Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

лабораторная работа

на тему

«**Основы системы контроля версий Git**»

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил: |  | Козяков Андрей Игоревич, магистрант группы №815042 |  |
| Проверил: |  | Алексеев Виктор Федорович,  канд.техн.наук, доцент |  |

Минск 2018

**Тема: «Основы системы контроля версий Git»**

**Цель работы:** познакомиться с системой контроля версий Git, изучить возможности ее использования. После изучения теоретического материала выполнить тестовые задания для усвоения материала.

**Необходимые знания:** базовое знание команд консоли.

**1 Общие теоретические сведения**

Рассмотрим базовую теорию для того чтобы Вы могли выполнить эту лабораторную работу, а именно узнаем, что такое системы контроля версий, зачем они нужны, и почему так важны.

**1.1 Что такое система контроля версий**

Что такое "система контроля версий", и почему это важно? Система контроля версий – это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии. Для контроля версий файлов в этой книге в качестве примера будет использоваться исходный код программного обеспечения, хотя на самом деле вы можете использовать контроль версий практически для любых типов файлов.

Если вы графический или web-дизайнер и хотите сохранить каждую версию изображения или макета (скорее всего, захотите), система контроля версий (далее СКВ) – как раз то, что нужно. Она позволяет вернуть файлы к состоянию, в котором они были до изменений, вернуть проект к исходному состоянию, увидеть изменения, увидеть, кто последний менял что-то и вызвал проблему, кто поставил задачу и когда, и многое другое. Использование СКВ также значит в целом, что, если вы сломали что-то или потеряли файлы, вы спокойно можете всё исправить. В дополнение ко всему вы получите всё это без каких-либо дополнительных усилий.

**1.2 Что такое локальная система контроля версий**

Многие люди в качестве метода контроля версий применяют копирование файлов в отдельную директорию (возможно даже, директорию с отметкой по времени, если они достаточно сообразительны). Данный подход очень распространён из-за его простоты, однако он невероятно сильно подвержен появлению ошибок. Можно легко забыть, в какой директории вы находитесь, и случайно изменить не тот файл или скопировать не те файлы, которые вы хотели.

Для того, чтобы решить эту проблему, программисты давным-давно разработали локальные СКВ с простой базой данных, которая хранит записи о всех изменениях в файлах, осуществляя тем самым контроль ревизий. Одной из популярных СКВ была система RCS, которая и сегодня распространяется со многими компьютерами.



Рисунок 1.1 – Локальный контроль версий.

Даже популярная операционная система Mac OS X предоставляет команду rcs, после установки Developer Tools. RCS хранит на диске наборы патчей (различий между файлами) в специальном формате, применяя которые она может воссоздавать состояние каждого файла в заданный момент времени.

**1.3 Централизованные системы контроля версий**

Следующая серьёзная проблема, с которой сталкиваются люди, — это необходимость взаимодействовать с другими разработчиками. Для того, чтобы разобраться с ней, были разработаны централизованные системы контроля версий (ЦСКВ). Такие системы, как: CVS, Subversion и Perforce, имеют единственный сервер, содержащий все версии файлов, и некоторое количество клиентов, которые получают файлы из этого централизованного хранилища. Применение ЦСКВ являлось стандартом на протяжении многих лет.

Такой подход имеет множество преимуществ, особенно перед локальными СКВ. Например, все разработчики проекта в определённой степени знают, чем занимается каждый из них. Администраторы имеют полный контроль над тем, кто и что может делать, и гораздо проще администрировать ЦСКВ, чем оперировать локальными базами данных на каждом клиенте.

Несмотря на это, данный подход тоже имеет серьёзные минусы. Самый очевидный минус — это единая точка отказа, представленная централизованным сервером. Если этот сервер выйдет из строя на час, то в течение этого времени никто не сможет использовать контроль версий для сохранения изменений, над которыми он работает, а также никто не сможет обмениваться этими изменениями с другими разработчиками. Если жёсткий диск, на котором хранится центральная БД, повреждён, а своевременные бэкапы отсутствуют, вы потеряете всё — всю историю проекта, не считая единичных снимков репозитория, которые сохранились на локальных машинах разработчиков. Локальные СКВ страдают от той же самой проблемы — когда вся история проекта хранится в одном месте, вы рискуете потерять всё.



Рисунок 1.2 – Централизованный контроль версий.

**1.3 Децентрализованные системы контроля версий**

Здесь в игру вступают децентрализованные системы контроля версий (ДСКВ). В ДСКВ (таких как Git, Mercurial, Bazaar или Darcs), клиенты не просто скачивают снимок всех файлов (состояние файлов на определённый момент времени): они полностью копируют репозиторий. В этом случае, если один из серверов, через который разработчики обменивались данными, умрёт, любой клиентский репозиторий может быть скопирован на другой сервер для продолжения работы. Каждая копия репозитория является полным бэкапом всех данных.



Рисунок 1.3 – Децентрализованный контроль версий.

Более того, многие ДСКВ могут одновременно взаимодействовать с несколькими удалёнными репозиториями, благодаря этому вы можете работать с различными группами людей, применяя различные подходы единовременно, в рамках одного проекта. Это позволяет применять сразу несколько подходов в разработке, например, иерархические модели, что совершенно невозможно в централизованных системах.

**2 Основы Git**

Что же такое Git, если говорить коротко? Очень важно понять эту часть материала, потому что если вы поймёте что такое Git и основы того, как он работает, тогда, возможно, вам будет гораздо проще его использовать. Пока вы изучаете Git, попробуйте забыть всё, что вы знаете о других СКВ, таких как Subversion и Perforce; это позволит вам избежать определённых проблем при использовании утилиты. Git хранит и использует информацию совсем иначе по сравнению с другими системами, даже несмотря на то, что интерфейс пользователя достаточно похож, и понимание этих различий поможет вам избежать путаницы во время использования.

**2.1 Снимки, а не различия**

Основное отличие Git’а от любой другой СКВ (Subversion и её собратья включительно), это подход Git’а к работе со своими данными. Концептуально, большинство других систем хранят информацию в виде списка изменений в файлах. Эти системы (CVS, Subversion, Perforce, Bazaar и т.д.) представляют информацию в виде набора файлов и изменений, сделанных в каждом файле, по времени.



Рисунок 2.1 – Хранение данных относительно первоначальной версии каждого из файлов.

Git не хранит и не обрабатывает данные таким способом. Вместо этого, подход Git’а к хранению данных больше похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы делаете коммит, то есть сохраняете состояние своего проекта в Git’е, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок. Для увеличения эффективности, если файлы не были изменены, Git не запоминает эти файлы вновь, а только создаёт ссылку на предыдущую версию идентичного файла, который уже сохранён. Git представляет свои данные как, скажем, **поток снимков.**



Рисунок 2.2 – Хранение данных как снимков проекта во времени.

Это очень важное отличие между Git и почти любой другой СКВ. Git переосмысливает практически все аспекты контроля версий, которые были скопированы из предыдущего поколения большинством других систем. Это делает Git больше похожим на миниатюрную файловую систему с удивительно мощными утилитами, надстроенными над ней, нежели просто на СКВ. Когда мы будем рассматривать управление ветками в Ветвление в Git, мы увидим, какие преимущества вносит такой подход к работе с данными в Git.

Когда вы производите какие-либо действия в Git, практически все из них только добавляют новые данные в базу Git. Очень сложно заставить систему удалить данные либо сделать что-то, что нельзя впоследствии отменить. Как и в любой другой СКВ, вы можете потерять или испортить свои изменения, пока они не закоммичены, но после того, как вы закоммитите снимок в Git, будет очень сложно что-либо потерять, особенно, если вы регулярно синхронизируете свою базу с другим репозиторием.

Это самая важная вещь, которую нужно запомнить о Git, если вы хотите, чтобы остаток процесса обучения прошёл гладко. Git имеет три основных состояния, в которых могут находиться ваши файлы: зафиксированном (committed), изменённом (modified) и подготовленном (staged). “Зафиксированный” значит, что файл уже сохранён в вашей локальной базе. К изменённым относятся файлы, которые поменялись, но ещё не были зафиксированы. Подготовленные файлы — это изменённые файлы, отмеченные для включения в следующий коммит.

Мы подошли к трём основным секциям проекта Git: Git-директория (Git directory), рабочая директория (working directory) и область подготовленных файлов (staging area).

Область подготовленных файлов — это файл, располагающийся в вашей Git-директории, в нём содержится информация о том, какие изменения попадут в следующий коммит. Эту область ещё называют “индекс”, однако называть её stage-область также общепринято.

Базовый подход в работе с Git выглядит так:

1. Вы изменяете файлы в вашей рабочей директории.
2. Вы добавляете файлы в индекс, добавляя тем самым их снимки в область подготовленных файлов.
3. Когда вы делаете коммит, используются файлы из индекса как есть, и этот снимок сохраняется в вашу Git директорию.

**3 Установка Git**

Для установки Git в Windows также имеется несколько способов. Официальная сборка доступна для скачивания на официальном сайте Git’а. Просто перейдите на страницу <http://git-scm.com/download/win>, и загрузка запустится автоматически. Обратите внимание, что это проект, называемый Git для Windows (другое название msysGit), который отделён от самого Git; для получения дополнительной информации о нём перейдите на <http://msysgit.github.io/>.

Другой простой способ установки Git — установить GitHub для Windows. Его установщик включает в себя утилиты командной строки и GUI Git’а. Он также корректно работает с Powershell, обеспечивает чёткое сохранение учётных данных и правильные настройки CRLF. Вы познакомитесь с этими вещами подробнее несколько позже, здесь же отметим, что они будут вам необходимы. Вы можете загрузить GitHub для Windows с сайта [http://windows.github.com](http://windows.github.com/).

**3.1 Первоначальная настройка Git**

Теперь, когда Git установлен в вашей системе, самое время настроить среду для работы с Git’ом под себя. Это нужно сделать только один раз — при обновлении версии Git’а настройки сохранятся. Но, при необходимости, вы можете поменять их в любой момент, выполнив те же команды снова.

В состав Git’а входит утилита git config, которая позволяет просматривать и настраивать параметры, контролирующие все аспекты работы Git’а, а также его внешний вид.

Первое, что вам следует сделать после установки Git’а, — указать ваше имя и адрес электронной почты. Это важно, потому что каждый коммит в Git’е содержит эту информацию, и она включена в коммиты, передаваемые вами, и не может быть далее изменена:

$ git config --global user.name "John Doe"

$ git config --global user.email johndoe@example.com

Опять же, если указана опция --global, то эти настройки достаточно сделать только один раз, поскольку в этом случае Git будет использовать эти данные для всего, что вы делаете в этой системе. Если для каких-то отдельных проектов вы хотите указать другое имя или электронную почту, можно выполнить эту же команду без параметра --global в каталоге с нужным проектом.

Многие GUI-инструменты предлагают сделать это при первом запуске.

Если вы хотите проверить используемую конфигурацию, можете использовать команду git config --list, чтобы показать все настройки, которые Git найдёт:

$ git config --list

user.name=John Doe

user.email=johndoe@example.com

color.status=auto

color.branch=auto

color.interactive=auto

color.diff=auto

...

**3.2 Создание Git-репозитория**

Если вы собираетесь начать использовать Git для существующего проекта, то вам необходимо перейти в директорию проекта и в командной строке ввести

$ git init

Эта команда создаёт в текущей директории новую поддиректорию с именем .git, содержащую все необходимые файлы репозитория — основу Git-репозитория. На этом этапе ваш проект ещё не находится под версионным контролем.

**3.3 Определение состояния файлов**

Основной инструмент, используемый для определения, какие файлы в каком состоянии находятся — это команда git status. Если вы выполните эту команду сразу после клонирования, вы увидите что-то вроде этого:

$ git status

On branch master

nothing to commit, working directory clean

Это означает, что у вас чистый рабочий каталог, другими словами – в нем нет отслеживаемых измененных 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. В [Ветвление в Git](https://git-scm.com/book/ru/v2/ch00/ch03-git-branching) будут рассмотрены ветки и ссылки более детально.

Предположим, вы добавили в свой проект новый файл, простой файл README. Eсли этого файла раньше не было, и вы выполните git status, вы увидите свой неотслеживаемый файл вот так:

$ echo 'My Project' > README

$ git status

On branch master

Untracked files:

(use "git add <file>..." to include in what will be committed)

README

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

Понять, что новый файл README неотслеживаемый можно по тому, что он находится в секции ``Untracked files`` в выводе команды ``status``. Статус ``Untracked files``, по сути, означает, что Git видит файл, отсутствующий в предыдущем снимке состояния (коммите); Git не станет добавлять его в ваши коммиты, пока вы его явно об этом не попросите. Это предохранит вас от случайного добавления в репозиторий сгенерированных бинарных файлов или каких-либо других, которые вы и не думали добавлять. Мы хотели добавить README, так давайте сделаем это.

**3.4 Отслеживание новых файлов**

Для того чтобы начать отслеживать (добавить под версионный контроль) новый файл, используется команда git add. Чтобы начать отслеживание файла README, вы можете выполнить следующее:

$ git add README

Если вы снова выполните команду status, то увидите, что файл README теперь отслеживаемый и индексированный:

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: README

Вы можете видеть, что файл проиндексирован по тому, что он находится в секции ``Changes to be committed``. Если вы выполните коммит в этот момент, то версия файла, существовавшая на момент выполнения вами команды git add, будет добавлена в историю снимков состояния. Как вы помните, когда вы ранее выполнили git init, затем вы выполнили git add (файлы) — это было сделано для того, чтобы добавить файлы в вашем каталоге под версионный контроль. Команда git add принимает параметром путь к файлу или каталогу, если это каталог, команда рекурсивно добавляет (индексирует) все файлы в данном каталоге.

**3.5 Индексация изменённых файлов**

Давайте модифицируем файл, уже находящийся под версионным контролем. Если вы измените отслеживаемый файл ``CONTRIBUTING.md`` и после этого снова выполните команду git status, то результат будет примерно следующим:

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: README

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: CONTRIBUTING.md

Файл ``CONTRIBUTING.md`` находится в секции ``Changes not staged for commit`` — это означает, что отслеживаемый файл был изменён в рабочем каталоге, но пока не проиндексирован. Чтобы проиндексировать его, необходимо выполнить команду git add. Это многофункциональная команда, она используется для добавления под версионный контроль новых файлов, для индексации изменений, а также для других целей, например для указания файлов с исправленным конфликтом слияния. Вам может быть понятнее, если вы будете думать об этом как ``добавить этот контент в следующий коммит``, а не как ``добавить этот файл в проект``. Выполним git add, чтобы проиндексировать ``CONTRIBUTING.md``, а затем снова выполним git status:

$ git add CONTRIBUTING.md

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: README

modified: CONTRIBUTING.md

Теперь оба файла проиндексированы и войдут в следующий коммит. В этот момент вы, предположим, вспомнили одно небольшое изменение, которое вы хотите сделать в ``CONTRIBUTING.md`` до коммита. Вы открываете файл, вносите и сохраняете необходимые изменения и вроде бы готовы к коммиту. Но давайте-ка ещё раз выполним git status:

$ vim CONTRIBUTING.md

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: README

modified: CONTRIBUTING.md

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: CONTRIBUTING.md

Теперь ``CONTRIBUTING.md`` отображается как проиндексированный и непроиндексированный одновременно. Как такое возможно? Такая ситуация наглядно демонстрирует, что Git индексирует файл в точности в том состоянии, в котором он находился, когда вы выполнили команду git add. Если вы выполните коммит сейчас, то файл ``CONTRIBUTING.md`` попадёт в коммит в том состоянии, в котором он находился, когда вы последний раз выполняли команду git add , а не в том, в котором он находится в вашем рабочем каталоге в момент выполнения git commit. Если вы изменили файл после выполнения git add, вам придётся снова выполнить git add, чтобы проиндексировать последнюю версию файла:

$ git add CONTRIBUTING.md

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: README

modified: CONTRIBUTING.md

**3.6 Коммит изменений**

Теперь, когда ваш индекс находится в таком состоянии, как вам и хотелось, вы можете зафиксировать свои изменения. Запомните, всё, что до сих пор не проиндексировано — любые файлы, созданные или изменённые вами, и для которых вы не выполнили git add после момента редактирования — не войдут в этот коммит. Они останутся изменёнными файлами на вашем диске. В нашем случае, когда вы в последний раз выполняли git status, вы видели что всё проиндексировано, и вот, вы готовы к коммиту.

Вы можете набрать свой комментарий к коммиту в командной строке вместе с командой commit указав его после параметра -m, как в следующем примере:

$ git commit -m "Story 182: Fix benchmarks for speed"

[master 463dc4f] Story 182: Fix benchmarks for speed

2 files changed, 2 insertions(+)

create mode 100644 README

Итак, вы создали свой первый коммит! Вы можете видеть, что коммит вывел вам немного информации о себе: на какую ветку вы выполнили коммит (master), какая контрольная сумма SHA-1 у этого коммита (463dc4f), сколько файлов было изменено, а также статистику по добавленным/удалённым строкам в этом коммите.

Запомните, что коммит сохраняет снимок состояния вашего индекса. Всё, что вы не проиндексировали, так и висит в рабочем каталоге как изменённое; вы можете сделать ещё один коммит, чтобы добавить эти изменения в репозиторий. Каждый раз, когда вы делаете коммит, вы сохраняете снимок состояния вашего проекта, который позже вы можете восстановить или с которым можно сравнить текущее состояние.

**3.7 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 вы в следующий раз не увидите его как “неотслеживаемый”.

Если вы просто удалите файл из своего рабочего каталога, он будет показан в секции ``Changes not staged for commit`` (измененные, но не проиндексированные) вывода команды git status:

$ rm PROJECTS.md

$ git status

On branch master

Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:

(use "git add/rm <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

deleted: PROJECTS.md

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

Затем, если вы выполните команду git rm, удаление файла попадёт в индекс:

$ git rm PROJECTS.md

rm 'PROJECTS.md'

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

deleted: PROJECTS.md

После следующего коммита файл исчезнет и больше не будет отслеживаться. Если вы изменили файл и уже проиндексировали его, вы должны использовать принудительное удаление с помощью параметра -f. Это сделано для повышения безопасности, чтобы предотвратить ошибочное удаление данных, которые ещё не были записаны в снимок состояния и которые нельзя восстановить из Git.

**3.8 Просмотр истории коммитов**

После того, как вы создали несколько коммитов или же склонировали 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.

Следующие несколько примеров используют очень простой проект “simplegit”. Чтобы склонировать проект, используйте команду:

git clone https://github.com/schacon/simplegit-progit

Если вы запустите команду git log в папке склонированного проекта, вы увидите следующий вывод:

$ git log

commit ca82a6dff817ec66f44342007202690a93763949

Author: Scott Chacon <schacon@gee-mail.com>

Date: Mon Mar 17 21:52:11 2008 -0700

changed the version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7

Author: Scott Chacon <schacon@gee-mail.com>

Date: Sat Mar 15 16:40:33 2008 -0700

removed unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6

Author: Scott Chacon <schacon@gee-mail.com>

Date: Sat Mar 15 10:31:28 2008 -0700

first commit

По умолчанию (без аргументов) git log перечисляет коммиты, сделанные в репозитории в обратном к хронологическому порядке – последние коммиты находятся вверху. Из примера можно увидеть, что данная команда перечисляет коммиты с их SHA-1 контрольными суммами, именем и электронной почтой автора, датой создания и сообщением коммита.

Команда git log имеет очень большое количество опций для поиска коммитов по разным критериям.

**3.9 Операции отмены**

В любой момент вам может потребоваться что-либо отменить. Здесь мы рассмотрим несколько основных способов отмены сделанных изменений. Будьте осторожны, не все операции отмены в свою очередь можно отменить! Это одна из редких областей Git’а, где неверными действиями можно необратимо удалить результаты своей работы.

Отмена может потребоваться, если вы сделали коммит слишком рано, например, забыв добавить какие-то файлы или комментарий к коммиту. Если вы хотите переделать коммит, можно запустить commit с параметром --amend (дополнить):

$ git commit --amend

Эта команда использует для дополнения коммита вашу область подготовки (индекс). Если вы ничего не меняли с момента последнего коммита (например, команда запущена сразу после предыдущего коммита), то снимок состояния останется в точности таким же, а изменится лишь комментарий к коммиту.

Запустится тот же редактор комментария к коммиту, но уже с комментарием к предыдущему коммиту. Комментарий можно отредактировать точно так же, как обычно, просто он заменит собой предыдущий.

Например, если вы фиксируете изменения, и понимаете, что забыли проиндексировать изменения в файле, который хотели включить в коммит, можно сделать примерно так:

$ git commit -m 'initial commit'

$ git add forgotten\_file

$ git commit --amend

В итоге получится единый коммит — второй коммит заменит результаты первого.

**3.10 Отмена подготовки файла**

В следующих двух разделах показано, как разбираться с изменениями вашей области подготовки (staging area) и рабочего каталога. Радует, что команда, которой вы определяете состояние этих областей, также напоминает вам, как отменять их изменения. Например, скажем, вы изменили два файла, и хотите закоммитить их двумя раздельными изменениями, но случайно набрали git add \*, и добавили оба в индекс. Как отменить добавление одного из них? Команда git status напомнит вам:

$ git add .

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

renamed: README.md -> README

modified: CONTRIBUTING.md

Прямо под текстом “Changes to be committed” говорится: git reset HEAD <file>... для отмены добавления в индекс. Давайте последуем этому совету, и отменим индексирование файла CONTRIBUTING.md:

$ git reset HEAD CONTRIBUTING.md

Unstaged changes after reset:

M CONTRIBUTING.md

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

renamed: README.md -> README

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: CONTRIBUTING.md

Команда выглядит несколько странно, но — работает! Файл CONTRIBUTING.md изменен, но снова не добавлен в область подготовки к коммиту.

**4 Практика**

**4.1 Контрольные вопросы**

* 1. Что такое система контроля версий?
  2. Какие различные системы контроля версий Вы знаете?
  3. Для чего нужен Git?
  4. В чем отличия Git от других систем контроля версий?
  5. Как начать работу с Git?
  6. Какие основные функции Git Вы знаете?

**4.2 Порядок выполнения работы**

* 1. Изучение теоретические сведения.
  2. Ознакомиться с полезными источниками из презентации.
  3. Ответить на контрольные вопросы.
  4. Выполнить просто задание.

**4.3 Задание**

* 1. Установить Git на рабочую машину;
  2. Проинициализировать тестовый Git-репозиторий;
  3. В настройках указать свое имя и фамилию;
  4. Добавить в репозиторий новый файл и закоммитить его с сообщением «Init commit»;
  5. Произвести изменения в файле и закоммитить из с сообщением «Change file 1»;
  6. Добавить новый файл в репозиторий и закоммитить его с произвольным сообщением;
  7. Удалить первый файл и закоммитить данные изменения с сообщением «Remove file [имя удаленного файла]»;
  8. Просмотреть историю сделанных коммитов;

**Памятка:** пользоваться ресурсами можно не только из раздела с теорией, а также интернетом ввиду популярности данной и темы, и большого источника с информацией.