**ОТЧЕТ**

**о выполнении лабораторной работы №5**

**Дисциплина «МДК 03.01 Моделирование и анализ программного обеспечения»**

**Тема: «Основные команды Git»**

**Специальность: 09.02.07 Информационные системы и программирование**

**Квалификация: Специалист по информационным системам**

Выполнил: Хотин М.Ю.

Студент группы: ИС50-11-22

Проверил:

Преподаватель: Кретова Е.М.

Дата: «\_\_\_» \_\_\_\_\_\_\_\_\_ 2023 г.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Теоретические сведения**

**Контроль версий**, также известный как управление исходным кодом, — это практика отслеживания изменений программного кода и управления ими.

**Системы контроля версий** — это программные инструменты, помогающие командам разработчиков управлять изменениями в исходном коде с течением времени.

**Git** — система управления версиями с распределенной архитектурой. В отличие от некогда популярных систем вроде CVS и Subversion (SVN), где полная история версий проекта доступна лишь в одном месте, в Git каждая рабочая копия кода сама по себе является репозиторием. Это позволяет всем разработчикам хранить историю изменений в полном объеме.

Преимущества:

* Git показывает очень высокую производительность в сравнении со множеством альтернатив. Это возможно благодаря оптимизации процедур фиксации коммитов, создания веток, слияния и сравнения предыдущих версий.
* При разработке в Git прежде всего обеспечивается целостность исходного кода под управлением системы. Содержимое файлов, а также объекты репозитория, фиксирующие взаимосвязи между файлами, каталогами, версиями, тегами и коммитами, защищены при помощи криптографически стойкого алгоритма хеширования SHA1.
* Гибкость — одна из основных характеристик Git. Она проявляется в поддержке различных нелинейных циклов разработки, эффективности использования с малыми и крупными проектами, а также совместимости со многими системами и протоколами.

**Порядок выполнения работы**

1. Изучить книгу «Pro Git»
2. Выписать из нее основные команды с описанием их действия для работы с консолью
   1. Создание репозитория в существующем каталоге





Эта команда создаёт в текущем каталоге новый подкаталог с именем .git, содержащий все

необходимые файлы репозитория — структуру Git репозитория. На этом этапе ваш проект

ещё не находится под версионным контролем. Подробное описание файлов, содержащихся

в только что созданном вами каталоге .git, приведено в главе Git изнутри

* 1. Клонирование существующего репозитория



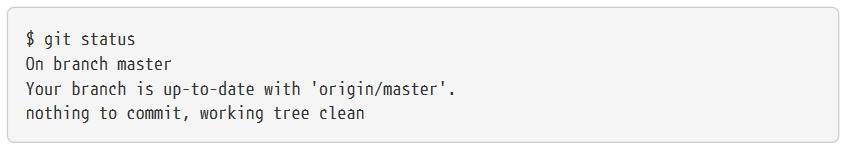
Эта команда создаёт каталог libgit2, инициализирует в нём подкаталог .git, скачивает все

данные для этого репозитория и извлекает рабочую копию последней версии. Если вы

перейдёте в только что созданный каталог libgit2, то увидите в нём файлы проекта,

готовые для работы или использования.

* 1. Определение состояния файлов



Это означает, что у вас чистый рабочий каталог, другими словами — в нём нет

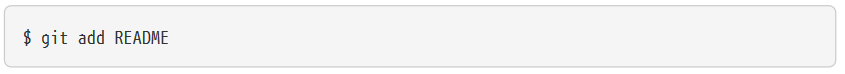
отслеживаемых изменённых файлов. Git также не обнаружил неотслеживаемых файлов, в

противном случае они бы были перечислены здесь. Наконец, команда сообщает вам на

какой ветке вы находитесь и сообщает вам, что она не расходится с веткой на сервере. Пока

что это всегда ветка master, ветка по умолчанию; в этой главе это не важно.

* 1. Отслеживание (добавление) новых файлов



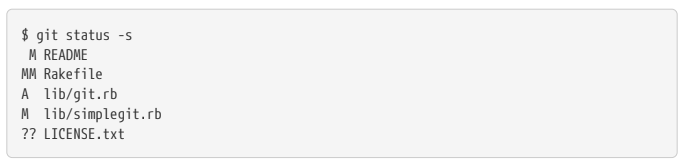
Для того чтобы начать отслеживать (добавить под версионный контроль) новый файл,

используется команда git add

* 1. Индексация изменённых файлов



* 1. Сокращённый вывод статуса



Вывод команды git status довольно всеобъемлющий и многословный. Git также имеет флаг

вывода сокращённого статуса, так что вы можете увидеть изменения в более компактном

виде.

* 1. Игнорирование файлов



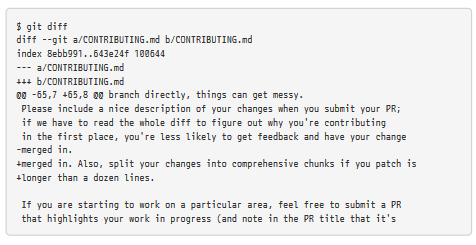
Зачастую, у вас имеется группа файлов, которые вы не только не хотите автоматически

добавлять в репозиторий, но и видеть в списках неотслеживаемых. К таким файлам обычно

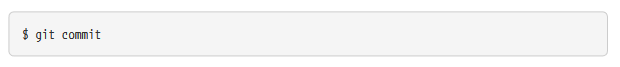
относятся автоматически генерируемые файлы (различные логи, результаты сборки

программ и т. п.).

* 1. Просмотр индексированных и неиндексированных изменений



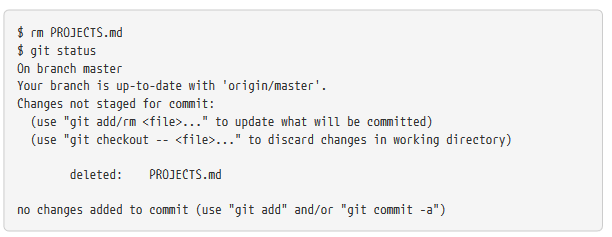
* 1. Коммит изменений



Теперь, когда ваш индекс находится в таком состоянии, как вам и хотелось, вы можете

зафиксировать свои изменения.

* 1. Удаление файлов



Для того чтобы удалить файл из Git, вам необходимо удалить его из отслеживаемых файлов

(точнее, удалить его из вашего индекса) а затем выполнить коммит. Это позволяет сделать

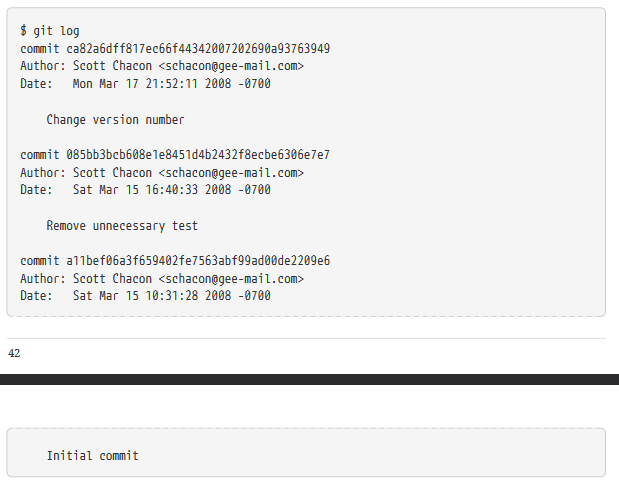
команда git rm, которая также удаляет файл из вашего рабочего каталога, так что в

следующий раз вы не увидите его как «неотслеживаемый»

* 1. Перемещение файлов



* 1. Просмотр истории коммитов



После того, как вы создали несколько коммитов или же клонировали репозиторий с уже

существующей историей коммитов, вероятно вам понадобится возможность посмотреть

что было сделано — историю коммитов. Одним из основных и наиболее мощных

инструментов для этого является команда git log.

* 1. Операции отмены



Отмена может потребоваться, если вы сделали коммит слишком рано, например, забыв

добавить какие-то файлы или комментарий к коммиту. Если вы хотите переделать

коммит — внесите необходимые изменения, добавьте их в индекс и сделайте коммит ещё

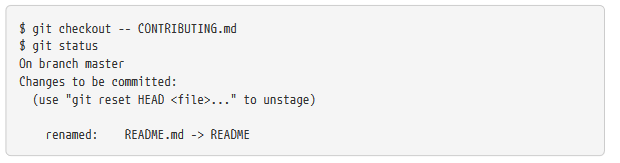
раз, указав параметр --amend

* 1. Отмена индексации файла

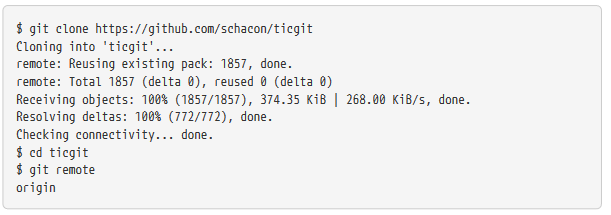


Прямо под текстом «Changes to be committed» говорится: используйте git reset HEAD <file>… для исключения из индекса.

* 1. Отмена изменений в файле



* 1. Просмотр удалённых репозиториев



Просмотр удалённых репозиториев

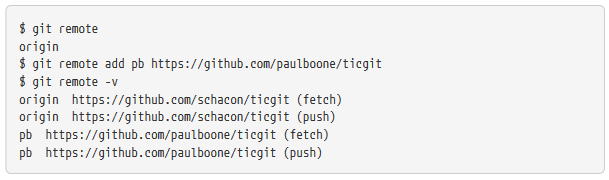
Для того, чтобы просмотреть список настроенных удалённых репозиториев, вы можете

запустить команду git remote. Она выведет названия доступных удалённых репозиториев.

Если вы клонировали репозиторий, то увидите как минимум origin — имя по умолчанию,

которое Git даёт серверу, с которого производилось клонирование

* 1. Добавление удалённых репозиториев



Для того, чтобы добавить удалённый репозиторий и присвоить ему имя (shortname), просто выполните команду git remote add <shortname> <url>

* 1. Получение изменений из удалённого репозитория



Данная команда связывается с указанным удалённым проектом и забирает все те данные проекта, которых у вас ещё нет.

* 1. Отправка изменений в удалённый репозиторий

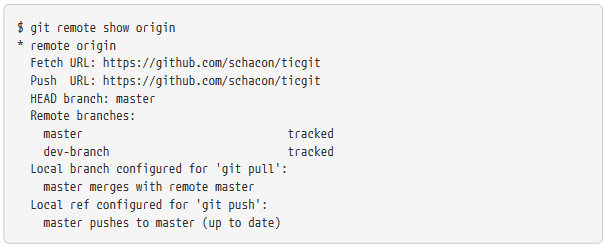


Когда вы хотите поделиться своими наработками, вам необходимо отправить их в

удалённый репозиторий. Команда для этого действия простая: git push <remote-name>

<branch-name>.

* 1. Просмотр удалённого репозитория



Если хотите получить побольше информации об одном из удалённых репозиториев, вы

можете использовать команду git remote show <remote>.

* 1. Удаление и переименование удалённых репозиториев



Для переименования удалённого репозитория можно выполнить git remote rename

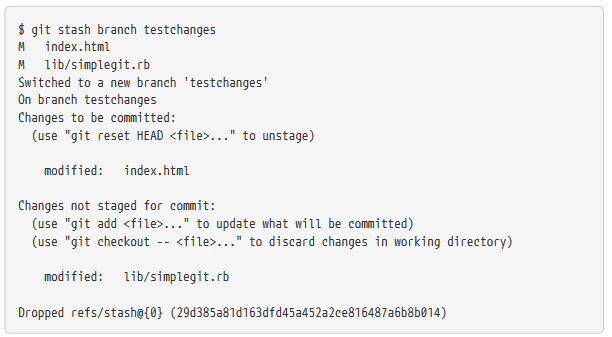
* 1. Просмотр списка тегов



Просмотреть список имеющихся тегов в Git можно очень просто. Достаточно набрать

команду git tag (параметры -l и --list опциональны)

* 1. Создание ветки



@@ -16,6 +16,10 @@ class SimpleGit

return `#{git\_cmd} 2>&1`.chomp

end

end

+

+ def show(treeish = 'master')

+ command("git show #{treeish}")

+ end

end

test

Stash this hunk [y,n,q,a,d,/,e,?]? y

Saved working directory and index state WIP on master: 1b65b17 added the index file

Создание ветки из припрятанных изменений

Если вы спрятали некоторые изменения, оставили их на время, а сами продолжили

работать в той же ветке, у вас могут возникнуть проблемы с восстановлением наработок.

Если восстановление будет затрагивать файл, который уже был изменён с момента

сохранения наработок, то вы получите конфликт слияния и должны будете попытаться

разрешить его. Если вам нужен более простой способ снова протестировать припрятанные

изменения, вы можете выполнить команду git stash branch, которая создаст для вас новую

ветку, перейдёт на коммит, на котором вы были, когда прятали свои наработки, применит

на нём эти наработки и затем, если они применились успешно, удалит эти припрятанные

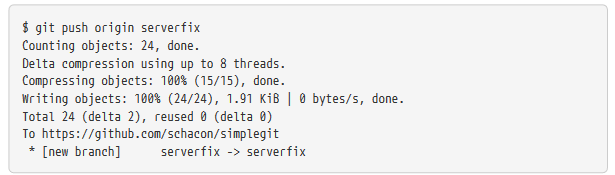
изменения

* 1. Переименование ветки



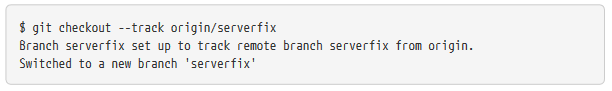
Переименуйте ветку локально с помощью команды git branch --move:

* 1. Отправка изменений (ветка на удаленный сервер)



Выполните команду git push <remote> <branch>

* 1. Отслеживание веток



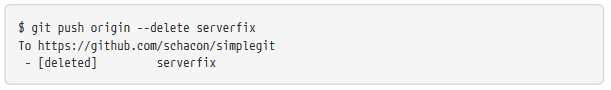
* 1. Получение изменений (ветка на удаленный сервер)

Команда git fetch получает с сервера все изменения, которых у вас ещё нет, но не будет

изменять состояние вашей рабочей копии. Эта команда просто получает данные и

позволяет вам самостоятельно сделать слияние.

* 1. Удаление веток на удалённом сервере



Вы можете

удалить ветку на удалённом сервере используя параметр --delete для команды git push.