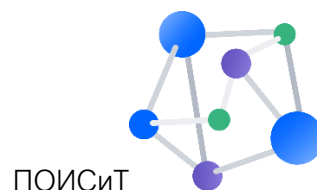
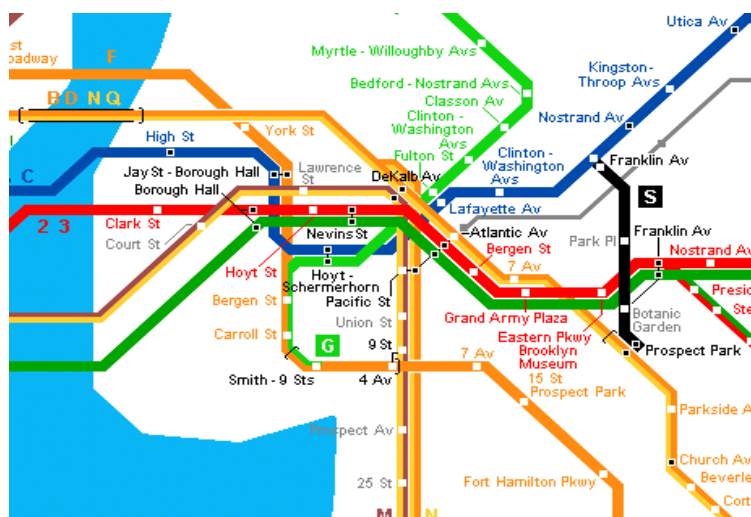


ЛАБОРАТОРНАЯ РАБОТА № 1₆
СИСТЕМЫ КОНТОРОЛЯ ВЕРСИЙ
GitBash



Лабораторная работа №16

СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ

Цель работы

Ознакомиться с системами контроля версий для разработки программного обеспечения. Освоить способы организации работы в системах контроля версий. Получить навыки командной работы.

<i>Норма времени выполнения:</i>	<i>3 академических часа.</i>
<i>Оценка работы:</i>	<i>3 зачётных единицы.</i>

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ

Интерфейс командной строки

По сути, внутреннее устройство Git представляет собой набор служебных программ командной строки, предназначенных для выполнения в *Unix*-подобных средах. Современные операционные системы, такие как *Linux* и *macOS*, имеют встроенные терминалы *Unix*-систем командной строки. Благодаря этому они особенно удобны для работы с Git. В *Microsoft Windows* используется командная строка Windows (*git-cmd*), отличная от терминала *Unix*-систем.

В средах Windows система Git часто упаковывается в виде части высокоуровневого приложения с графическим интерфейсом. Графические интерфейсы для Git могут абстрагировать и скрывать базовые компоненты системы контроля версий, которая лежит в основе. Это удобно для новичков в Git, чтобы они могли быстро внести свой вклад в проект. Но когда требования повышаются и предполагается работа с остальными членами команды, необходимо понимать принципы работы исходных методов Git. Тогда может быть важно отказаться от версии с графическим интерфейсом в пользу инструментов командной строки. Интерфейс терминала Git предлагается в виде приложения Git Bash.

Git Bash – это приложение для сред Microsoft Windows, эмулирующее работу командной строки Git. Командная строка или как её часто называют – *консоль*, является непосредственным, прямым, самым быстрым, наиболее гибким, и в некоторых средах единственным способом общения программиста с

компьютером. В программировании понятие – Оболочка (Shell) представляет собой приложение терминала для взаимодействия с операционной системой с помощью письменных (консольных) команд.

Аббревиатура «Bash» происходит от Bourne Again Shell (Возрожденный Shell). В настоящее время Bash является самой популярной оболочкой, используемой и предустановленной по умолчанию в Linux и macOS. Таким Git Bash представляет собой пакет, который устанавливает в операционную систему Windows оболочку Bash, содержащие некоторые распространенные утилиты Bash и систему Git. Git Bash поддерживает те же операции, что и стандартная оболочка Bash. Полезно изучить основные примеры использования Bash.

Учимся работать с терминалом

Набирать такие команды с клавиатуры посимвольно неудобно и долго. Нам в помощь предусмотрены различные вспомогательные приемы. Для выполнения ряда действий в терминале предусмотрен список горячих клавиш, делающих работу в нем еще удобнее. Для ускорения и удобства вашей работы с терминалом изучим некоторые приемы.

Горячие клавиши. Большинство комбинаций клавиш стандартны для «командной строки Linux», но часть из этих комбинаций специфичны для разных командных интерпретаторов (bash, dach, zsh, fish и т.п.), в них могут быть небольшие отличия в работе. Также некоторые клавиши могут быть настроены и перехватываться графической оболочкой (терминалом), в которой запущен командный интерпретатор.

Таким образом часть комбинаций относятся к «настройкам терминала». А другая часть к командному интерпретатору (*sh).

Клавиши из настроек терминала можно посмотреть, выполнив команду:

```
$ stty -a
```

Чтобы узнать какие клавиши привязаны к каким действиям редактирования командной строки — для этого можно воспользоваться командой «**bind -P**».

Привычные в Windows сочетания клавиш **Ctrl+C** и **Ctrl+V** в терминале работают по-другому. Они могут остановить работу UNIX-терминала. Для копирования и вставки есть добрая пара **Ctrl+Insert** с **Shift+Insert** или же сочетания с **Shift**: **Ctrl+Shift+C** и **Ctrl+Shift+V** соответственно.

Если вы хотите прервать работу уже запущенной в терминале программы, для этого можно использовать сочетание горячих клавиш **Ctrl+C**.

Другое управляющие сочетания, например **Ctrl+D** (окончание ввода данных) посылает сигнал конца файла запущенному приложению, а без запущенных утилит делает тоже, что и терминальная команда `exit`. Таким образом вы можете быстро закрыть терминал.

Также дополнительную информацию о горячих клавишах можно легко получить в Интернете.

Профессионалы *не копируют команды*, а набирают их вручную. Для ускорения на помощь приходит великолепное свойство терминала – автодополнение.

Автодополнение. Наберите в терминале первые символы нужной вам команды, а затем клавишу `Tab`. Терминал автоматически дополнит за вас команду. Если клавишу `Tab` нажать два раза подряд, терминал представит список похожие команды на выбор.

Автодополнение в терминале работает практически везде, и не только для команд, но также для их аргументов и имён файлов. *Используйте ее постоянно.*

История введённых команд. Терминал хранит историю введённых пользователем команд, которую вы можете листать в реальном режиме стрелками `↑` «вверх» и `↓` «вниз» на клавиатуре. *Это очень удобно* для повторного исполнения введённых ранее команд. А посмотреть всю историю можно командой:

```
$ history
```

У каждой команды в истории есть номер, выполнить снова команду с определённым номером можно, набрав в терминале восклицательный знак «`!`» и без пробела номер нужной команды, например, «`!73`». А повторить предыдущую набранную команду можно просто написав два восклицательных знака «`!!`».

Итак, мы рассмотрели только основные и первые необходимые приемы работы с терминалом, совершенствуйте свою работу с терминалом и эти навыки (*скилы*) вам пригодятся в дальнейшей профессиональной деятельности.

Команды Git Bash

Git Bash поставляется с дополнительными командами, которые можно найти в эмулируемом каталоге `/usr/bin`. В вашей системе это будет путь: `C:\Program Files\Git\usr\bin`. В целом Git Bash предоставляет достаточно функциональную оболочку для Windows. В пакет также входят следующие команды оболочки, рассмотрение которых не входит в задачи данной работы: `ssh`, `scp`, `cat`, `find`.

Помимо уже рассмотренного набора команд Bash, пакет Git Bash содержит полный набор всех команд Git. Каждый раз, когда вы набираете команду git в начале командной строки терминала Git Bash, терминал обращается к программе git вот по этому пути: C:\Program Files\Git\bin\git. А программист, абстрагируясь от всего этого понимает так: слово git в начале, это признак того, что он работает с Git-ом.

Подробнее по командам Git смотрите на соответствующих страницах документации: git clone, git commit, git checkout, git push и другим командам.

Как получить помощь?

Если вам нужна помощь при использовании Git, есть три способа открыть страницу руководства по любой команде Git:

```
$ git help <команда>  
$ git <команда> --help  
$ man git-<команда>
```

Эти команды хороши тем, что ими можно пользоваться всегда, даже *без подключения к сети*.

Например, так можно открыть руководство по команде git config:

```
$ git help config
```

Если нужно посмотреть только список опций, чтобы не читать полную документацию по команде, вы можете использовать опцию **-h** для вывода *краткой инструкции* по использованию, например:

```
$ git add -h
```

Заключение

Теперь вы умеете выполнять все базовые локальные операции с Git: создавать или клонировать репозиторий, вносить изменения, индексировать и фиксировать эти изменения, а также просматривать историю всех изменений в репозитории.

<https://git-scm.com/book/ru/v2/Введение-Командная-строка>

Best of LUCK with it, and remember to HAVE FUN while you're learning :)
Sergey Stankevich



УПРАЖНЕНИЯ

Упражнение №1. Работа с Git в терминале

1. Создание репозитория, коммитов и веток

Для начала и удобства откроем терминал сразу в нужном каталоге. В Windows всё достаточно просто – клик правой кнопкой мыши на каталоге и выбор *Git Bash Here*:



Ура! Получилось!

Создадим новую песочницу для работы с Git Bash, и назовем ее, например, *GitSandbox2_Terminal*. Снова откроем терминал (*Рисунок 1*).

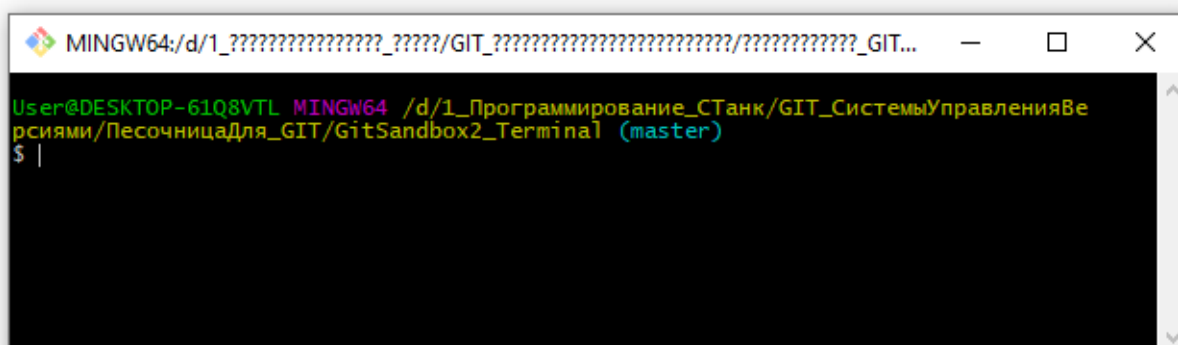


Рисунок 1 -

Однако, поясните почему так много вопросительных знаков в строке заголовка окна терминала?

Для создания нового репозитория достаточно просто зайти в папку проекта и набрать:

```
$ git init
```

Затем выполните ряд консольных команд, как показано на *Рисунок 2*, тем самым проверив содержимое директории до и после создания репозитория.

В итоге был создан пустой репозиторий-папка «.git» в корне проекта, в которой и будет собираться вся информация о дальнейшей работе.

Предположим, в проекте уже существует несколько файлов, а если нет, то добавьте из предыдущего проекта, и эти файлы требуется проиндексировать командой `git add`:

```
git add .
```

```

MINGW64: d:/1_????????????????_????/GIT_????????????????????/????????????_GIT/GitSandbox2_Terminal
User@DESKTOP-61Q8VTL MINGW64 /d/1_Программирование_СТанк/GIT_СистемыУправленияВерсиями/ПесочницаДля_GIT/GitSandbox2_Terminal
$ cd GitSandbox2_Terminal/

User@DESKTOP-61Q8VTL MINGW64 /d/1_Программирование_СТанк/GIT_СистемыУправленияВерсиями/ПесочницаДля_GIT/GitSandbox2_Terminal
$ ls
init          instaweb      interpret-trailers

User@DESKTOP-61Q8VTL MINGW64 /d/1_Программирование_СТанк/GIT_СистемыУправленияВерсиями/ПесочницаДля_GIT/GitSandbox2_Terminal
$ git init
Initialized empty Git repository in D:/1_Программирование_СТанк/GIT_СистемыУправленияВерсиями/ПесочницаДля_GIT/GitSandbox2_Terminal/.git/

User@DESKTOP-61Q8VTL MINGW64 /d/1_Программирование_СТанк/GIT_СистемыУправленияВерсиями/ПесочницаДля_GIT/GitSandbox2_Terminal (master)
$ ls -l
total 0

User@DESKTOP-61Q8VTL MINGW64 /d/1_Программирование_СТанк/GIT_СистемыУправленияВерсиями/ПесочницаДля_GIT/GitSandbox2_Terminal (master)
$ ls -a
./ ../ .git/

User@DESKTOP-61Q8VTL MINGW64 /d/1_Программирование_СТанк/GIT_СистемыУправленияВерсиями/ПесочницаДля_GIT/GitSandbox2_Terminal (master)
$ ls -al
total 4
drwxr-xr-x 1 User 197121 0 anp 29 00:23 ./
drwxr-xr-x 1 User 197121 0 anp 29 00:20 ../
drwxr-xr-x 1 User 197121 0 anp 29 00:23 .git/

User@DESKTOP-61Q8VTL MINGW64 /d/1_Программирование_СТанк/GIT_СистемыУправленияВерсиями/ПесочницаДля_GIT/GitSandbox2_Terminal (master)
$ |
  
```

Рисунок 2 – по

Внесем изменения в репозиторий:

```
git commit -m "Первоначальный коммит"
```

Однако коммит может не получиться, и `git` выдаст фатальную ошибку:

```

$ git commit -m "Pershy kammit"

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'User@DESKTOP-61Q8VTL.(none)')
  
```

Это значит, что при начальных настройках `git` для работы вы не указали очень важные данные.

Используйте следующие команды:

```
git config --global user.email "you@example.com"  
git config --global user.name "Your Name"
```

Git всегда хочет знать своих «героев». Я думаю, вам тоже будет интересно кто внес изменения в ваш проект.

[Git Error - fatal: Not a valid object name: 'master' - Hire Amir](#)

Имеется готовый репозиторий с единственной веткой “master”.

Подробно о создании репозитория можно посмотреть здесь:

<https://git-scm.com/book/ru/v2/Основы-Git-Создание-Git-репозитория>

2. Ветвление

Допустим, потребовалось разработать какой-то новый функционал. Для этого создадим новую ветку:

```
git branch new-feature
```

И переключимся на нее:

```
git checkout new-feature
```

Вносим необходимые изменения, после чего смотрим на них, индексируем и коммитим:

```
git status
```

```
git add .
```

```
git commit -m "new feature added"
```

Теперь у нас есть две ветки, одна из которых (master) является условно основной (технически же ничем не отличается). Переключаемся на нее и включаем изменения (сливаем с другой веткой):

```
git checkout master
```

```
git merge new-feature
```

Веток может быть неограниченное количество, из них можно создавать патчи, определять **diff** с любым из совершенных коммитов.

Теперь предположим, что во время работы выясняется: нашелся небольшой баг, требующий срочного внимания. Есть два варианта действий в таком случае. Первый состоит из создания новой ветки, переключения в нее, слияния с основой. Второй, команда **git stash**. Она сохраняет все изменения по сравнению с последним коммитом во временной ветке и сбрасывает состояние кода до исходного:

```
git stash
```

Команда `git stash` (тайник, закладка) предназначена для того, чтобы поместить текущие изменения, сделанные в файлах, в отдельное хранилище, и вернуть файлы к исходному состоянию.

То есть `git stash` «прячет» изменения в файлах и сохраняет эти изменения отдельно, чтобы потом можно было их вернуть.

Например, вы сделали какие-нибудь изменения в файлах и хотите переключиться на другую ветку, но чтобы там не было этих текущих изменений. С помощью команды `git stash` можно спрятать такие изменения. Изменения помещаются в отдельное хранилище — в стек, а вы можете спокойно переключиться на другую ветку.

Всё, что вы спрячете с помощью `git stash`, попадает в отдельный список. Затем вы можете извлекать оттуда то, что туда спрятали — ваши «прятанья».

Команда `git stash` имеет собственные вспомогательные команды. Подробно можно почитать в документации, например здесь:

[Команда Git stash. Как прятать изменения в Git \(pingvinus.ru\)](http://pingvinus.ru)

Исправляем баг и накладываем поверх произведенные до того действия (проводим слияние с веткой `stash`):

```
git stash apply
```

На самом деле таких «закладок» (`stashes`) может быть сколько угодно, для этого они просто нумеруются.

При такой работе появляется необычная гибкость; но среди всех этих веточек теряется понятие *ревизии*, характерное для линейных моделей разработки. Вместо этого каждый из коммитов (строго говоря, каждый из объектов в репозитории) однозначно определяется хэшем. Естественно, это несколько неудобно для восприятия, поэтому разумно использовать механизм тэгов для того, чтобы выделять ключевые коммиты:

```
git tag
```

просто именуется последний коммит;

```
git tag -a
```

также дает имя коммиту, и добавляет возможность оставить какие-либо комментарии (аннотацию). По этим тегам можно будет в дальнейшем обращаться к истории разработки.

Плюсы такой системы очевидны. Вы получаете возможность «колдовать» с кодом как вашей душе угодно, а не подчиняться логике системы контроля версий. Вы можете разрабатывать параллельно несколько «фишек» в собственных ветках, исправлять баги, чтобы затем все это сливать в единую кашу главной ветки. Папки **.git** с репозиторием удобным и быстрым способом создаются, удаляются или копируются куда и как угодно.

Стратегия:

Гораздо удобней такую легковесную систему использовать для хранения версий документов, файлов настроек и т.д. К примеру, настройки и плагины для текстового редактора «Емакс» можно хранить в директории `~/site-lisp`, и держать в том же месте репозиторий. Рабочее пространство можно организовать в виде двух веток: `work` и `home`; иногда бывает удобно похожим образом управлять настройками в `/etc`. Таким образом, каждый Ваш личный проект любых документов может находиться под управлением `git`. Используйте `Git` где угодно и, как угодно.

3. Просмотр истории

Чтобы уменьшить объем файла *main.c*, вынесем код, спрашивающий имя пользователя, в отдельную функцию. Также можно вынести эту функцию в отдельный файл. Хранилище теперь содержит файлы *main.c*, *askname.c*, и *askname.h*.

```
/* main.c */
#include <stdio.h>

#include "askname.h"

int main(int argc, char **argv)
{
    char first[255], last[255];
    askname(first, last);
    printf("Hello, %s %s!\n", first, last);
    return 0;
}
```

```
/* askname.h */
void askname(char *first, char *last);
```


4. Отмена изменений (revert или reset)

Для отмены внесенных изменений до состояния последней фиксации:

Меню: Состояния – Отменить изменения.

Для отката к конкретной фиксации (reset):

Меню *Репозиторий – История ветки* (выбрать нужное состояние и в контекстном меню выбрать: «Установить для ветки это состояние»). Возможны варианты (мягкий – изменение только индекса или жесткий – изменения в индексе и на диске).

5. Публикация изменений на удалённом сервере

Зарегистрируйтесь на сайте <https://github.com/> (укажите логин, почту и пароль, дальше выберите бесплатный доступ). Github – не единственный доступный удаленный репозиторий Git. Это, однако, самое популярное решение, и мы будем использовать его в качестве примера.

Создайте новый репозиторий. Для этого на сайте выберите *Create new repository*, укажите его название, тип (публичный) и нажмите *Create repository*. После создания репозитория будет отображено его имя - адрес (в формате HTTPS) и команды для работы с ним в режиме командной строки (для желающих).

Далее из каталога репозитория на локальном ПК вызовите *Git Gui*, выберите меню: *Внешние репозитории → Добавить*, в новом окне укажите псевдоним удаленного репозитория (любой, на Рисунок 3 это *Test1*) и его адрес (<https://github.com/ivanov/test2.git>, где *ivanov* – логин пользователя сайта, *test2* – название репозитория на сайте). Рекомендуется адрес скопировать с сайта.

Затем выбрать в меню *Внешние репозитории → Отправить*, указать псевдоним удаленного репозитория и отправляемую ветку, нажать «Отправить» (см. Рисунок 4). При запросе ввести логин и пароль, с которыми регистрировались на сайте.

Если все успешно (см. Рисунок 5), то на сайте перейти в репозиторий и просмотреть его содержимое. Можно просматривать историю изменений репозитория, содержимое фиксаций, изменения.

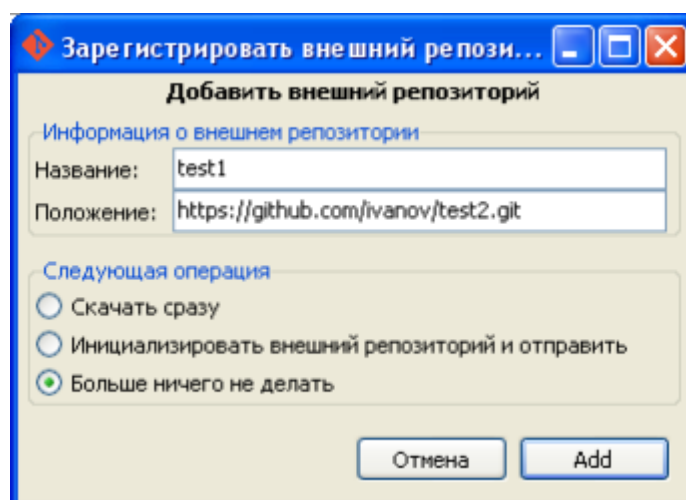


Рисунок 3 – Добавление внешнего репозитория.

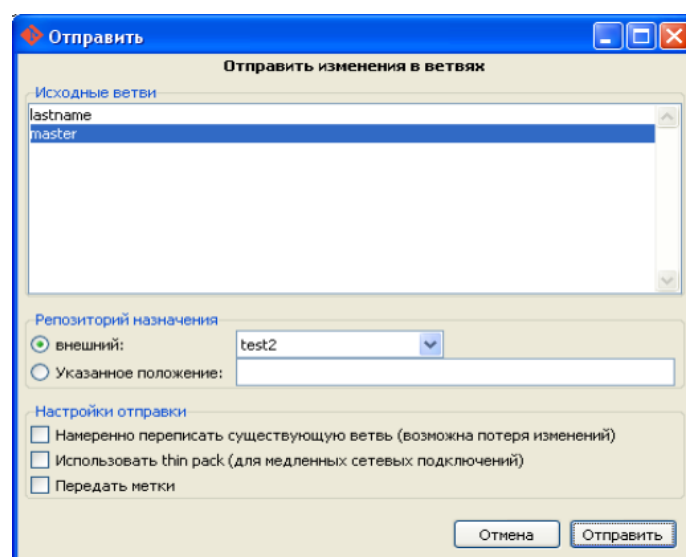


Рисунок 4 – Отправка изменений в ветвях.

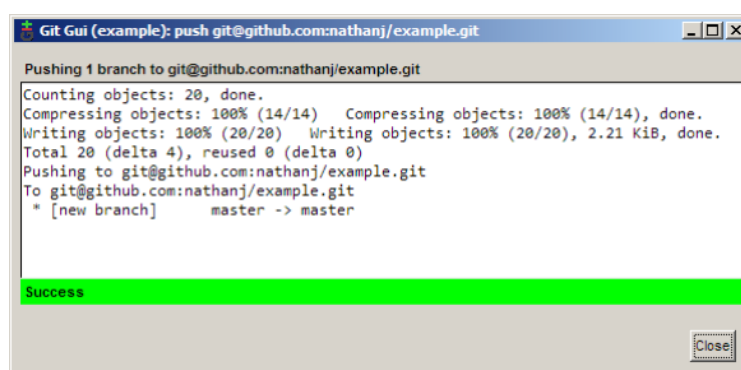


Рисунок 5 – Подтверждение отправки.

Подробнее как выполнить задание можно прочитать здесь:

<https://coderlessons.com/articles/veb-razrabotka-articles/git-na-windows-dlia-novichkov>

6. Получение изменений с удалённого сервера

Для получения копии удаленного репозитория нужно открыть проводник, щелкнуть правой кнопкой мыши и из контекстного меню выбрать Git GUI. Вам будет показан диалог создания (см. Рисунок 27).

Выбрать Clone (Клонировать существующий репозиторий). Будет открыт диалог клонирования (см. Рисунок 28). В качестве источника укажите удаленный репозиторий (его адрес), в качестве приемника – новый каталог.

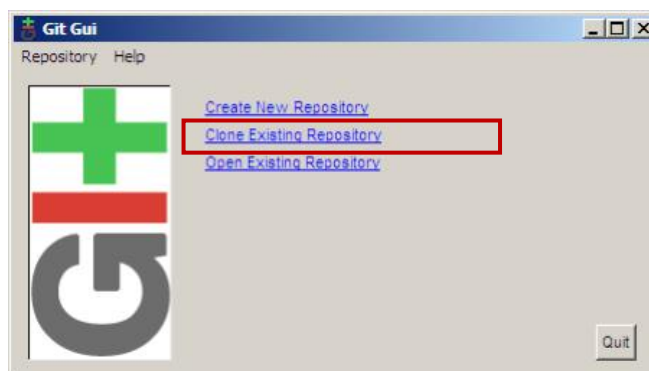


Рисунок 6 – д

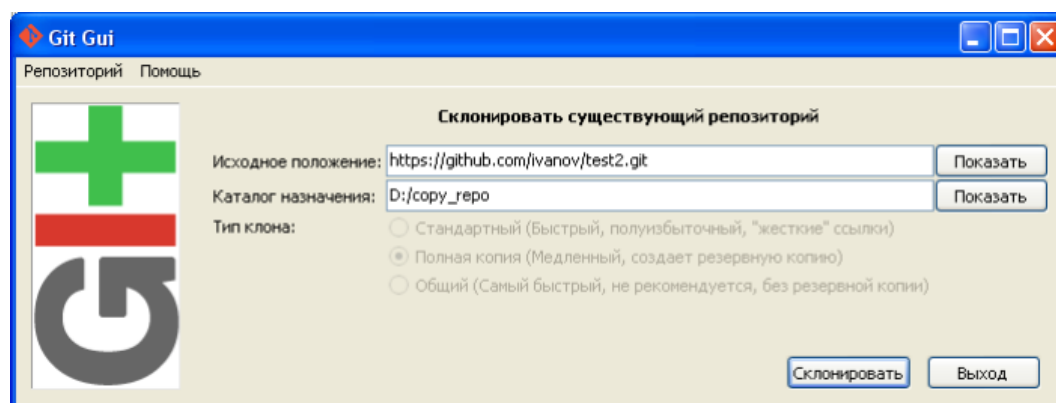


Рисунок 7 –

Нажать «Клонировать». Если все успешно, то откроется среда Git Gui, в которой возможно посмотреть содержимое файлов и историю изменений.

Дальше можно работать над проектами независимо. После внесения изменений и их фиксации отправим изменения обратно на сервер (меню Внешние репозитории → Отправить, указать псевдоним удаленного репозитория и отправляемую ветку, нажать «Отправить»).

Предварительно необходимо разрешение владельца репозитория на внесение изменений. Для этого владелец проекта на сайте выбирает (сверху вверху) New Collaborator, откроется страница участников проектов. На ней нужно указать имя добавляемого участника (он должен быть найден по имени или его

части) и нажать Add Collaborator. Новый участник будет добавлен в список участников.

Все участники проекта могут вносить свои изменения, используя адрес репозитория, свои логин и пароль.

Код, залитый на github, могут смотреть и скачивать другие люди и использовать программу. Один человек, Фред, скопировал к себе репозиторий (сделал вилку – Fork) и добавил в него собственные изменения. Теперь, когда он добавил свой код, мы можем «перетянуть» его изменения в своё хранилище.

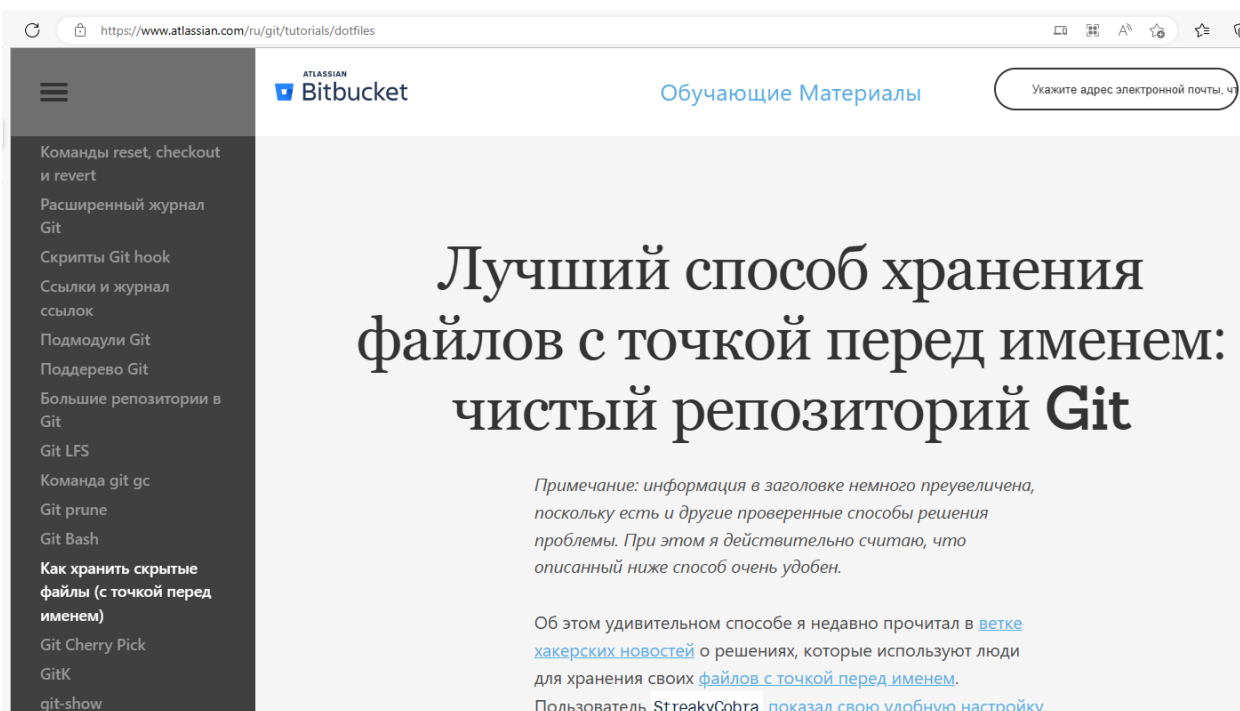
Для получения изменений Фреда, необходимо использовать Remote → Fetch from → fred (Внешние репозитории → Получить) (см. Рисунок 29).

We hope you enjoy working with GIT!



Упражнение №2. Лучший способ хранения файлов с точкой перед именем: чистый репозиторий Git

[Хранение файлов с точкой перед именем — чистый репозиторий Git \(atlassian.com\)](https://www.atlassian.com/ru/git/tutorials/dotfiles)



The screenshot shows a web browser window with the URL <https://www.atlassian.com/ru/git/tutorials/dotfiles>. The page is from Atlassian Bitbucket and is titled "Обучающие Материалы". The main heading is "Лучший способ хранения файлов с точкой перед именем: чистый репозиторий Git". Below the heading, there is a note in Russian: "Примечание: информация в заголовке немного преувеличена, поскольку есть и другие проверенные способы решения проблемы. При этом я действительно считаю, что описанный ниже способ очень удобен." This is followed by a paragraph: "Об этом удивительном способе я недавно прочитал в [ветке хакерских новостей](#) о решениях, которые используют люди для хранения своих [файлов с точкой перед именем](#). Пользователь StreakyCobra [показал свою удобную настройку](#),". On the left side, there is a sidebar menu with various Git-related links, including "Команды reset, checkout и revert", "Расширенный журнал Git", "Скрипты Git hook", "Ссылки и журнал ссылок", "Подмодули Git", "Поддерво Git", "Большие репозитории в Git", "Git LFS", "Команда git gc", "Git prune", "Git Bash", "Как хранить скрытые файлы (с точкой перед именем)", "Git Cherry Pick", "GitK", and "git-show".

We hope you enjoy working with GIT!



ЗАДИНИЯ

Выполните вышеуказанные упражнения, покажите результаты и ход выполнения укажите в отчете подтверждая скриншотами.

Выберите приемлемую вам систему контроля версий, зарегистрируйтесь в ней. Создайте команду разработчиков, добавив членов команды. Создайте проект (предприятие), и начните работу.

Название проекта должно содержать атрибуты нашей дисциплины, номера лабораторной работы, имена разработчиков и т.п. Ваш проект должен легко идентифицироваться среди тысяч других.

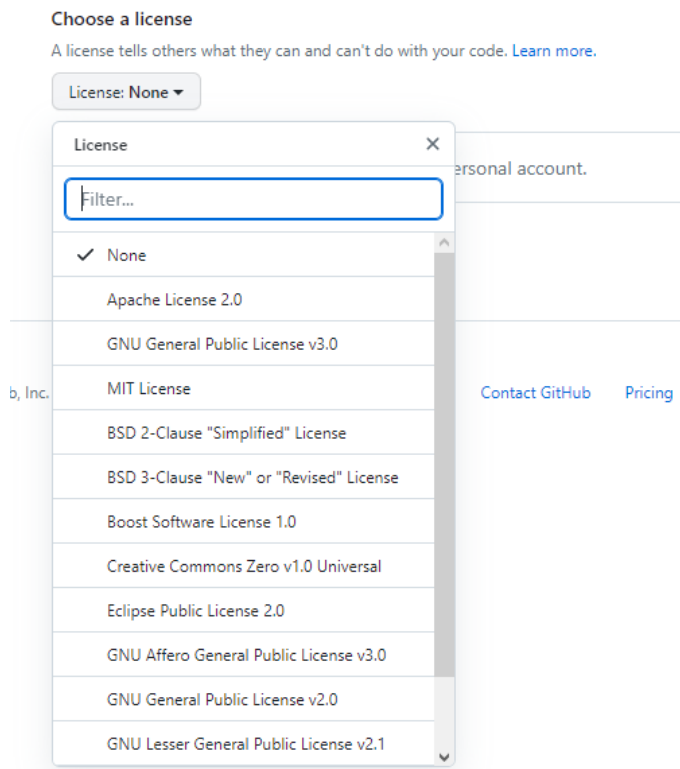
Желательно ограничить доступ к проекту, но у преподавателя должна быть возможность просматривать и работать с этим проектом.

Напишите отчет. Ключевые моменты вашей работы должны быть отражены с помощью скриншотов. Отчет должен храниться в проекте в системе контроля версий.

Индивидуальное задание

Изучите типы лицензий и сделайте реферат по типу лицензии в соответствии с вариантами заданий, представленными ниже:

№ варианта	Тип лицензии
1.	
2.	
3.	
4.	
5.	



«Easy things should be easy and hard things should be possible»
«Простые вещи должны быть простыми, а сложные вещи должны быть
возможными»



КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое системы контроля версий?
2. В каких сферах деятельности могут использоваться системы контроля версий?
3. Назовите наиболее популярные СКВ для разработки ПО.
4. Какую СКВ вы выбрали для работы и почему?

5. Какие виды и типы контроля версий существуют?
6. Что такое GIT?
7. Знаете ли вы историю создания GIT?
8. Кто является создателем GIT?
9. Как слово «git» переводится с английского языка?
10. Укажите основные команды, компоненты и процедуры работы в СКВ.
11. Что такое система управления версиями?
12. Как создать репозиторий?
13. Что такое commit, и почему Git называют системой управления коммитами?
14. Как создать ветку?
15. Как провести слияние? Как разрешить конфликт и что это такое?
16. Как зафиксировать изменения?
17. Как провести откат? Различия в reset и revert, мягкий и жесткий reset.
18. Какова последовательность действий при работе с локальным репозиторием?
19. Какова последовательность действий при работе с удаленным репозиторием?
20. Каковы возможности при работе с удаленным репозиторием? Как его клонировать, получать и отправлять данные?
21. Что будет если в процессе работы изменить название папки с рабочим проектом, у нас это «песочница».

ИСТОЧНИКИ

[Git bash: определение, команды и начало работы | Atlassian](#)

[Git. Краткое руководство по терминалу | guides \(netology-code.github.io\)](#) !!!! о работе с терминалом.

Установка GitHub Desktop

<https://docs.github.com/ru/desktop/installing-and-configuring-github-desktop/installing-and-authenticating-to-github-desktop/installing-github-desktop>

Проверка подлинности в GitHub

<https://docs.github.com/ru/desktop/installing-and-configuring-github-desktop/installing-and-authenticating-to-github-desktop/authenticating-to-github>

<https://coderlessons.com/articles/veb-razrabotka-articles/git-na-windows-dlia-novichkov>

Небольшое руководство по Git и развертыванию

Прежде чем я отпущу вас, вот список фантастических ресурсов для продолжения изучения Git.

- [Git Essentials \(курс Tuts + Premium\)](#)
- [Github](#) – неограниченные бесплатные публичные репозитории
- [Bitbucket](#) – неограниченное количество бесплатных публичных и частных репозиторий
- [Beanstalk](#) – Частный Git с отличными развертываниями FTP
- [DeployHQ](#) – Развертывание любого [репозитория](#) Git через FTP
- [Простое управление версиями с помощью Git](#)
- [Terminal, Git и GitHub для остальных](#)