Оглавление

[Общие понятия систем контроля версий 1](#_Toc79405411)

[Порядок работы с Git 3](#_Toc79405412)

[Установка, настройка 3](#_Toc79405413)

[Сценарий 1: Выгрузка проекта в репозиторий. 4](#_Toc79405414)

[Сценарий 2: Выгрузка локального репозитория в удаленный (Github). 6](#_Toc79405415)

[Сценарий 3: Клонирование репозитория. 7](#_Toc79405416)

[Сценарий 4: Выгрузка изменений в удаленный репозиторий 8](#_Toc79405417)

[Сценарий 5: Загрузка изменений из удаленного репозитория. 8](#_Toc79405418)

[Сценарий 6: Создание ответвления от основного проекта. 9](#_Toc79405419)

[Основные команды 10](#_Toc79405420)

[Задание 10](#_Toc79405421)

[Дополнительное задание 11](#_Toc79405422)

[Контрольные вопросы 12](#_Toc79405423)

[Список литературы 13](#_Toc79405424)

Полная документация по git: [**https://git-scm.com/book/ru/v2**](https://git-scm.com/book/ru/v2)

Далее приведена краткая инструкция, показывающая основы работы с этой системой.

# Общие понятия систем контроля версий

В процессе разработки программного обеспечения достаточно часто возникает ситуация, когда необходимо иметь несколько версий одного и того же продукта. В простейшем случае можно просто хранить несколько вариантов файлов исходного кода, нумеруя их соответствующим образом. Однако, такой способ неэффективен (приходится хранить несколько практически идентичных копий), требует повышенного внимания и дисциплины и часто ведёт к ошибкам. В случае, если одна из версий содержала ошибку, но до ее обнаружения, на базе этой версии было создано еще несколько, то исправление этой ошибки должно так же затрагивать и все созданные копии. Кроме этого, при таком способе организации затруднена совместная работа нескольких человек – чтобы избежать одновременного редактирования файлов, будет необходимо блокировать открытые файлы на все время работы (в противном случае высока вероятность перезаписать изменения, сделанные другими участниками), что снижает общую продуктивность команды.

Для решения этих задач были разработаны системы управления версиями, отслеживающие изменения проекта и возможные конфликты, и предлагающие средства их решения. Большинство систем может автоматически объединить (слить) изменения, сделанные разными разработчиками. Однако такое автоматическое объединение изменений обычно возможно только для текстовых файлов и при условии, что изменялись разные (непересекающиеся) части этого файла. Такое ограничение связано с тем, что большинство систем управления версиями ориентированы на поддержку процесса разработки программного обеспечения, а исходные коды программ хранятся в текстовых файлах. Если автоматическое объединение выполнить не удалось, система может предложить решить проблему вручную.

Введем используемую в системах контроля версий терминологию. Набор файлов, с которым мы работаем в данный момент, называется рабочая копия (*working copy*). После того, как решено, что все нужные изменения на данный момент внесены, и об этом можно сообщить системе контроля версий, разработчик производит отправку изменений в репозиторий (*repository*). Репозиторий – это хранилище для нашего проекта, которое обслуживает система контроля версий. Сама операция отправки изменений называется *commit*, на русском языке ее так и называют – коммит. Если нам необходимо взять данные из репозитория, то мы осуществляем операцию *checkout*, которая на русском произносится как чекаут.

Для одновременной работы группы разработчиков над одним проектом необходимо наличие общего репозитория, в который будут «сливаться» изменения отдельных разработчиков. Разные системы решают эту задачу по-разному, но большее распространение получила децентрализованная архитектура систем контроля версий, общий вид которой показан на рисунке 1. Для хранения общей копии проекта создаются один или несколько репозиториев на удаленных серверах, и репозитории, размещенные на рабочих компьютерах разработчиков. Таким образом, каждый разработчик имеет свою локальную копию репозитория и соответственно проекта, и может производить с ним какую-то работу, не затрагивая код других участников, а затем, при необходимости, отправлять их на удаленный сервер (команда push), и/или загружать актуальную версию (команды fetch/pull).

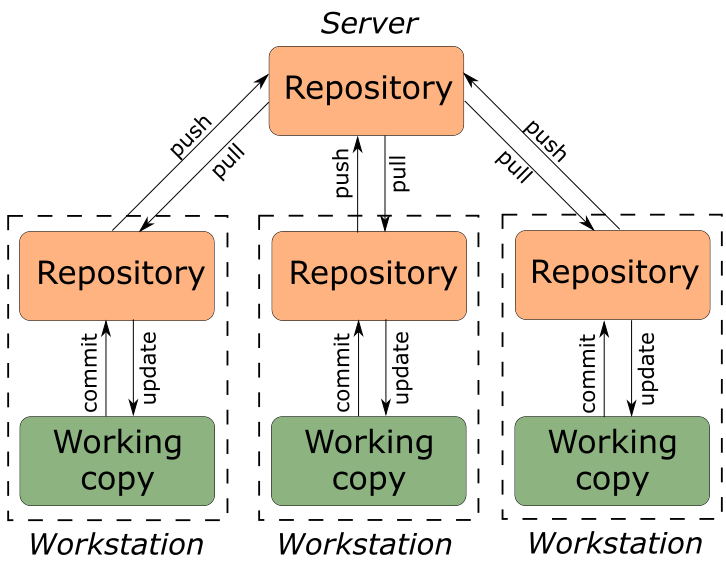


Рисунок 1- Архитектура распределенных систем контроля версий

Одним из представителей таких систем является Git, работа с которой будет рассмотрена далее.

# Порядок работы с Git

## Установка, настройка

Для начала необходимо проверить, установлен ли git, для этого достаточно выполнить в командной строке команду “git”. Если установка завершена корректно, то будет показан список дополнительных опций (рисунок 3). Если вместо этого вы получили сообщение «git не является внутренней или внешней командой, исполняемой программой или пакетным файлом», то необходимо произвести установкуt.

Для скачивания git перейдите на страницу <https://git-scm.com/> и выберите подходящую версию, затем запустите скачанный дистрибутив, для всех предлагаемых параметров можно оставить значения по умолчанию. Со значением параметров можно ознакомиться в [2].

После установки снова проверьте работу команды git, если и в этот раз ответ был неудовлетворительным, пропишите путь к исполняемым файлам git в переменную среды окружения path.

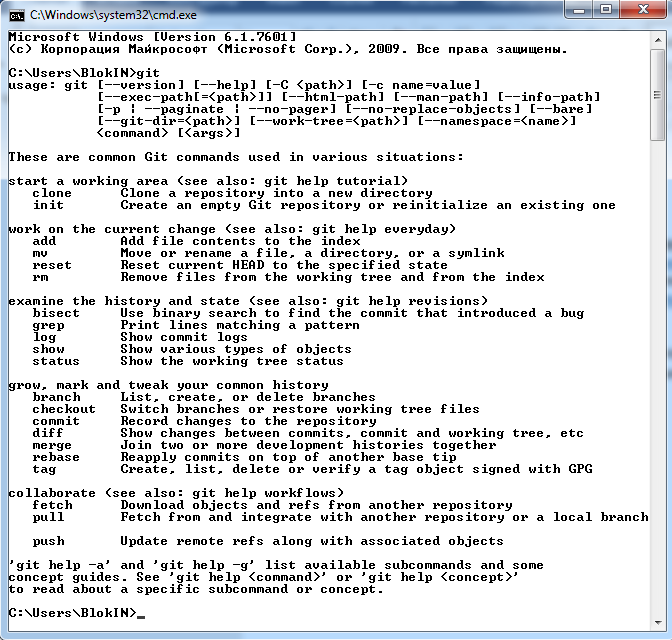


Рисунок 2 – Проверка установки Git под Windows

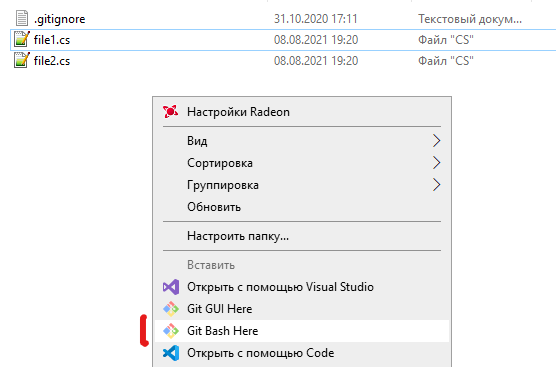
Далее будет показан порядок работы с консольной версией git’а, т. к. она обладает большей гибкость при работе по сравнению со встроенным в Visual Studio средствами и другими утилитами. Будут рассмотрены базовые сценарии, с которыми вы можете столкнуться в начале работы с этой системой.

## Сценарий 1: Выгрузка проекта в репозиторий.

*В каких случаях применять*: создан новый проект, который необходимо «выгрузить» в локальную систему контроля версий.

Переходим в папку проекта и копируем туда файл .gitignore (можно взять из папки с материалами к лабораторным работам). Этот файл определяет типы файлов, имена папок, для которых не следует отслеживать изменения. Как правило, к ним относятся различные временные файлы, пересоздаваемые во время каждой сборки проекта, и отслеживание изменения которых может существенно увеличить размер хранилища.

Затем из этой папки вызовем Git Bash через контекстное меню проводника:



*Рисунок 3 – Вызов Git Bash*

В открывшейся консоли, чтобы создать новый репозиторий, введите команду:

git init

Это создаст репозиторий, однако, сразу после создания он будет не синхронизирован с содержимым рабочей директории. Чтобы добавить существующие файлы в репозиторий необходимо выполнить последовательность из двух команд:

git add .

git commit -m “Описание изменений”

Команда git add позволяет добавить в индекс новые, а также измененные файлы (индексированные файлы могут быть затем сохранены в репозитории, в противном случае git их «не увидит»). После add должно идти имя файла, изменения которого должны отслеживаться, либо . для отслеживания всех вложенных файлов и папок.

commit выполняет непосредственно сохранение изменений в репозитории. Чтобы отличать эти изменения между собой, следует задать краткое описание того, что было сделано с помощью параметра -m. Если этот параметр не указывать, то будет открыт текстовый редактор по умолчанию для описания фиксируемых изменений.

Необходимость сразу двух команд связана с тем, что git оперирует тремя состояниями, в которых могут находиться файлы: зафиксированном (committed), изменённом (modified) и подготовленном (staged). К изменённым относятся файлы, которые изменялись, но ещё не были зафиксированы. Подготовленные файлы — это изменённые файлы, отмеченные для включения в следующий коммит (после выполнения git add).



Рисунок 4 - Состояния файлов проекта в git

Просмотреть текущее состояние файлов рабочей директории и историю фиксаций (commit’ов) можно с помощью команд git status и git log:

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

*Рисунок 5 – Результат выполнения команд git status и git log*

## Сценарий 2: Выгрузка локального репозитория в удаленный (Github).

*В каких случаях применять*: создан локальный репозиторий, который необходимо выгрузить в удаленный (например, на Github).

Для начала создайте новый репозиторий на github.com, при создании сняв переключатели «Add a Readme file» и «Add .gitignore» (чтобы создать пустой репозиторий, не содержащий никаких файлов):

Изображение выглядит как текст

Автоматически созданное описание

*Рисунок 6 – Создание нового проекта на github*

После создания репозитория вы получите ссылку вида: <https://github.com/blockin90/TestRepo2.git> , которая является адресом вашего удаленного репозитория. На этой же странице будет показан необходимый для выгрузки локального репозитория набор команд (в разделе «…or push an existing repository from the command line»):

git remote add origin https://github.com/blockin90/TestRepo2.git

git branch -M main

git push -u origin main

git remote добавляет новый репозиторий для синхронизации под именем origin,

git branch создает новую ветку (об этом позднее) с названием main,

git push выгружает изменения ветки main в удаленный репозиторий, соответствующий имени origin.

Если все сделано правильно, то после выполнения этих команд, актуальное состояние проекта должно отобразиться на github’е:

Изображение выглядит как текст

Автоматически созданное описание

*Рисунок 7 – внешний вид репозитория из веб-интерфейса gitub.com*

## Сценарий 3: Клонирование репозитория.

*В каких случаях применять*: Проект уже на GitHub’е, нужно его скопировать на другой компьютер.

Запускаем git bash в папке, в которой вы хотите разместить проект, и вводим команду git clone адрес\_репозитория

Изображение выглядит как текст

Автоматически созданное описание

*Рисунок 8 – результат выполнения git clone*

Эта команда выполняет копирование удаленного репозитория в локальную папку с названием, соответствующим имени репозитория (для скриншота выше это будет TestRepo2).

Когда репозиторий склонирован, для продолжения работы с ним, необходимо перейти в него командой (вместо TestRepo2 нужно подставить имя своей папки):

*cd* TestRepo2

## Сценарий 4: Выгрузка изменений в удаленный репозиторий

*В каких случаях применять*: Проект в локальном репозитории + на GitHub’е, в него внесены какие-то изменения, необходимо их зафиксировать, а затем выгрузить в удаленный репозиторий

В git bash введите команды:

git add .

git commit -m “Описание изменений”

Это позволит зафиксировать изменения локально. Для их отправки на github необходимо там же выполнить:

git push origin main

, где origin – имя, соответствующее удаленному репозиторию (подробнее в предыдущем разделе), main – название отправляемой ветки.

Изображение выглядит как текст

Автоматически созданное описание

*Рисунок 9 – результат выполнения git push*

## Сценарий 5: Загрузка изменений из удаленного репозитория.

*В каких случаях применять*: Проект уже на GitHub’е, а также на вашей машине. Вы, или кто-то другой, внесли правки на другом компьютере, и они были выгружены на GitHub согласно предыдущему описанию, и теперь вы хотите подтянуть их к себе на компьютер (только последние правки, не весь проект целиком).

В git bash введите команду:

git pull origin main

значение параметров origin и main здесь аналогичны предыдущем разделу.

## Сценарий 6: Создание ответвления от основного проекта.

*В каких случаях применять*: Проект находится в репозитории, необходимо сделать правки, не меняя исходное состояние проекта, и иметь возможность в любой момент их откатить и/или переключиться на другой набор изменений.

В некоторых случаях бывает нужно выполнить определенную работу, не затрагивая основной проект до тех пор, пока эта работа не будет выполнена до конца. Такая ситуация может возникать в условиях командной работы над одним проектом, правок ошибок, внедрения новых, экспериментальных функций. В этих ситуациях нужно иметь рабочую версию проекта и при этом параллельно вести разработку нового функционала.

Для этих целей в системах контроля версий введено понятие ветвления. Если кратко, ветвь – это набор изменений (коммитов) под заданным именем. У каждой ветки есть предок (коммит с которого было начато ветвление), ветвь может сливаться с другими.

На рисунке 10 показан пример ветвления. На базе коммита C2 была создана ветвь iss53. При работе над этой ветвью были созданы 2 коммита C3, C5, параллельно велась работа над ветвью master. После завершения разработки функционала ветви iss53, она была объединена с ветвью master и зафиксирована коммитом C6.



Рисунок 10 - Пример ветвления в git

Чтобы **создать ветвь** в git и затем переключиться на нее (сделать активной), необходимо выполнить следующие команды:

git branch название\_ветки

git checkout название\_ветки

После выполнения этих команды все новые изменения файлов будут направлены в новую ветвь, не затрагивая файлы основного проекта.

При работе с изменения ветви нужно также, как и раньше выполнять команды git add, git commit для фиксации локальных изменений и git push/pull для синхронизации локального репозитория с удаленным (не забудьте вместо main указывать имя новой ветви).

Чтобы переключить разработку на другую ветвь, например, вернуться на основную, нужно опять выполнить git checkout, передав в качестве параметра имя нужной ветви, например git checkout main.

Чтобы объединить 2 ветви в одну, нужно сначала перейти в ту ветвь, которая примет изменения, например main, а затем выполнить команду git merge, как показано ниже:

git checkout main

git merge название\_ветки

При выполнении git pull или git merge может возникнуть ситуация, когда в двух ветках в разных версиях одного файла находится разное содержимое. В этом случае выполняется попытка автослияния (auto-merging), если она окажется неуспешной, то будет необходимо разрешить этот конфликт вручную. В этих случаях git вставляет в место спорного контента следующие инструкции:

<<<<HEAD

один контент

<<< commit1111

другой контент

<<<end

Вам придётся решить, что делать с такими изменениями: что убрать, а что оставить (как правило, конфликты решаются с помощью специальных утилит). После того как конфликт разрешен, необходимо выполнить git commit для фиксации своего варианта решения конфликта.

В случае, если в двух ветках разные файлы (не противоречащие друг другу), то merge просто размещает их вместе, как будто вы скопировали файлы из одной ветки в другую.

# Основные команды

Далее приведен список основных команд, которые могут потребоваться при выполнении данной лабораторной работы:

* git init - создание нового репозитория;
* git clone [url] – клонирование существующего репозитория;
* git add – добавление файлов в проект/для индексирования (в staging область);
* git commit – m ”комментарий” – фиксация изменений, занесений их в репозиторий (добавление параметра –a позволяет провести индексацию);
* git remote – работа с удаленными репозиториями;
* git fetch [remote-name] – извлечение изменений с удаленного репозитория без слияния;
* git merge – слияние извлеченных изменений с текущей ветвью;
* git pull – извлечение изменений + слияние с текущим проектом;
* git checkout – переключение ветви/загрузка файлов в рабочую область;
* git branch – работа с ветвями.

# Задание

1. Повторите действия, описанные в теоретической части лабораторной работы, для этого:
   1. Создайте папку, в которой разместите текстовый файл index.txt, содержащий ваши ФИО. На основе содержимого этой папки создайте репозиторий, после чего выгрузите его на github.
   2. Измените содержимое текстового файла, добавив в него название группы и подгруппы. Зафиксируйте изменения, выгрузите их на github.
   3. Создайте новую ветвь с названием test, переключитесь на нее с помощью команды checkout. Добавьте новый текстовый файл произвольного содержания, а также добавьте в файле index.txt текущие дату и время.
   4. Переключитесь на основную ветвь (main или master), убедитесь, что в рабочей директории проекта остался только файл index.txt и в нем отсутствуют изменения, сделанные на шаге 1.c текущего задания.
2. Необходимо разработать приложения, проверяющее год на високосность (год вводится с клавиатуры) на любом известном вам языке программирования с использованием системы контроля версий git. Для этого:
   1. Создайте пустой проект и локальный репозиторий (будем называть его Repo1). Чтобы избежать отслеживания ненужных файлов и папок, добавьте стандартный файл .gitignore в корневую папку репозитория. Разработайте реализацию проверки на високосность через деления введенного числа на 4 – если делится без остатка, то будем считать год вискосным, иначе – нет.
   2. Зафиксируйте изменения и выгрузите их в репозиторий на github.
   3. Клонируйте репозиторий в другую рабочую директорию (желательно на другом компьютере), будем называть новую копию репозитория Repo2.
   4. Продолжая работать в Repo2 измените код расчета високосности. Правильная процедура определяется следующим описанием: високосным считается год, делящийся на 4, за исключением тех годов, которые делятся на 100 и не делятся на 400 (например, годы 300, 1300 и 1900 не являются високосными, а 1200 и 2000 — являются). Исправьте алгоритм. Зафиксируйте и выгрузите эти изменения.
   5. Оставаясь в Repo2 создайте новую ветвь “Interface”, в которой измените внешний вид вашего приложения (если это консольное приложение, то измените тексты выводимых сообщений). Зафиксируйте изменения и выгрузите их.
   6. Вернитесь в “Repo1”. Очевидно, его состояние в данный момент не актуально. Синхронизируйте его состояние с помощью команды git pull. Запустите новую версию проекта.
   7. Находясь в Repo1, переключитесь на ветвь Interface, запустите проект, продемонстрируйте изменения интерфейса.

# Дополнительное задание

1. Скомпилируйте проект. Ранее вы отключили отслеживание изменений исполняемых файлов, поэтому файлы с расширение .exe по умолчанию не попадут в репозиторий. Однако, допустим, заказчик проекта в данных момент не имеет установленных средств разработки, и хочет иметь возможность проверять проект на разных стадиях его выполнения, запуская готовую программу. Для этого скопируйте полученный исполняемый файл в корневую папку репозитория и добавьте его в репозиторий с помощью команды «git add имя\_файла». Зафиксируйте изменения, выполните синхронизацию с удаленным репозиторием.
2. Настройте репозиторий таким образом, чтобы в процессе работы с ним не отслеживались (игнорировались) изменения содержимого папки BuildLogs в папке проекта, а также все файлы с расширением .log Для этого необходимо ознакомиться с содержимым файла .gitignore и модифицировать его соответствующим образом, после чего зафиксировать изменения.

После изменения настроек, проверьте их работоспособностью. Для этого создайте папку BuildLogs, поместите в нее несколько произвольных файлов, а также файлы test.log и test.txt в папку проекта. Затем вызовите команду «git add . » и сразу после нее «git status». Если все сделано верно, то git «увидит» изменения только файла test.txt.

# Контрольные вопросы

1. Что такое системы контроля версий (СКВ)? Какие СКВ вы знаете?
2. Что такое репозиторий?
3. Каково предназначение файлов .gitignore, .gitattributes?

# Список литературы

Git для начинающих. Часть 2. Установка Git [В Интернете] // Development Practice. - 18 2 2018 г.. - http://devpractice.ru/git-for-beginners-part-2-install-git/.

**Scott Chacon Ben Straub** Pro Git book [В Интернете]. - Apress. - https://git-scm.com/book/ru/v2.