

*МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ*

*НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ*

*„КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. СІКОРСЬКОГО”*

*НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС*

*„ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ”*

# *Лабораторна робота №1*

## *з курсу «ПРОЕКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ*»

## *Тема:* «***Системи контролю версій SVN, GIT***»

*Варіант №8*

Виконав

студент IV курсу

групи ДА-61

*Калюжний Єгор*

***Київ – 2019***

**Мета роботи:** за допомогою системи контролю версій завантажити коди програми у репозиторій. Відтворити типовий цикл розробки програмного забезпечення з використанням системи контролю версій.

1. **Обрати безкоштовну систему репозиторія для системи контролю версіями, наприклад projectlocker, або інш.**

Для виконання лабораторних робіт було вибрано систему контролю версій Github.

Було створено репозиторій для половини студентів группи, у якому є дві гілки: dev and master. В гільці dev буде проводись вся робота.

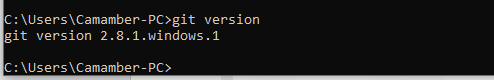
1. **Встановити клієнтське безкоштовне програмне забезпечення для роботи с системою контролю версій (GIT, SVN clients).**

**SVN или Subversion** является свободной централизованной системой управления версиями. Такие системы используются для облегчения работы с информацией, которая постоянно изменяется. Это может быть исходный код программы, скрипта, веб-страницы, сайта, текстового документа и других. Позволяет хранить несколько версий одного и того же документа, с возможностью возврата к более ранним версиям и просмотра информации о том, кто и какие изменения вносил.

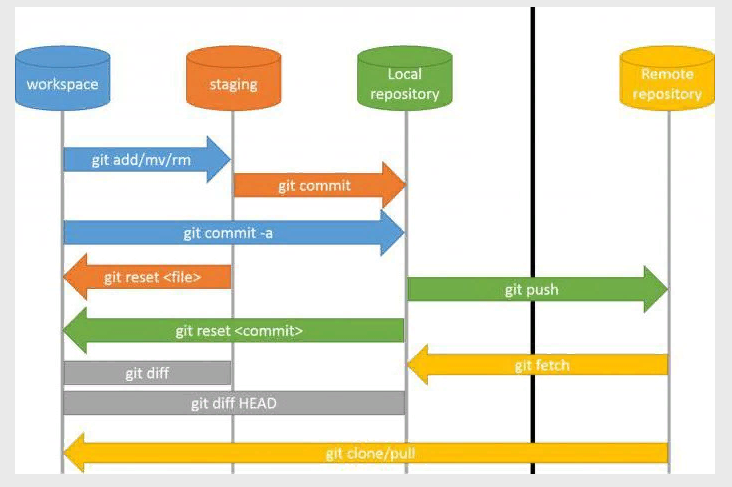
В отличии от систем распределения таких как Git, Subversion хранит информацию в одном едином хранилище, которое может быть или на локальном диске или на сетевом сервере. Простая в использовании и одновременно мощная по функционалу система пользуется популярностью у веб-разработчиков.

**Git** данная система уже считается распределенной, в отличии от предыдущей.

Эта система удобна, когда над одним проектом работают одновременно несколько разработчиков. Гит позволяет обновлять и править файлы сайта учитывая изменения, которые были внесенные другими.



1. **Опис команд, які використовувалися протягом виконання роботи з системою контролю версіями.**



## Основные команды

Всего несколько команд нужно для базового варианта использования Git для ведения истории изменений.

### git add

Команда git add добавляет содержимое рабочей директории в индекс (staging area) для последующего коммита. По умолчанию git commit использует лишь этот индекс, так что вы можете использовать git add для сборки слепка вашего следующего коммита.

Это одна из ключевых команд Git, мы упоминали о ней десятки раз на страницах книги. Ниже перечислены наиболее интересные варианты использования этой команды.

Знакомство с этой командой происходит в главе [Отслеживание новых файлов](https://git-scm.com/book/ru/v2/ch00/r_tracking_files).

О том как использовать git add для разрешения конфликтов слияния написано в главе [Основные конфликты слияния](https://git-scm.com/book/ru/v2/ch00/r_basic_merge_conflicts).

В главе [Интерактивное индексирование](https://git-scm.com/book/ru/v2/ch00/r_interactive_staging) показано как использовать git add для добавления в индекс лишь отдельных частей изменённого файла.

В главе [Деревья](https://git-scm.com/book/ru/v2/ch00/r_tree_objects) показано как эта команда работает на низком уровне, чтобы вы понимали, что происходит за кулисами.

### git status

Команда git status показывает состояния файлов в рабочей директории и индексе: какие файлы изменены, но не добавлены в индекс; какие ожидают коммита в индексе. Вдобавок к этому выводятся подсказки о том, как изменить состояние файлов.

Мы познакомили вас с этой командой в главе [Определение состояния файлов](https://git-scm.com/book/ru/v2/ch00/r_checking_status), разобрали стандартный и упрощённый формат вывода. И хотя мы использовали git status повсеместно в этой книге, практически все варианты использования покрыты в указанной главе.

### git diff

Команда git diff используется для вычисления разницы между любыми двумя Git деревьями. Это может быть разница между вашей рабочей директорией и индексом (собственно git diff), разница между индексом и последним коммитом (git diff --staged), или между любыми двумя коммитами (git diff master branchB).

Мы познакомили вас с основами этой команды в главе [Просмотр индексированных и неиндексированных изменений](https://git-scm.com/book/ru/v2/ch00/r_git_diff_staged), где показали как посмотреть какие изменения уже добавлены в индекс, а какие — ещё нет.

О том как использовать эту команду для проверки на проблемы с пробелами с помощью аргумента --check можно почитать в главе [Правила создания коммитов](https://git-scm.com/book/ru/v2/ch00/r_commit_guidelines).

Мы показали вам как эффективно сравнивать ветки используя синтаксис git diff A...B в главе [Определение применяемых изменений](https://git-scm.com/book/ru/v2/ch00/r_what_is_introduced).

В главе [Продвинутое слияние](https://git-scm.com/book/ru/v2/ch00/r_advanced_merging) показано использование опции -w для скрытия различий в пробельных символах, а также рассказано как сравнивать конфликтующие изменения с опциями --theirs, --ours и --base.

Использование этой команды с опцией --submodule для сравнения изменений в субмодулях показано в главе [Начало работы с подмодулями](https://git-scm.com/book/ru/v2/ch00/r_starting_submodules).

### git commit

Команда git commit берёт все данные, добавленные в индекс с помощью git add, и сохраняет их слепок во внутренней базе данных, а затем сдвигает указатель текущей ветки на этот слепок.

Вы познакомились с основами модели коммитов в главе [Коммит изменений](https://git-scm.com/book/ru/v2/ch00/r_committing_changes). Там же мы продемонстрировали использование опций -a для добавления всех изменений в индекс без использования git add, что может быть удобным в повседневном использовании, и -m для передачи сообщения коммита без запуска полноценного редактора.

В главе [Операции отмены](https://git-scm.com/book/ru/v2/ch00/r_undoing) мы рассказали об опции --amend, используемой для изменения последнего совершённого коммита.

В главе [О ветвлении в двух словах](https://git-scm.com/book/ru/v2/ch00/r_git_branches_overview) мы более подробно познакомились с тем, что делает команда git commit и почему она делает это именно так.

Мы показали вам как подписывать ваши коммиты, используя опцию -S в главе [Подпись коммитов](https://git-scm.com/book/ru/v2/ch00/r_signing_commits).

И наконец мы заглянули внутрь команды git commit в главе [Commit Objects](https://git-scm.com/book/ru/v2/ch00/r_git_commit_objects) и узнали что она делает за кулисами.

### git branch

Команда git branch — это своего рода "менеджер веток". Она умеет перечислять ваши ветки, создавать новые, удалять и переименовывать их.

Большая часть главы [Ветвление в Git](https://git-scm.com/book/ru/v2/ch00/ch03-git-branching) посвящена этой команде, она используется повсеместно в этой главе. Впервые команда branch была представлена в разделе [Создание новой ветки](https://git-scm.com/book/ru/v2/ch00/r_create_new_branch), а большинство таких её фич как перечисление и удаление веток были разобраны в разделе [Управление ветками](https://git-scm.com/book/ru/v2/ch00/r_branch_management).

В главе [Отслеживание веток](https://git-scm.com/book/ru/v2/ch00/r_tracking_branches) мы показали как использовать сочетание git branch -u для отслеживания веток.

Наконец, мы разобрались что происходит за кулисами этой команды в главе [Ссылки в Git](https://git-scm.com/book/ru/v2/ch00/r_git_refs).

### git checkout

Команда git checkout используется для переключения веток и выгрузки их содержимого в рабочую директорию.

Мы познакомились с этой командой в главе [Переключение веток](https://git-scm.com/book/ru/v2/ch00/r_switching_branches) вместе с git branch.

В главе [Отслеживание веток](https://git-scm.com/book/ru/v2/ch00/r_tracking_branches) мы узнали как использовать флаг --track для отслеживания веток.

В главе [Использование checkout в конфликтах](https://git-scm.com/book/ru/v2/ch00/r_checking_out_conflicts) мы использовали эту команду с опцией --conflict=diff3 для разрешения конфликтов заново, в случае если предыдущее решение не подходило по некоторым причинам.

Мы рассмотрели детали взаимосвязи этой команды и git reset в главе [Раскрытие тайн reset](https://git-scm.com/book/ru/v2/ch00/r_git_reset).

Мы исследовали внутренние механизмы этой команды в главе [HEAD](https://git-scm.com/book/ru/v2/ch00/r_the_head).

### git merge

Команда git merge используется для слияния одной или нескольких веток в текущую. Затем она устанавливает указатель текущей ветки на результирующий коммит.

Мы познакомили вас с этой командой в главе [Основы ветвления](https://git-scm.com/book/ru/v2/ch00/r_basic_branching). И хотя git merge встречается в этой книге повсеместно, практически все использования имеют вид git merge <branch> с указанием единственной ветки для слияния.

Мы узнали как делать "сплющенные" слияния (когда Git делает слияние в виде нового коммита, без сохранения всей истории работы) в конце главы [Форк публичного проекта](https://git-scm.com/book/ru/v2/ch00/r_public_project).

В главе [Продвинутое слияние](https://git-scm.com/book/ru/v2/ch00/r_advanced_merging) мы глубже разобрались с процессом слияния и этой командой, включая флаги -Xignore-all-whitespace и --abort, используемый для отмены слияния в случае возникновения проблем.

Мы научились проверять криптографические подписи перед слияниями если ваш проект использует GPG в главе [Подпись коммитов](https://git-scm.com/book/ru/v2/ch00/r_signing_commits).

Ну и наконец в главе [Слияние субдеревьев](https://git-scm.com/book/ru/v2/ch00/r_subtree_merge) мы познакомились со слиянием поддеревьев.

### git stash

Команда git stash используется для временного сохранения всех незакоммиченных изменений для очистки рабочей директории без необходимости коммитить незавершённую работу в новую ветку.

Эта команда практически целиком раскрыта в главе [Прибережение и очистка](https://git-scm.com/book/ru/v2/ch00/r_git_stashing).

### git fetch

Команда git fetch связывается с удалённым репозиторием и забирает из него все изменения, которых у вас пока нет и сохраняет их локально.

Мы познакомились с ней в главе [Получение изменений из удалённого репозитория - Fetch и Pull](https://git-scm.com/book/ru/v2/ch00/r_fetching_and_pulling) и продолжили знакомство в [Удалённые ветки](https://git-scm.com/book/ru/v2/ch00/r_remote_branches).

Мы использовали эту команду в нескольких примерах из главы [Участие в проекте](https://git-scm.com/book/ru/v2/ch00/r_contributing_project).

Мы использовали её для скачивания запросов на слияние (pull request) из других репозиториев в главе [Pull Request Refs](https://git-scm.com/book/ru/v2/ch00/r_pr_refs), также мы рассмотрели использование git fetch для работы с упакованными репозиториями в главе [Создание пакетов](https://git-scm.com/book/ru/v2/ch00/r_bundling).

Мы рассмотрели тонкую настройку git fetch в главe и [Спецификации ссылок](https://git-scm.com/book/ru/v2/ch00/r_refspec).

### git pull

Команда git pull работает как комбинация команд git fetch и git merge, т.е. Git вначале забирает изменения из указанного удалённого репозитория, а затем пытается слить их с текущей веткой.

Мы познакомились с ней в главе [Получение изменений из удалённого репозитория - Fetch и Pull](https://git-scm.com/book/ru/v2/ch00/r_fetching_and_pulling) и показали как узнать, какие изменения будут приняты в случае применения в главе [Просмотр удаленного репозитория](https://git-scm.com/book/ru/v2/ch00/r_inspecting_remote).

Мы также увидели как она может оказаться полезной для разрешения сложностей при перемещении веток в главе [Меняя базу, меняй основание](https://git-scm.com/book/ru/v2/ch00/r_rebase_rebase).

Мы показали как можно использовать только URL удалённого репозитория без сохранения его в списке удалённых репозиториев в главе [Проверка удаленных веток](https://git-scm.com/book/ru/v2/ch00/r_checking_out_remotes).

И наконец мы показали как проверять криптографические подписи полученных коммитов, используя опцию --verify-signatures в главе [Подпись коммитов](https://git-scm.com/book/ru/v2/ch00/r_signing_commits).

### git push

Команда git push используется для установления связи с удалённым репозиторием, вычисления локальных изменений отсутствующих в нём, и собственно их передачи в вышеупомянутый репозиторий. Этой команде нужно право на запись в репозиторий, поэтому она использует аутентификацию.

Мы познакомились с этой командой в главе [Отправка изменений в удаленный репозиторий (Push)](https://git-scm.com/book/ru/v2/ch00/r_pushing_remotes). Там мы рассмотрели основы обновления веток в удалённом репозитории. В главе [Отправка изменений](https://git-scm.com/book/ru/v2/ch00/r_pushing_branches) мы подробнее познакомились с этой командой, а в [Отслеживание веток](https://git-scm.com/book/ru/v2/ch00/r_tracking_branches) мы узнали как настроить отслеживание веток для автоматической передачи на удалённый репозиторий. В главе [Удаление веток на удалённом сервере](https://git-scm.com/book/ru/v2/ch00/r_delete_branches) мы использовали флаг --delete для удаления веток на сервере, используя git push.

На протяжении главы [Участие в проекте](https://git-scm.com/book/ru/v2/ch00/r_contributing_project) мы показали несколько примеров использования git push для совместной работы в нескольких удалённых репозиториях одновременно.

В главе [Публикация изменений в подмодуле](https://git-scm.com/book/ru/v2/ch00/r_publishing_submodules) мы использовали опцию --recurse-submodules чтобы удостовериться, что все субмодули будут опубликованы перед отправкой на проекта на сервер, что может быть реально полезным при работе с репозиториями, содержащими субмодули.

В главе [Other Client Hooks](https://git-scm.com/book/ru/v2/ch00/r_other_client_hooks) мы поговорили о триггере pre-push, который может быть выполнен перед отправкой данных, чтобы проверить возможность этой отправки.

Наконец, в главе [Спецификации ссылок для отправки данных на сервер](https://git-scm.com/book/ru/v2/ch00/r_pushing_refspecs) мы рассмотрели передачу данных с полным указанием передаваемых ссылок, вместо использования распространённых сокращений. Это может быть полезным если вы хотите очень точно указать, какими изменениями хотите поделиться.

### git remote

Команда git remote служит для управления списком удалённых репозиториев. Она позволяет сохранять длинные URL репозиториев в виде понятных коротких строк, например "origin", так что вам не придётся забивать голову всякой ерундой и набирать её каждый раз для связи с сервером. Вы можете использовать несколько удалённых репозиториев для работы и git remote поможет добавлять, изменять и удалять их.

Эта команда детально рассмотрена в главе [Работа с удалёнными репозиториями](https://git-scm.com/book/ru/v2/ch00/r_remote_repos), включая вывод списка удалённых репозиториев, добавление новых, удаление или переименование существующих.

Она используется практически в каждой главе, но всегда в одном и том же виде: git remote add <имя> <URL>.

### git submodule

Команда git submodule используется для управления вложенными репозиториями. Например, это могут быть библиотеки или другие, используемые не только в этом проекте ресурсы. У команды submodule есть несколько под-команд — add, update, sync и др. — для управления такими репозиториями.

Эта команда упомянута и полностью раскрыта в главе [Подмодули](https://git-scm.com/book/ru/v2/ch00/r_git_submodules).

**Высновок:** за допомогою системи контролю версій завантажили коди програми у репозиторій. Відтворили типовий цикл розробки програмного забезпечення з використанням системи контролю версій.