити» і «рефакторити» не додають багато інформації до повідомлення про фіксацію.

Properly escape special characters in XML generation.

Issue: gitlab.com/gitlab-org/gitlab/-/issues/1

Щоб додати більше контексту до повідомлення про фіксацію, подумайте про те, щоб додати інформацію про походження зміни. Наприклад, URL випуску GitLab або номер випуску Jira, що містить додаткову інформацію для користувачів, яким потрібен більш детальний контекст про зміну.

Об'єднання комітів за допомогою rebase

У Git'i ви можете використовувати інтерактивний ребаз (rebase -i), щоб об'єднати декілька комітів в один або змінити їх порядок. Ця функція допоможе вам замінити кілька невеликих коммітів одним, або якщо ви хочете зробити порядок більш логічним:

```
pick c6ee4d3 add a new file to the repo
pick c3c130b change readme

# Rebase 168afa0..c3c130b onto 168afa0
#
# 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
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
```

```
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
~
~
~
"~/demo/gitlab-ce/.git/rebase-merge/git-rebase-todo" 20L, 673C
```

Таким чином, **long-living branches у GitLab Flow** - це стабільні, захищені гілки, які підтримують різні етапи життєвого циклу продукту та забезпечують контроль якості і часу релізів у командній розробці.

Джерела:

1. Відео **SUMDU.csTA.S23 L02 VCS**
2. https://graphite.dev/guides/git-flow
3. https://docs.gitlab.co.jp/ee/topics/gitlab_flow.html
4. https://habr.com/ru/companies/softmart/articles/316686/