Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного образовательного
учреждения высшего образования
**«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»**
(КФ МГТУ им. Н.Э. Баумана)

К. А. АМЕЛИЧЕВА Г. Э. АМЕЛИЧЕВ

«Использование системы контроля версий и хостинга проектов»

Методические указания к выполнению домашней работы

по дисциплине «Основы программной инженерии»

Калуга – 2019

УДК 004.4 424

ББК 32.973.3

Методические указания составлены в соответствии с учебным планом КФ МГТУ им. Н.Э. Баумана по направлению подготовки 09.03.04 «Программная инженерия» кафедры «Программного обеспечения ЭВМ, информационных технологий».

Методические указания рассмотрены и одобрены:

- Кафедрой «» (ИУ4-КФ) протокол № 51.4/05 от «18» декабря 2019 г.

Зав. кафедрой ИУ4-КФ _____ к.т.н., доцент Ю.Е. Гагарин

- Методической комиссией факультета ИУ-КФ протокол №7 от «23» декабря 2019 г.

Председатель методической комиссии факультета ИУ-КФ _____ к.т.н., доцент М.Ю. Адкин

- Методической комиссией КФ МГТУ им.Н.Э. Баумана протокол № 4 от «24» 12 2019 г.

Председатель методической комиссии КФ МГТУ им.Н.Э. Баумана _____ д.э.н., профессор О.Л. Перерва

Рецензент:
Зав кафедрой ИУ5-КФ, к.ф-м.н., доцент _____ Е.В. Вершинин

Авторы
к.т.н., доцент кафедры ИУ4-КФ _____ К.А. Амеличева

ассистент кафедры ИУ4-КФ _____ Г.Э. Амеличев

Аннотация

Методические указания к выполнению домашней работы по дисциплине «Основы программной инженерии» ориентированы на студентов 1-го курса направления подготовки 09.03.04 «Программная инженерия» содержат рекомендации методические рекомендации по работе с распределенной системой контроля версий, индивидуальные варианты для самостоятельной работы, задания и вопросы для формирования и контроля владения компетенциями.

УДК 004.4 424

ББК 32.973.3

Методические указания к выполнению домашней работы по курсу «Основы программной инженерии». — М.: Издательство МГТУ им. Н.Э. Баумана, 2019. — 37 с.

Методические указания к выполнению домашней работы, в рамках самостоятельной работы студентов, по дисциплине «Основы программной инженерии» содержат описание выполнения домашнего задания: теоретическую часть, практическую часть, индивидуальные варианты и методические рекомендации к их выполнению.

Предназначены для студентов 1-го курса бакалавриата КФ МГТУ им. Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Дисциплина «основы программной инженерии» рассматривает методы и инструментальные средства программной инженерии для решения задач создания качественного программного обеспечения с применением современных информационных и компьютерных технологий.

Данное домашнее задание демонстрирует работу членов команды программного проекта над общей базой исходных кодов с использованием распределенной системы контроля версий Git и сервиса хостинга проектов BitBucket.

Цель настоящих методических указаний - облегчить самостоятельную работу студентов, на практике изучающих понятия и компоненты систем контроля версий (СКВ), порядок и приемы работы с ними. Даны методические рекомендации по первоначальной настройке системы контроля версии Git, создания и клонирования репозитория и выполнения основных действий с исходным кодом программы, связанные с контролем версий.

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ

Цель: формирование практических навыков применения систем контроля версий и сервисов хостинга проектов для организации коллективной работы.

Основными задачами выполнения домашней работы являются изучение основных приемов работы с распределенной системы контроля версий Git и сервиса хостинга проектов BitBucket, демонстрация работы членов команды программного проекта над общей базой исходных кодов.

Основными задачами выполнения домашнего задания являются:

1. Ознакомиться с понятием, назначением, порядком и общими правилами работы с системой контроля версий;
2. продемонстрировать командную работу при разработке программного продукта;
3. написать программный код для индивидуального варианта.

Результатами работы являются:

- Созданный открытий репозиторий, с размещенной в нем работоспособной программой, фиксацией вносимых изменений и синхронизацией их с репозиторием.
- Подготовленный отчет

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Система контроля версий

Контроль версий подразумевает под собой комплекс методов, направленных на систематизацию изменений, вносимых разработчиками в программный продукт в процессе его разработки и сопровождения, сохранение целостности системы после изменений, предотвращение нежелательных и непредсказуемых эффектов. Также использование систем контроля версий позволяет сделать процесс внесения изменений более формальным. Система управления версиями гарантирует, что каждый автор всегда работает с самой последней версией файла, а также исключает возможность случайной перезаписи любым из авторов работы своих коллег.

BitBucket – это сервис для разработчиков программного обеспечения, предоставляющий хостинг для проектов. Одним из основных сервисов, предоставляемых BitBucket, является контроль версий. Доступные системы контроля версий – Git и Mercurial. Также доступны сервисы просмотра исходного кода, issue-трекинга, ведения wiki проекта и другие.

Регистрация в сервисе хостинга проектов BitBucket

Страница сервиса: https://bitbucket.org//

На главной странице необходимо указать имя пользователя, адрес email и пароль. Нажать кнопку *«Sign up for free»*.

На странице *«Now tell us a little about yourself»* стоит указать дополнительную информацию (имя, фамилию) и нажать кнопку *«Submit»*. Данный шаг можно пропустить, нажав ссылку *«Skip»*.

На странице *«Create your first repository»* следует нажать *«No thanks»*.

Регистрация завершается, отображается страница «*Get started*».

На указанный при регистрации адрес *email* будет отправлено письмо с темой *«[Bitbucket] Confirm your email address».* Для подтверждения указанного адреса необходимо перейти по ссылке, указанной в письме (фактически, нажать на кнопку *«Confirm this email address»).*

Если письмо не было получено, то следует проверить правильность указания адреса и запросить повторную отправку письма-подтверждения. Для этого в меню пользователя (отображается при клике по аватарке в правом верхнем углу) нужно выбрать пункт *«Manage account»,* а затем пункт *«Email addresses»* в вертикальном меню слева.

Необходимо проверить правильность почтового адреса в таблице *«Email address*». Если адрес верный, то необходимо нажать на ссылку *«resend confirmation»* и ожидать письмо.

Если в адресе ошибка, но необходимо добавить правильный адрес, подтвердить его и удалить неверный адрес.

Подготовка рабочего места (ОС Windows)

Действия, описанные в данном разделе, позволяют настроить рабочее место, например, дома. Их не нужно выполнять в случае работы в лабораторных аудиториях.

Рекомендуется работу с Git осуществлять в среде *Atlassian SourceTree.* Перед ее установкой следует установить пакет *msysGit (Git для Windows),* утилиту сравнения и слияния *файлов P4Merge.* Ниже описана установка *SourceTree.* Следует учесть, что данное приложение работает только в ОС Windows 7 и выше.

В качестве альтернативы *SourceTree* можно использовать приложение *TortoiseGit.* Перед его установкой так же необходимо установить *msysGit.* Установка *TortoiseGit* описана ниже. Далее в данных указаниях описана работа с *SourceTree,* в случае использования *TortoiseGit* с программой надо будет разбираться самостоятельно.

Установка msysGit

а) Со страницы проекта *msysGit (http://msysgit.github.io/)* загрузить пакет установки, нажав кнопку *«Download».*

б) Запустить пакет установки. На шаге *«Adjusting your PATH environment»* выбрать *«Use Git from the Windows Command Prompt».*

Установка P4Merge

Данное приложение может свободно использоваться не более чем 20 пользователями на не более чем 20 рабочих местах.

а) *На странице* http://www.perforce.com/downloads/Perforce/20-User#10

в разделе *«Perforce Clients»* нажать «–» напротив *«P4V: Visual Client»* и выбрать корректную версию операционной системы напротив *«P4Merge: Visual Merge Tool»,* затем нажать кнопку *«Accept and Continue».* В появившемся диалоге *«Create. Develop. Deploy»* можно нажать кнопку *«Skip Registration»,* после чего начнется загрузка пакета установки.

б) Запустить пакет установки. На этапе *«Select Features»* убедиться, что компонент *«Visual Merge Tool (P4Merge)»* отмечен как устанавливаемый. Остальные компоненты можно отключить.

Установка и первоначальная настройка SourceTree

Следует обратить внимание, что данное приложение может использоваться только в ОС Windows 7 и выше (есть еще дистрибутив под Mac OS X).

а) Со страницы продукта *(http://www.sourcetreeapp.com/)* загрузить пакет установки, нажав кнопку *«Download SourceTree Free».*

б) Запустить пакет установки.

в) После завершения установки запустить приложение. В окне приветствия установить флаг *«I agree to the license agreement».*

г) В диалоге *«Install global ignore file?»* нажать кнопку *«Yes».*

д) На странице *«Add an account…»* ввести учетные данные (имя пользователя и пароль) от сервиса *BitBucket.*

е) На странице *«Clone your first repo»* нажать *«Skip setup».*

ж) После завершения начальной инициализации в главном окне программы выбрать пункт меню *«Tools → Options».* В открывшемся окне *«Options» на вкладке «General»* убедиться, что стоит галочка *«Allow SourceTree to modify your global Git and Mercurial config files».* В поле *«Full Name»* необходимо ввести свое имя и фамилию (латиницей), в поле *«Email Address»* – почту. Перейти на вкладку *«Diff».* В группе элементов *«External Diff/Merge»* в выпадающих списках *«External Diff Tool» и «Merge Tool»* выбрать *«P4Merge».* Нажать кнопку *«OK».*

Установка TortoiseGit

а) Со страницы проекта

TortoiseGit (http://code.google.com/p/tortoisegit/) загрузить пакет установки. Полный адрес версии 1.8.12*:*
для 32-битной ОС:
http://download.tortoisegit.org/tgit/1.8.12.0/TortoiseGit-1.8.12.0-32bit.msi
для 64-битной ОС:
http://download.tortoisegit.org/tgit/1.8.12.0/TortoiseGit-1.8.12.0-64bit.msi

б) Запустить пакет установки.

Создание команды в сервисе BitBucket

а) При необходимости выполнить вход в сервис *BitBucket.*

б) В горизонтальной панели в верхней части экрана в меню *«Teams»* выбрать пункт *«Create team».*

в) Открывается страница *«Create a team» (рис. 1).* На странице создания команды нужно указать название команды (описательный текст) и идентификатор команды *(указан в задании –3 п.).*

г) По умолчанию в команду добавляется текущий пользователь с правами администратора. Нужно добавить остальных членов

команды. Для этого в текстовоеполе можно начать вводить ФИО, имя пользователя или *email*, затем выбрать нужного пользователя, нажать кнопку *«Add»* и поставить галочку *«Administrator».*

д) Для создания команды с указанными реквизитами необходимо нажать кнопку *«Create».*

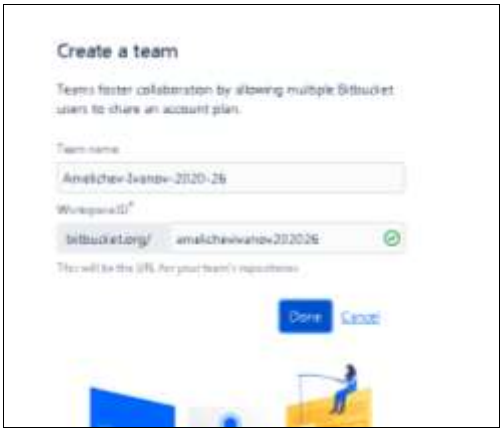

Рис. 1. Страница создания команды

После создания команда будет отображаться у всех членов в меню «Teams» горизонтальной панели в верхней части экрана. Если выбрать команду в меню, то отобразится рабочая панель команды, содержащая командные репозитории, а также средства настройки команды (если текущий пользователь является ее администратором).

Создание репозитория в сервисе BitBucket

а) При необходимости выполнить вход в сервис *BitBucket*.

б) В горизонтальной панели в верхней части экрана нажать кнопку *«Create»* или кнопку *«Create a repository»* на странице *«Get started».*

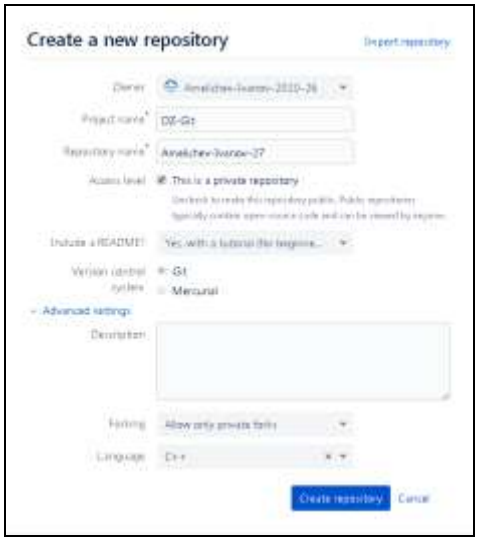

Рис. 2. Страница создания репозитория

в) На форме *«Create a new repository»* ввести название (указано в задании – см. п. 4) и описание репозитария, снять галочку *«This is a private repository»,* выбрать тип репозитария *«Git»,* выбрать основной язык разработки *«C++»,* нажать кнопку *«Create repository»* (см. рис. 2).

Клонирование репозитория

Рис. 3. Переход к репозиторию с рабочей панели

Рис. 4. Инициация клонирования репозитория

а) При необходимости выполнить вход в сервис BitBucket.

б) На рабочей панели пользователя (отображается при клике на Bitbucket слева вверху) или рабочей панели команды (отображается при выборе команды в меню «*Teams*» горизонтальной панели в верхней части экрана) выбрать репозиторий (рис. 3).

в) На странице репозитория (рис. 4) в левом меню в разделе «*Actions*» выбрать пункт «*Clone*» (маркер «1»), затем в появившемся всплывающем окне нажать кнопку «*Clone in SourceTree*». Пока репозиторий пуст, выполнить клонирование можно и быстрее – нажав кнопку «*Clone in SourceTree*»

г) Запустится приложение *SourceTree*, откроется диалог «*URL Actions*» (рис. 5).

В поле «*Destination Path*» нужно указать путь к папке, в которой будет создан репозиторий, убедиться, что стоит галочка «*Bookmark this repository*», в поле «*Name*» указать имя, под которым репозиторий будет отображаться в главном окне *SourceTree*. После указания всех реквизитов затем нажать кнопку «*Clone*».

Рис. 5. Подготовка к клонированию в приложении SourceTree

В ходе клонирования отображается окно «*Cloning from … to …*», после завершения оно закрывается, а в главном окне программы

SourceTree в панели слева появится закладка на клонированный репозиторий (рис. 6). При двойном клике по закладке отобразится вкладка с репозиторием.

Рис. 6. Главное окно приложения SourceTree с закладкой на репозиторий

Основы работы с репозиторием и фиксации изменений

В папке, в которую клонировался репозиторий, есть скрытая папка «.git» – это каталог Git, в котором хранятся метаданные и база данных объектов проекта. Фактически, при клонировании репозитория происходит копирование именно каталога Git.

Остальное содержимое папки – это рабочий каталог – извлеченная из базы данных Git некоторая версия файлов и папок проекта. Обычно работа с проектом осуществляется именно в этой папке: здесь создаются файлы проекта, файлы исходных кодов, здесь происходит добавление новых файлов, изменение существующих, удаление ненужных.

Как только в ходе работы достигнута некоторая контрольная точка: решена некоторая задача или реализован некоторый законченный шаг ее решения, – внесенные в проект изменения фиксируются (*commit*). Набор таких коммитов формирует историю работы над проектом.

Перед фиксацией (коммитом) необходимо отметить изменения, которые будут зафиксированы. Соответствующие записи вносятся в область подготовленных файлов (*staging area*) или индекс (*index*).

Обычный порядок работы с Git выглядит так:

1. осуществляется изменение/добавление/удаление файлов в рабочем каталоге;

2. файлы подготавливаются к фиксации путем добавления их текущего содержимого в область подготовленных файлов (индекс);

3. изменения фиксируются, при этом подготовленные файлы извлекаются из индекса и помещаются в каталог Git.

Следует обратить внимание, что Git «запоминает» именно то состояние файла, которое было на момент его добавление в индекс. Так, если файл изменить (состояние «A»), добавить в индекс и снова изменить (состояние «B»), зафиксировано будет состояние «A», которое было сохранено в индексе.

При работе с системами контроля версий необходимо следовать ряду правил:

- «одна задача – один коммит». В одном коммите нельзя совмещать изменения, относящиеся к разным задачам;
- «частые коммиты». Любое осмысленное состояние проекта должно быть зафиксировано в системе контроля версий;
- у каждого коммита должен быть комментарий. Вообще, Git не допускает коммита без комментария, но некоторые клиенты могут сами сгенерировать комментарий-заглушку, если пользователь ничего не указал;
- комментарий к коммиту должен описывать смысл внесенных изменений (например, «реализована такая-то возможность», «исправлена такая-то ошибка»), а не называть/перечислять сами изменения (типа «файл такой-то удален»);
- перед каждым коммитом необходимо внимательно просмотреть все фиксируемые изменения: очень часто фиксируется что-то

лишнее, или забывается что-то важное, или в один коммит попадают разные изменения и т. п.;

- в систему контроля версий не добавляют генерируемые в ходе сборки проекта файлы (например, исполняемые файлы *exe* и *dll*, объектные файлы *obj* и проч.).

Работа с репозиторием, фиксация изменений

Пусть, работая с рабочей копией (из файлового менеджера операционной системы, среды разработки и проч.), мы изменили файлы FOO.txt, BAR.txt, удалили файл BAZ.txt и добавили новый файл QUX.txt. И пусть изменение файла BAR.txt было внесено «по ходу», т.е. оно не относится к текущей решаемой задаче, а значит оно должно быть сохранено в рамках отдельного коммита.

В окне приложения *SourceTree* на закладке репозитория отображается состояние рабочей копии (см. рис. 7, маркер «2», слева направо: 2 файла изменены, 1 файл удален и 1 файл неизвестен системе контроля версий).

После завершения текущей работы для фиксации результатов следует нажать кнопку «*Commit*» в панели инструментов *SourceTree* (*рис. 7, маркер «1»).* На вкладке репозитория отобразится окно фиксации изменений (рис. 8).

В области «*Working Copy Changes* содержание индекса, т. е. перечень изменений, выбранных для фиксации.

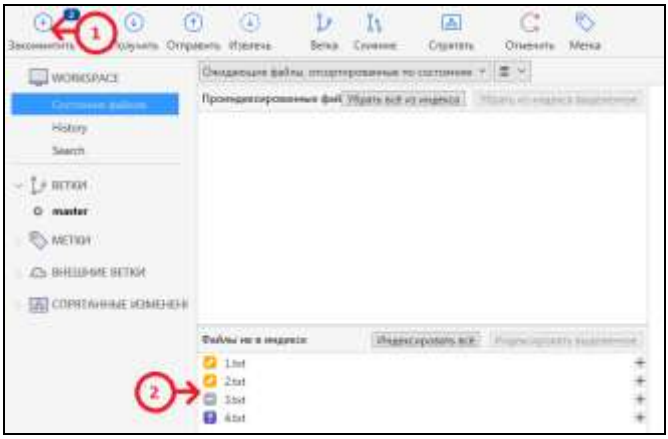

Рис. 7. Перед фиксацией изменений в рабочей копии

Рис. 8. Окно фиксации изменений

При выделении файла в области «*Working Copy Changes*» или «*Staged Changes*» в области справа (рис. 8, маркер «3») отображаются изменения, внесенные в файл: красным цветом отображаются удаленные строки, зеленым – добавленные. Для более информативного просмотра изменений можно использовать внешнюю (с точки зрения *SourceTree*) утилиту, для этого можно в контекстном меню файла выбрать пункт «*External Diff*», после чего запустится приложение *P4Merge* (в случае следования шагам см.

установка *P4Merge* и установка и первоначальная настройка *SourceTree*; см. рис. 9).

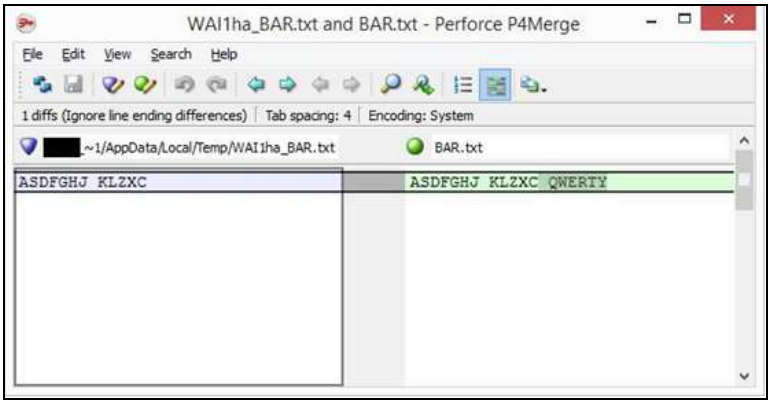

Рис. 9. Утилита P4Merge в режиме сравнения версий файла

Для того, чтобы добавить файл в индекс, нужно в области «Working Copy Changes» окна фиксации изменений выделить файл левым кликом мыши и в контекстном меню выбрать пункт «Add to index».

Если файл был изменен по ошибке, и необходимо вернуть его к предыдущему зафиксированному состоянию, то в контекстном меню необходимо выбрать пункт «*Discard*» и подтвердить запрос на откат изменений. Все внесенные изменения будут потеряны.

Если файл был по ошибке добавлен в индекс (т. е. его изменения не должны сейчас попасть в коммит), то в контекстном меню следует выбрать пункт «*Unstage from index*». Все изменения по-прежнему сохраняются, но не будут зафиксированы, файл будет исключен из индекса и отобразится в области «*Working Copy Changes*».

В данном примере файлы FOO.txt, BAZ.txt и QUX.txt будут добавлены в индекс, а BAR.txt будет оставлен в «*Working Copy Changes*». При этом файл QUX.txt будет помечен как добавленный (*Added*), а BAZ.txt – как удаленный (*Removed*) (см. рис. 10).

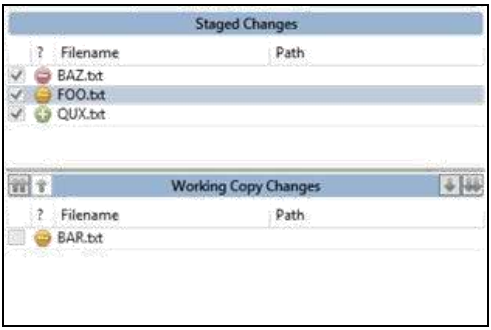

Рис. 10. Подготовленные к коммиту файлы

Перед фиксацией в текстовое поле «*Commit message*» (рис. 8, маркер «4») необходимо ввести комментарий к коммиту. Если данный коммит является частью серии коммитов в рамках одной задачи, то в выпадающем списке «*Recent messages*» (рис. 8, маркер «5») можно выбрать один из комментариев к предыдущим коммитам и отредактировать его.

После того, как файлы подготовлены к фиксации (помещены в индекс) и введен комментарий к коммиту, нужно нажать кнопку «Commit» в правом нижнем углу окна фиксации изменений (рис. 8).

В рассматриваемом примере, сразу после этого можно было бы зафиксировать и изменения файла BAR.txt, указав при этом соответствующий комментарий.

Игнорирование файлов

В ходе работы над проектом некоторые файлы никогда не будут добавляться в систему контроля версий, а значит их нет смысла и отображать в списке изменений рабочего каталога. В корень рабочего каталога можно поместить текстовый файл «.gitignore», в котором описать правила игнорирования файлов.

Пусть мы разрабатываем консольное приложение *Foo* в среде *Free Pascal Lazarus*.

В ходе работы над проектом и его компиляции проекта эта IDE генерирует следующую файловую структуру:

- файл «Foo.lpi» – файл проекта;
- файл «Foo.lps» – файл с информацией о сеансе работы с IDE;
- файл «Foo.lpr» – файл с исходным кодом основного модуля проекта;
- файлы с расширением «pas» – файлы с исходными кодами прочих модулей проекта;
- файл «Foo.exe» – исполняемый файл программы (результат компиляции проекта);
- папка «backup» – резервные копии файлов с исходным кодом (автосохранение);
- папка «lib» – промежуточные файлы компиляции.

В данном примере в систему контроля версий следует добавлять только файлы с исходным кодом и файл проекта.

Файл с информацией о сеансе работы с IDE индивидуален для каждого разработчика, нет смысла вести его историю и разделять с другими. Папку «*backup*» нет смысла хранить в системе контроля версий, т. к. сама система контроля версий хранит историю всех значимых изменений.

Результирующий исполняемый файл и промежуточные файлы из папки «*lib*» не имеет смысла хранить в системе контроля версий, т. к. их и так можно получить путем компиляции любой сохраненной корректной версии. Кроме того, хранение многих версий двоичных файлов обычно занимает гораздо больше дискового пространства, чем хранение версий текстовых файлов.

Итак, всю папку «*backup*», всю папку «*lib*», файлы «*exe*» и «*lps*» необходимо добавить в список игнорирования.

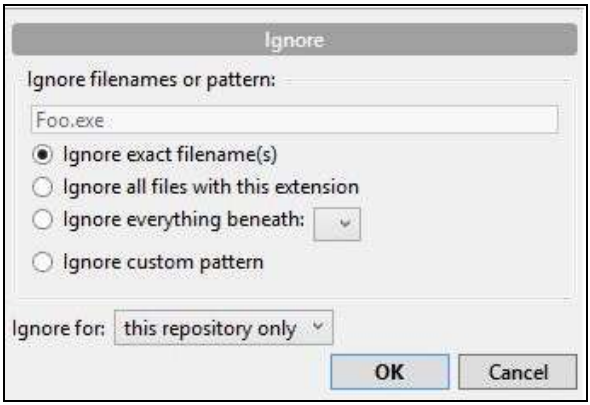

Рис. 11. Добавление записи в список игнорирования

Один из вариантов выполнения этого действия – при первом коммите. В области «*Working Copy Changes*» необходимо выделить игнорируемый файл (например, «*Foo.exe*») и в контекстном меню выбрать пункт «*Ignore…*». Отобразится окно добавления записи в список игнорирования (рис. 11).

Опция «*Ignore exact filename(s)*» позволяет игнорировать именно те файлы, которые были выделены (их имена отображаются в текстовом поле).

Опция «*Ignore all files with this extension*» позволяет игнорировать все файлы того же типа, что и выделенный (т. е. с таким же расширением).

Опция «*Ignore everything beneath*» позволяет игнорировать все файлы из выбранной папки.

Опция «*Ignore custom pattern*» позволяет вручную задать шаблон игнорирования.

Выпадающий список «*Ignore for*» позволяет указать, применить ли указанные настройки игнорирования только к текущему репозиторию (опция «*this repository only*») или ко всем репозиториям на рабочем месте (опция «*all repositories*»).

Рекомендуется настраивать игнорирование только для текущего репозитория. Настройка для всех локальных репозиториев требуется крайне редко, и пользоваться ей нужно очень осторожно.

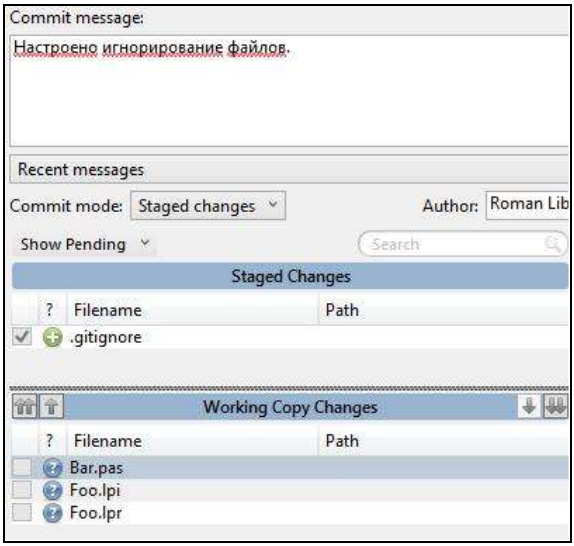

Рис. 12. Фиксация файла с настройками игнорирования.

В коммит включен только файл с настройками, остальные изменения будут зафиксированы отдельным коммитом.

После выбора варианта игнорирования нужно нажать кнопку «*OK*».

В рассматриваемом примере можно выбрать следующие опции:

- для «Foo.exe» и « Foo.lps» – «*Ignore all files with this extension*»;
- для папки «lib» – выделить любой файл, в окне игнорирования выбрать опцию «Ignore everything beneath» и выбрать папку «lib».

После добавления настроек игнорирования будет сгенерирован файл «.gitignore», который необходимо включить в коммит, желательно самостоятельный (рис. 12).

При необходимости файл с настройками игнорирования можно отредактировать вручную из рабочего каталога репозитория, а можно в приложении *SourceTree* в меню «*Repository*» выбрать пункт «*Repository Settings…*» (при этом в основной рабочей области должна быть открыта вкладка нужного репозитория). В открывшемся окне «*Repository Settings*» перейти на вкладку «*Advanced*», в ней в разделе «*Repository-specific ignore list*» нажать кнопку «*Edit*».

Синхронизация с удаленным репозиторием

В терминах *Git* удаленные репозитарии называются *remote*, за их счет обеспечиваются связи между репозиториями.

В данной работе удаленный репозиторий – это репозиторий BitBucket, который используется для обеспечения совместной работы над общим проектом.

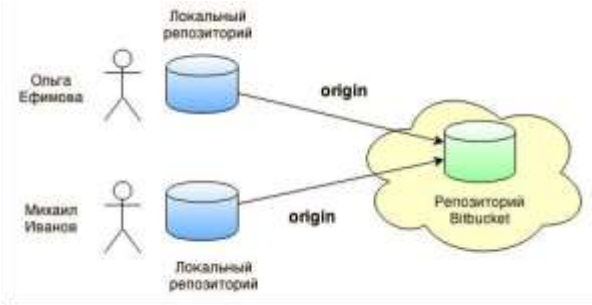

Рис. 13. Связь локального репозитория с репозиторием, с которого осуществлялось клонирование

При клонировании в результирующий репозиторий Git всегда добавляет ссылку на исходный репозиторий и называет эту ссылку «*origin*» (см. рис. 13).

В приложении *SourceTree* ссылки на все удаленные репозитории отображаются в разделе «*Remotes*» на вкладке репозитория (рис. 14).

Рис. 14. Отображение ссылок на удаленные репозитории

Для синхронизации с удаленным репозиторием служат следующие операции:

- *Fetch* – получает все изменения из удаленного репозитория, но никак не объединяет их с локальными изменениями и не меняет рабочую копию;
- *Pull* – получает все изменения из удаленного репозитория, объединяет (осуществляет слияние – merge) их с локальными изменениями и в рабочую копию загружает результат слияния;
- *Push* – отправляет изменения из локального репозитория в удаленный.

Получение изменений из удаленного репозитория (pull)

Для получения изменений в приложении *SourceTree* необходимо при открытой вкладке репозитория нажать кнопку «*Pull*» в панели инструментов. Отобразится окно «*Pull*» (рис. 15).

В выпадающем списке «*Pull from remote*» должен быть выбран удаленный репозиторий, из которого загружаются изменения. В нашем случае это «*origin*».

В выпадающем списке «*Remote branch to pull*» должно быть выбрано «*master*».

Если этот список пуст, а в удаленном репозитории сохранены изменения, то нужно нажать кнопку «*Refresh*», после чего содержимое списка обновится.

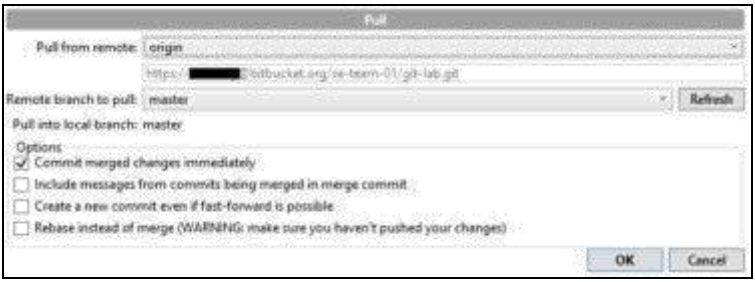

Рис. 15. Окно получения удаленных изменений (Pull)

Чтобы начать загрузку необходимо нажать кнопку «*OK*». После завершения загрузки на вкладке репозитория в истории изменений отобразятся загруженные изменения.

Если при этом произошло слияние (*merge*), то Git автоматически создаст соответствующий коммит с комментарием типа: «Merge branch 'master' of https://bitbucket.org/se-team-01/git-lab».

В некоторых случаях, например, если два разработчика одновременно меняли один файл, Git не может осуществить слияние самостоятельно. Такая ситуация называется конфликтом (*conflict*), разработчик должен вручную разрешить конфликт и завершить слияние.

Пусть над проектом работает 2 разработчика: Михаил Иванов и Ольга Ефимова. И пусть Михаил сделал коммит 0dae5fc и отправил его в BitBucket, а Ольга получила это изменение.

С этого момента Михаил и Ольга начинают работать параллельно. При этом они изменяют общий файл – «main.m» и фиксируют свои изменения: у Ольги коммит 4382977, а у Михаила – коммит 5f8ba0e, и оба этих коммита основаны на первом коммите Михаила – 0dae5fc. Ольга успевает отправить свои изменения в BitBucket. А когда Михаил получает изменения из BitBucket, возникает конфликтная ситуация для файла «main.m». Git вносит

изменения в рабочий каталог, но завершить слияние не может, поэтому изменения остаются незафиксированы.

Приложение *SourceTree* на закладке репозитория отображает метку незавершенного слияния и наличия конфликтных файлов, в истории отображает незафиксированные изменения (рис. 16, маркер «1»), а в области индекса выделяет конфликтные файлы, показывая иконку с восклицательным знаком (рис. 16, маркер «2»).

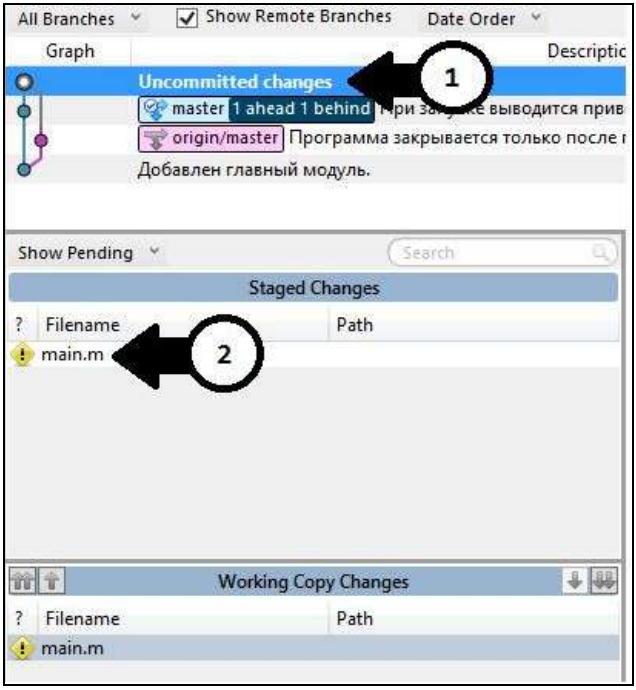

Рис. 16. Незавершенное слияние с конфликтными файлами

Для разрешения конфликта необходимо выделить файл и в контекстном меню выбрать пункты «Resolve Conflicts → Launch External Merge Tool». Будет запущена утилита *P4Merge*, которая отобразит исходное состояние файла (BASE), и конфликтующие изменения (LOCAL и REMOTE) (см. рис. 17).

В верхней части окна (область «A» на рис. 17) схематически показан граф изменений. В средней части окна (область «B» на

рис. 17) в трех столбцах отображаются: исходная версия файла (в середине – *BASE*), изменения, внесенные в локальный репозиторий (слева – *LOCAL*), и изменения, полученные из удаленного репозитория (справа – *REMOTE*). В нижней части окна (область «C» на рис. 17) показан результат слияния.

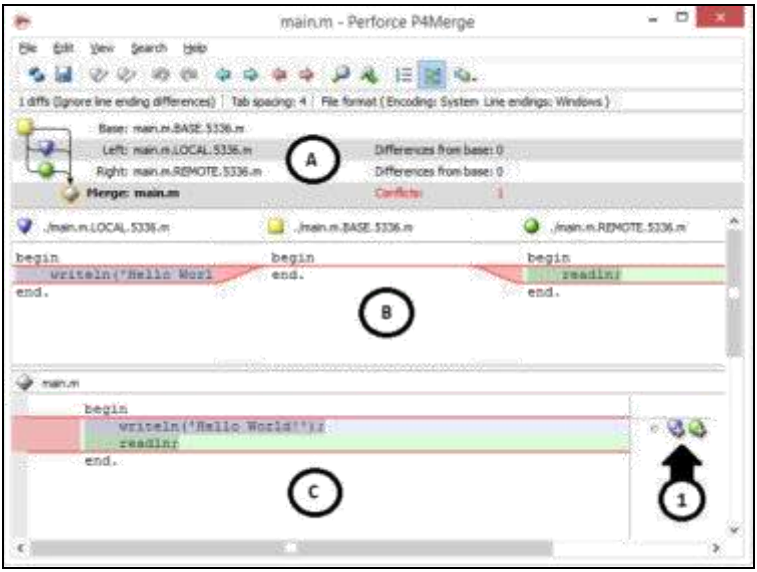

Рис. 17. Разрешение конфликтов с помощью утилиты P4Merge

При помощи иконок-кнопок (маркер «1» на рис. 17) можно выбирать, какие из изменений попадут в результат. Кроме того, в область результата слияния (область «C») можно вносить изменения вручную.

После разрешения конфликта нужно сохранить внесенные изменения, затем закрыть окно утилиты *P4Merge.* Приложение *SourceTree* после завершения работы утилиты помечает конфликт как разрешенный *(resolved),* внесенные изменения добавляет в индекс, а конфликтное состояние файла сохраняет в файле *«main.m.orig».* Если разрешение конфликтов выполнено правильно, то этот файл можно удалить.

После разрешения всех конфликтов необходимо зафиксировать результат слияния: нажать *кнопку «Commit»* в панели инструментов *SourceTree*. Автоматически будет сгенерирован комментарий о слиянии, в него будут включены имена файлов, в которых были конфликты.

Отправка изменений в удаленный репозиторий (push)

Для отправки изменений в приложении *SourceTree* необходимо при открытой вкладке репозитория нажать кнопку «*Push*» в панели инструментов. Отобразится окно «*Push*» (рис. 18).

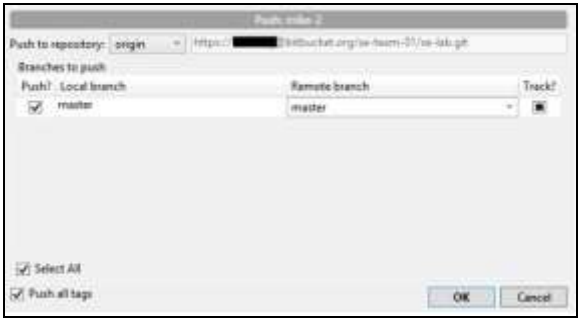

Рис. 18. Окно отправки изменений в удаленный репозиторий (Push)

В выпадающем списке «*Push to repository*» должен быть выбран удаленный репозиторий, в который загружаются изменения. В нашем случае это «*origin*».

В списке «*Branches to push*» галочкой должно быть отмечено «*master*».

Рис. 19. Окно с сообщением об отказе в принятии изменений удаленным репозиторием

Чтобы начать загрузку необходимо нажать кнопку *«OK»*. После завершения загрузки у последнего коммита рядом с меткой *«master»* отобразится метка «*origin/master*».

Следует помнить, что удаленный репозиторий примет изменения, только если все его изменения уже есть в локальном репозитории. Иначе говоря, отправка допускается, только если с момента последнего получения изменений никто другой своих изменений не отправлял. Если это условие не выполняется, то операция *push* отклоняется, и отображается сообщение об ошибке (рис. 19).

В случае этой ошибки следует сначала получить изменения из удаленного репозитория (*pull*) и выполнить слияние, а затем повторить операцию отправки (*push*).

Добавление метки

Посредством меток (tags) можно выделить определенные коммиты в истории. Чаще всего метки используются для маркировки состояний проекта на базе которых осуществляется выпуск версий программы.

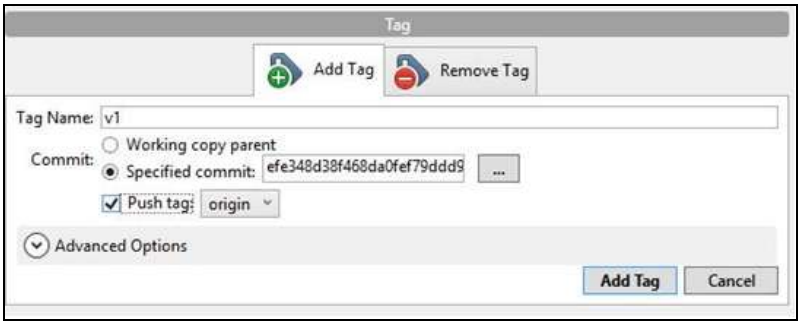

Рис. 20. Окно работы с метками «Tags»

Для добавления метки в приложении *SourceTree* необходимо в истории коммитов кликнуть правой кнопкой по нужному коммиту и в контекстном меню выбрать пункт «*Tag…*». Отобразится окно «*Tag*» (рис. 20).

В текстовое поле «*Tag Name*» необходимо ввести наименование метки.

В поле «*Commit*» необходимо выбрать коммит, который будет помечен. При выборе опции «*Working copy parent*» метка помещается на коммит, на базе которого сформирован рабочий каталог (обычно это самый последний коммит). При выборе опции «*Specified commit*» можно самостоятельно выбрать помечаемый коммит из истории. В случае создания метки из истории коммитов в окне уже выбрана последняя опция и указан нужный коммит.

В домашнем задание требуется, чтобы метка после создания была синхронизирована с репозиторием BitBucket. Для этого необходимо поставить галочку «*Push tag*» и в выпадающем списке выбрать удаленный репозиторий «*origin*».

После указания всех настроек необходимо нажать кнопку «*Add Tag*». После завершения процесса создания метки и синхронизации, история коммитов обновится, и в ней отобразится созданная метка.

История изменений

Для просмотра истории изменений в приложении *SourceTree* необходимо открыть вкладку репозитория и в разделе *«Branches»* выбрать «*master*» (см. рис. 21, маркер «1»).

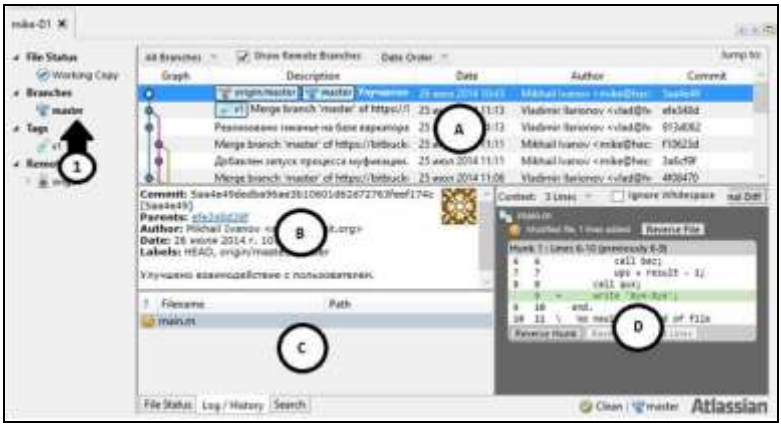

Рис. 21. Вкладка репозитория

В области «A» представлена история изменений (коммитов) в виде графа. Меткой «*master*» помечен последний коммит в локальный репозиторий, меткой «*origin/master*» – последний синхронизированный коммит из удаленного репозитория.

В области «B» отображается детальная информация о выделенном в области «A» коммите.

В области «C» отображается перечень изменений файлов, вошедший в выделенный коммит.

В области «D» отображаются изменения, внесенные в выделенный в области «C» файл.

Визуализация этой истории работы приложением *SourceTree* представлена на рис. 22

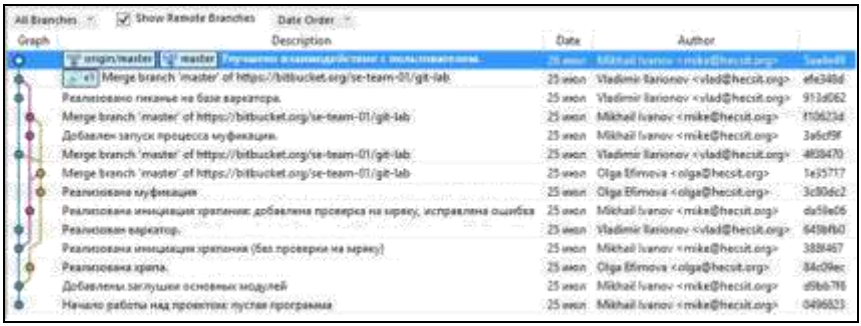

Рис. 22. Пример истории изменений

Git позволяет не только просматривать историю изменений, но и «переместиться» в любой момент этой истории.

Для того, чтобы привести рабочий каталог в соответствие некоторому коммиту, нужно выделить требуемый коммит и в контекстном меню выбрать пункт «*Checkout…*» и в диалоге «*Confirm change working copy*» нажать кнопку «*OK*».

ВАРИАНТЫ ЗАДАНИЙ

Варианты №№ 1, 7, 13, 19

Квадратную матрицу 6х6 заполнить случайными числами из диапазона [-17..20], вывести ее на экран.
Найти максимальный из элементов, расположенных на главной диагонали, вывести на экран его позицию (строку и столбец) и величину.
Найти максимальный из элементов, расположенных ниже главной диагонали, вывести на экран его позицию и величину.
Заменить максимальный из элементов, расположенных на главной диагонали, на максимальный из элементов, расположенных ниже главной диагонали. Вывести результирующую матрицу на экран.

Варианты №№ 2, 8, 14, 20

Квадратную матрицу 8х8 заполнить случайными числами из диапазона [-11..9], вывести ее на экран.
Найти минимальный из элементов, расположенных на побочной диагонали, вывести на экран его позицию (строку и столбец) и величину.
Найти минимальный из элементов, расположенных ниже побочной диагонали, вывести на экран его позицию и величину.
Заменить минимальный из элементов, расположенных на побочной диагонали, на минимальный из элементов, расположенных ниже побочной диагонали. Вывести результирующую матрицу на экран.

Варианты №№ 3, 9, 15, 21

Квадратную матрицу 7х7 заполнить случайными числами из диапазона [-14..12], вывести ее на экран.
Найти и вывести на экран количество отрицательных элементов на главной диагонали.
Заменить все положительные элементы выше побочной диагонали на 0. Вывести результирующую матрицу на экран.

Варианты №№ 4, 10, 16, 22

Квадратную матрицу 9x9 заполнить случайными числами из диапазона [-21..23], вывести ее на экран.
Найти и вывести на экран количество положительных элементов на побочной диагонали.
Заменить все отрицательные элементы ниже главной диагонали на 0. Вывести результирующую матрицу на экран.

Варианты №№ 5, 11, 17, 23

Квадратную матрицу 7x7 заполнить случайными числами из диапазона [-5..3], вывести ее на экран.
Найти максимальный из элементов, расположенных на побочной диагонали, вывести на экран его позицию (строку и столбец) и величину.
Найти и вывести на экран среднее арифметическое элементов, расположенных ниже главной диагонали.
Заменить максимальный из элементов, расположенных на побочной диагонали, на среднее арифметическое элементов, расположенных ниже главной диагонали. Вывести результирующую матрицу на экран.

Варианты №№ 6, 12, 18, 24

Квадратную матрицу 5x5 заполнить случайными числами из диапазона [-7..5], вывести ее на экран.
Найти минимальный из элементов, расположенных на главной диагонали, вывести на экран его позицию (строку и столбец) и величину.
Найти и вывести на экран среднее арифметическое элементов, расположенных ниже побочной диагонали.
Заменить минимальный из элементов, расположенных на главной диагонали, на среднее арифметическое элементов, расположенных ниже побочной диагонали. Вывести результирующую матрицу на экран.

ФОРМА ОТЧЕТА ПО ДОМАШНЕЙ РАБОТЕ

На выполнение домашней работы отводится 8 академических часа: 7 часов на выполнение и сдачу домашней работы и 1 часа на подготовку отчета.

Номер варианта студенту выдается преподавателем.

Отчет на защиту предоставляется в печатном виде.

- Структура отчета (на отдельном листе(-ах)): титульный лист, оглавление, цели и задачи выполнения домашней работы, формулировка задания (вариант), текст варианта программы и результаты ее работы; окно истории изменения разрабатываемой программы, демонстрирующее коллективную работу; вывод; список литературы.

КОНТРОЛЬНЫЕ ВОПРОСЫ:

1. Для чего при разработке программных систем используются система контроля версий?
2. Опишите устройство и общий порядок работы с централизованной системой контроля версий.
3. Назовите особенности распределенных систем контроля версий.
4. В чем преимущества распределенных систем контроля версий по сравнению с централизованными?
5. Приведите примеры распределенных систем контроля версий.
6. Что такое репозиторий?
7. В каких состояниях может быть файл с точки зрения Git?
8. Что такое рабочий каталог (working directory)?
9. Что такое область подготовленных файлов (staging area) или индекс (index)?
10. Поясните назначение списка игнорирования Git. Какие файлы/папки обычно в него вносятся?
11. Опишите назначение команды «Clone».
12. Опишите назначение команды «Add».
13. Опишите назначение команды «Commit».
14. Опишите назначение команды «Checkout».
15. Опишите назначение команды «Pull».
16. Опишите назначение команды «Fetch».
17. Опишите назначение команды «Push».
18. Опишите назначение команды «Log».
19. Что такое метки (tags)? Как они обычно используются?
20. Сформулируйте правила, которым необходимо следовать при работе с системой контроля версий.
21. Для чего при разработке программных систем используются сервисы хостинга проектов?

ЛИТЕРАТУРА

Основная литература

1. Киселева, Т. В. Программная инженерия. Часть 1 [Электронный ресурс]: учебное пособие / Т. В. Киселева. — Электрон. текстовые данные — Ставрополь : Северо-Кавказский федеральный университет, 2017. — 137 с.: http://www.iprbookshop.ru/69425.html
2. Программная инженерия.Часть II [Электронный ресурс]: учебное пособие / составители Т. В. Киселева. — Электрон. текстовые данные — Ставрополь : Северо-Кавказский федеральный университет, 2017. — 100 с.: http://www.iprbookshop.ru/83193.html
3. Полетайкин, А. Н. Учебно-методическое пособие по выполнению лабораторных работ по дисциплине «Программная инженерия». Часть I. Реализация жизненного цикла программного обеспечения [Электронный ресурс]: учебно-методическое пособие / А. Н. Полетайкин. — Электрон. текстовые данные — Новосибирск : Сибирский государственный университет телекоммуникаций и информатики, 2016. — 97 с: http://www.iprbookshop.ru/69565.html

Дополнительная литература

1. Мейер, Б. Объектно-ориентированное программирование и программная инженерия [Электронный ресурс]/ Б. Мейер. — 3-е изд. — Электрон. текстовые данные — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Эр Медиа, 2019. — 285 с.: http://www.iprbookshop.ru/79706.html