Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

УТВЕРЖДАЮ
Зав. каф АОИ, д.т.н., проф.
_____ Ю.П. Ехлаков
«___» _____ 2017 г.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ И САМОСТОЯТЕЛЬНОЙ РАБОТЫ

по дисциплине «УПРАВЛЕНИЕ ЖИЗНЕННЫМ ЦИКЛОМ ПРОГРАММНЫХ ПРОДУКТОВ»

для студентов направления подготовки

«Бизнес информатика»

Разработчик:
ассистент каф. АОИ
_____ В.С. Масляев

Томск 2017

СОДЕРЖАНИЕ

Введение

Данное учебно-методическое пособие предназначено для подготовки и выполнения лабораторных работ по дисциплине «Управление жизненным циклом программных продуктов» для студентов направления «Бизнес информатика».

Лабораторные работы по данной части дисциплины имеют целью: закрепление теоретического материала, получение навыков самостоятельной работы с системой контроля версий Git.

Цель изучения дисциплины «Управление жизненным циклом программных продуктов» является формирование у студентов профессиональных знаний, умений и навыков о методах и средствах управления жизненным циклом программных продуктов, использование информационных технологий на всех стадиях их жизненного цикла.

Задачи:

получение практических и теоретических навыков использования информационных технологий на всех этапах жизненного цикла программных продуктов;

формирование умений решения задач хранения информации на различных этапах жизненного цикла;

получение опыта управления жизненным циклом программных продуктов;

приобретение навыков использования систем контроля версий в области управления жизненным циклом программных продуктов;

изучение современных информационных технологий необходимых для управления проектами.

Лабораторная работа №1 «Первоначальная настройка git»

Тема: Первоначальная настройка git. Инициализация каталога. Состояния фалов в git. Первый коммит.

Цель работы: провести первоначальную настройку системы контроля версии git, после установки инициализировать каталог для работы, разобраться с существующими состояниями файлов в git, сделать первый коммит.

Продолжительность: 4 часа.

В состав git'a входит утилита **git config**, которая позволяет просматривать и устанавливать параметры, контролирующие все аспекты работы git'a и его внешний вид.

Первое, что необходимо сделать после установки git'a, — указать имя и адрес электронной почты. Это важно, потому что каждый коммит в git'e содержит эту информацию, и она включена в коммиты, передаваемые разработчиками, и не может быть далее изменена.

Для того чтобы начать использовать git для существующего проекта, необходимо перейти в проектный каталог и в командной строке ввести **git init**.

Эта команда создаёт в текущем каталоге новый подкаталог с именем .git содержащий все необходимые файлы репозитория — основу git-репозитория. На этом этапе проект ещё не находится под версионным контролем. Данная команда инициализирует возможность работы с git, но не вносит файлы под контроль.

Порядок выполнения работы

1. Изучить теоретическую часть работы.
2. Зайти в папку T://{Номер группы} и в ней создать папку соответствующую инициалам студента на английском языке. Например, для студента Иванов Петр Петрович, папка будет иметь имя IPP.
3. Провести инициализацию репозитория в созданной папке. Для этого, открыть программу **Git Bash**, перейти в созданную папку (для перемещения используется команда **cd T://{Номер группы}/{Инициалы}**).
4. Установить настройки имени и e-mail’a, не используя опцию **--global.**
5. Создать в папке файл my_first_file.txt и проиндексировать его.
6. Сделать первый коммит.
7. Открыть файл my_first_file.txt и добавить в него строку “test row”. Проиндексировать изменения.
8. Создать новый файл my_second_file.txt. Проиндексировать изменения.
9. Сделать второй коммит.
10. Продемонстрировать преподавателю ход работы, ответить на уточняющие вопросы.

Лабораторная работа № 2 «Игнорирование, сравнение, удаление и перемещение файлов»

Тема: Игнорирование файлов. Сравнение изменений. Удаление и перемещение файлов.

Цель работы: научиться исключать файлы, которые нет необходимости вести в системе контроля версий. Получить практические навыки сравнения проделанных изменений в файлах.

Продолжительность: 4 часа.

Зачастую, имеется группа файлов, которые не только нет необходимости автоматически добавлять в репозиторий, но и видеть в списках неотслеживаемых. К таким файлам обычно относятся автоматически генерируемые файлы (различные логи, результаты сборки программ и т.п.). В таком случае, необходимо создать файл *.gitignore* с перечислением шаблонов соответствующих таким файлам.

К шаблонам в файле .gitignore применяются следующие правила:

- Пустые строки, а также строки, начинающиеся с # (символ комментария), игнорируются.
- Можно использовать стандартные glob шаблоны.
- Можно заканчивать шаблон символом слэша (/) для указания каталога.
- Можно инвертировать шаблон, использовав восклицательный знак (!) в качестве первого символа.

Glob-шаблоны представляют собой упрощённые регулярные выражения используемые командными интерпретаторами. Символ * соответствует 0 или более символам; последовательность [abc] — любому символу из указанных в скобках (в данном примере a, b или c); знак вопроса (?) соответствует одному символу; [0-9] соответствует любому символу из интервала (в данном случае от 0 до 9).

Порядок выполнения работы

1. Изучить теоретическую часть работы.
2. Продолжить работу с созданным репозиторием на первой лабораторной работе.
3. Создать папку temp в своем репозитории.
4. Создать папку log и добавить в нее 2 файла: main.html и some.tmp.
5. Создать файл .gitignore и добавить в игнорирование папку temp и файлы с расширением .tmp из папки log.
6. Закоммитить добавление файла .gitignore.
7. Внести изменения в файл my_first_file.txt, добавив строку “row to index”, проиндексировать данные изменения. Еще раз внести изменения в файл, добавив строку “row no index”.
8. Посмотреть индексированные и неиндексированные изменения используя команду **git diff**.
9. Удалить файл my_first_file.txt, зафиксировать данное удаление.

10. Переименовать файл my_second_file.txt в my_first_file.txt, зафиксировать изменение.
11. Продемонстрировать преподавателю ход работы, ответить на уточняющие вопросы.

Лабораторная работа №3 «Просмотр истории коммитов»

Тема: Просмотр истории коммитов, команда git log.
Цель работы: освоить механизм работы с командой git log для получения информации об истории коммитов.
Продолжительность: 4 часа.

После того как будет создано несколько коммитов, вероятнее всего появится необходимость просмотреть, что же происходило с этим репозиторием. Наиболее простой и в то же время мощный инструмент для этого — команда **git log**.

По умолчанию, без аргументов, **git log** выводит список коммитов созданных в данном репозитории в обратном хронологическом порядке. То есть самые последние коммиты показываются первыми.

Один из наиболее полезных параметров — это **-p**, который показывает дельту (разницу/diff), привнесенную каждым коммитом. Также можно использовать **-2**, что ограничит вывод до 2-х последних записей.

Порядок выполнения работы

1. Изучить теоретическую часть работы.
2. Продолжить работу с созданным репозиторием.
3. Изучить возможности команды **git log**, выполнить различные варианты вывода информации и ее отбора.
4. Выполнить задание согласно варианту.
5. Продемонстрировать преподавателю ход работы, ответить на уточняющие вопросы.

Варианты

Задание	Номер
Вывести коммиты, автором которых являетесь Вы, за последний месяц.	1
Вывести все коммиты в формате: короткий хеш, автор, комментарий.	2
Вывести все коммиты, в сообщении которых присутствует слово my.	3
Вывести все коммиты за текущий месяц с информацией о том, какие файлы были изменены.	4
Вывести информацию о первом коммите в системе, с выводом дельты	5

(diff).	
Вывести коммиты сделанные за последний месяц назад, но исключая последнюю неделю.	6
Вывести информацию о коммитах в формате: автор, дата, список измененных файлов.	7
Вывести коммиты, автором которых являетесь Вы, с выводом дельты (diff).	8
Вывести коммиты, в которых происходили изменения файла my_first_file.txt, за последние 2 недели.	9
Вывести последние 3 коммита в формате: автор, комментарий.	10
Вывести информацию о первом коммите в формате: дата, автор, комментарий, а также список измененных файлов.	11
Вывести коммиты, автором которых являетесь Вы, со списком измененных файлов.	12
Вывести все коммиты за текущий месяц в формате: сокращенный хеш, дата, комментарий.	13
Вывести все коммиты в формате: e-mail автора, дата коммита, хеши родительских коммитов.	14
Вывести коммиты, в которых происходили изменения файла my_first_file.txt.	15

Лабораторная работа №4 «Отмена изменений. Работа с метками»

Тема: Отмена внесенных изменений. Работа с метками.

Цель работы: научиться отменять сделанные изменения, работать с метками.

Продолжительность: 4 часа.

Одна из типичных отмен происходит тогда, когда коммит сделан слишком рано, например, не были добавлены какие-либо файлы, или перепутан комментарий к коммиту. Если необходимо сделать этот коммит ещё раз, можно выполнить **git commit** с опцией **–amend.**

Эта команда берёт индекс и использует его для коммита. Если после последнего коммита не было никаких изменений (например, приведенная команда была запущена сразу после предыдущего коммита), то состояние проекта будет абсолютно таким же и всё, что изменится, это комментарий к коммиту.

Git использует два основных типа меток: легковесные и аннотированные. Легковесная метка — это что-то весьма похожее на ветку, которая не меняется —

это просто указатель на определённый коммит. А вот аннотированные метки хранятся в базе данных Git'a как полноценные объекты. Они имеют контрольную сумму, содержат имя поставившего метку, e-mail и дату, имеют комментарий и могут быть подписаны и проверены с помощью GNU Privacy Guard (GPG). Обычно рекомендуется создавать аннотированные метки, чтобы иметь всю перечисленную информацию; но если необходимо сделать временную метку или по какой-то причине нет необходимости сохранять остальную информацию, то для этого годятся и легковесные метки.

Порядок выполнения работы

1. Изучить теоретическую часть работы.
2. Продолжить работу с созданным репозиторием.
3. Создать три файла: 1.txt, 2.txt, 3.txt.
4. Проиндексировать первый файл и сделать коммит с комментарием “add 1.txt file”.
5. Проиндексировать второй и третий файлы.
6. Удалить из индекса второй файл.
7. Перезаписать уже сделанный коммит с новым комментарием “add 1.txt and 3.txt”
8. Создать аннотированную метку с названием v0.01.
9. Создать легковесную ветку указывающую на первый коммит в репозитории.
10. Продемонстрировать преподавателю ход работы, ответить на уточняющие вопросы.

Лабораторная работа №5 «Ветвление. Конфликты»

Тема: Работа с ветками, решение конфликтов.

Цель работы: научиться создавать ветки, перемещаться по ним, объединять и удалять их. Решать конфликты слияния.

Продолжительность: 8 часа.

Ветка в git'e — это просто легковесный подвижный указатель на один из коммитов. Ветка по умолчанию в git'e называется **master**. Когда происходит создание коммита на начальном этапе, доступна ветка **master**, указывающая на последний сделанный коммит. При каждом новом коммите она сдвигается вперёд автоматически.

Для того чтобы создать новую ветку используется команда **git branch**.

Эта команда создаст новый указатель на тот самый коммит, на котором сейчас находится git.

Порядок выполнения работы

1. Изучить теоретическую часть работы.
2. Продолжить работу с созданным репозиторием.
3. Создать новую ветку my_first_branch.
4. Перейти на ветку и создать новый файл in_branch.txt, закоммитить изменения.
5. Вернуться на ветку master.
6. Создать и сразу перейти на ветку new_branch.
7. Сделать изменения в файле 1.txt, добавить строку “new row in 1.txt file”, закоммитить изменения.
8. Перейти на ветку master и слить ветки master и my_first_branch, после чего слить ветки master и new_branch.
9. Удалить ветки my_first_branch и new_branch.
10. Создать ветки branch_1 и branch_2.
11. Перейти на ветку branch_1 и изменить файл 1.txt, удалить все содержимое и добавить текст “fix in 1.txt”, изменить файл 3.txt, удалить все содержимое и добавить текст “fix in 3.txt”, закоммитить изменения.
12. Перейти на ветку branch_2 и также изменить файл 1.txt, удалить все содержимое и добавить текст “My fix in 1.txt”, изменить файл 3.txt, удалить все содержимое и добавить текст “My fix in 3.txt”, закоммитить изменения.
13. Слить изменения ветки branch_2 в ветку branch_1.
14. Решить конфликт файла 1.txt в ручном режиме, а конфликт 3.txt используя команду git mergetool с утилитой Meld.
15. Продемонстрировать преподавателю ход работы, ответить на уточняющие вопросы.

Лабораторная работа №6 «Прятанье»

Тема: Механизм прятанья.

Цель работы: научиться использовать механизм прятанья, а также расширить знания в управлении веток.

Продолжительность: 4 часа.

Часто возникает такая ситуация, что пока идет работа над частью своего проекта, всё находится в беспорядочном состоянии, а нужно переключить ветки, чтобы немного поработать над чем-то другим. Проблема в том, что делать коммит с наполовину доделанной работой только для того, чтобы позже можно было вернуться в это же состояние не хотелось бы. Ответ на эту проблему — команда **git stash**.

Прятанье поглощает грязное состояние рабочего каталога, то есть изменённые отслеживаемые файлы и изменения в индексе, и сохраняет их в стек незавершённых изменений, которые потом в любое время можно снова применить.

Порядок выполнения работы

1. Изучить теоретическую часть работы.
2. Продолжить работу с созданным репозиторием.
3. Проверить какие ветки слиты с веткой master, а какие нет.
4. Удалить все ветки слитые с master.
5. Создать новую ветку work и перейти в нее.
6. Изменить файл 1.txt.
7. Спрятать данные изменения.
8. Развернуть обратно данные изменения с опцией --index.
9. Развернуть спрятанные изменения в новую ветку.
10. Продемонстрировать преподавателю ход работы, ответить на уточняющие вопросы.

Лабораторная работа №7 «Работа с удаленным репозиторием»

Тема: Работа с удаленным репозиторием. Github.com.

Цель работы: научиться работать с удаленным репозиторием, использовать платформу github.com.

Продолжительность: 8 часа.

Если необходимо получить копию существующего репозитория Git, например, проекта, в котором разработчик планирует поучаствовать, то необходимо использовать команду **git clone**. Каждая версия каждого файла из истории проекта забирается (*pulled*) с сервера, когда выполняется команда **git clone**. Фактически, если серверный диск выйдет из строя, можно использовать любой из клонов на любом из клиентов, для того чтобы вернуть сервер в то состояние, в котором он находился в момент клонирования.

Клонирование репозитория осуществляется командой **git clone [url]**.

Порядок выполнения практической работы

1. Изучить теоретическую часть работы.
2. Продолжить работу с созданным репозиторием.
3. Пройти регистрацию на сайте github.com.
4. Настроить доступ к github по SSH.
5. Склонировать репозиторий git@github.com:mavose/tusur_{номер группы}.git.

6. Продемонстрировать преподавателю ход работы, ответить на уточняющие вопросы.

Методические указания по выполнению самостоятельной работы

Самостоятельная работа предусмотрена учебным планом. Цель самостоятельной работы студента в рамках курса — закрепление и расширение знаний, полученных во время проведения аудиторных занятий.

Согласно рабочей программе в качестве самостоятельной работы для студентов являются следующие виды:

- проработка лекционного материала;
- подготовка к лабораторным работам;
- выполнение индивидуального задания.

Проработка лекционного материала осуществляется студентом с использованием конспекта лекций и рекомендуемых учебников.

Для подготовки к лабораторным работам студент должен изучить теоретический материал согласно теме лабораторной работы, подготовиться к практической части реализации за компьютером.

В качестве индивидуального задания, перед студентом ставится задача подготовить доклад с презентацией на выбранную тему. Выполнение индивидуального задания является формой самостоятельной деятельности студента, позволяющей раскрыть его способности по работе с литературными и (или) иными источниками.

Для доклада предлагаются следующие темы:

1. Обзор системы управления проектами Jira.
2. Обзор Helpdesk системы Freshservice.
3. Подход к разработке Agile.
4. Обзор системы контроля версий SVN.
5. Обзор системы управления проектами Redmine.
6. Методология Gir flow.
7. Обзор системы контроля версий Git.
8. Сравнение систем контроля версий Git и SVN.
9. Обзор платформы github.com.
10. Обзор Helpdesk системы Kayako.

Визуальные клиенты git.

Рекомендуемая литература

1. Ехлаков Ю.П. Управление программными проектами: учебник. – Томск : Изд-во Томск. гос. ун-та систем управления и радиоэлектроники, 2015. – 216 с. [Электронный ресурс]: научно-образовательный портал ТУСУРа. – URL: https://edu.tusur.ru/training/publications/6024

2. Ехлаков Ю.П. Организация бизнеса на рынке программных продуктов: учебник. – Томск: Изд-во Томск. гос. ун-та систем управления и радиоэлектроники, 2012. – 312 с. **гриф УМО** [Электронный ресурс]: научно-образовательный портал ТУСУРа. – URL: http://edu.tusur.ru/training/publications/970