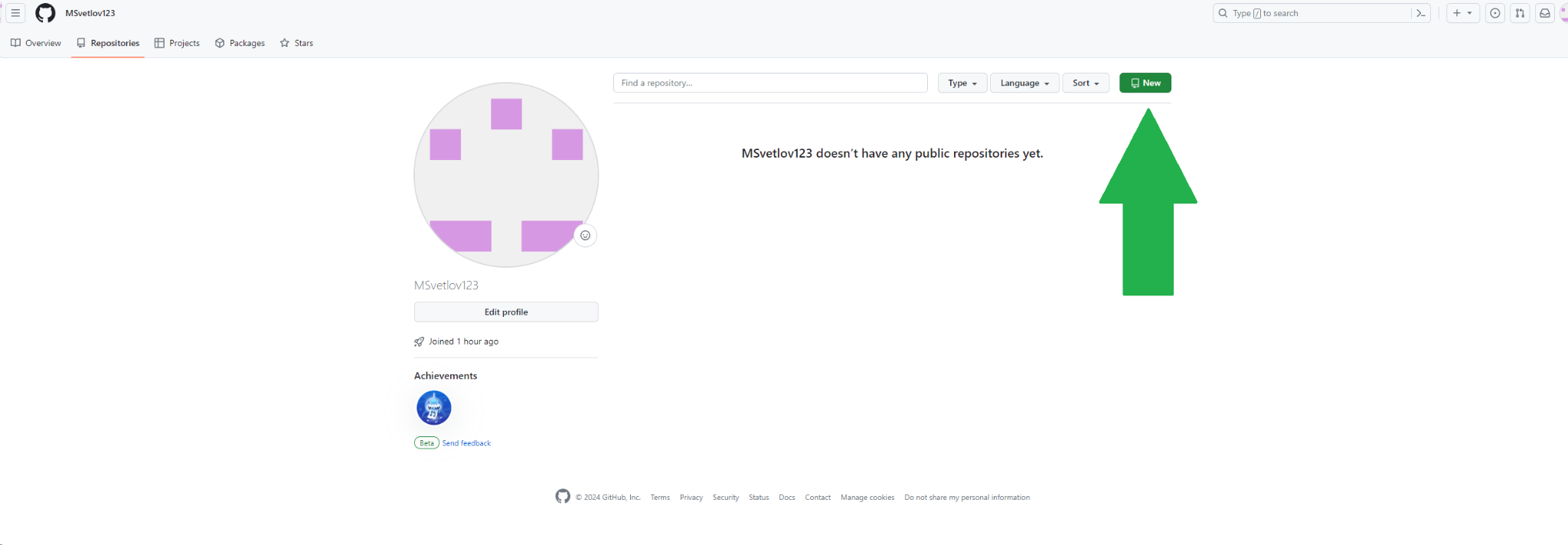
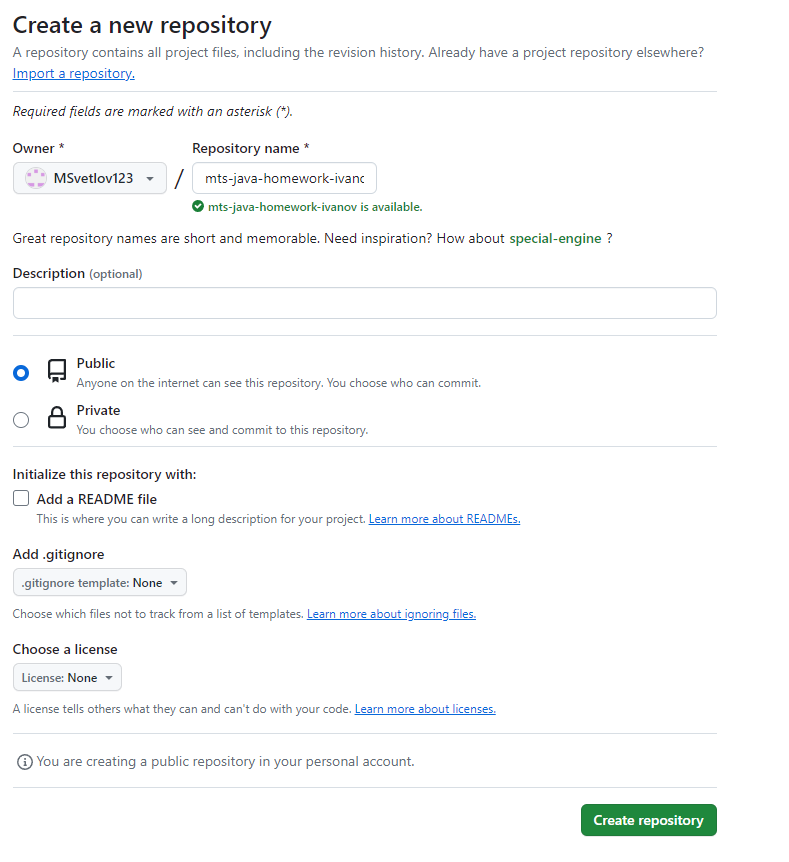
# Задание к занятию по теме “Основы git”

## Подготовка к этому уроку и последующим:

1. Установка Git
   1. MacOS: Рекомендуется установить менеджер пакетов [Homebrew](https://brew.sh/). После этого можно выполнить команду brew install git. Если это по какой-то причине не представляется возможным, то установщик можно найти [тут](https://sourceforge.net/projects/git-osx-installer/). Данный способ не является рекомендованным, последняя доступная посредством установщика версия 2.33 была выпущена в 2021 году.
   2. Windows: Установщик можно найти тут <https://git-scm.com/download/win>. [Ссылка на версию 2.43](https://github.com/git-for-windows/git/releases/download/v2.43.0.windows.1/Git-2.43.0-64-bit.exe).
   3. Для рабочей VDI: Можно заказать через центр установки ПО.
2. Убедитесь, что git был корректно установлен. Для этого в терминале можно выполнить команду git -v, после чего при правильной установке вы сможете увидеть установленную версию git.
3. Создать аккаунт на [github.com](https://github.com/).
   1. Если по какой-то причине github недоступен с рабочего устройства, а личное не подходит для работы, то можно воспользоваться рабочим [gitlab](https://gitlab.services.mts.ru)(доступ возможен только из рабочей сети). Если нет возможности использовать ни личный github, ни банковский gitlab, то прошу сообщить об этом в чате заранее.
4. Установка IDE: Рекомендуется использовать [IntelliJ IDEA](https://www.jetbrains.com/ru-ru/idea/download/). Для целей этого обучения вполне подойдет бесплатная версия Community.
5. Установить JDK. Основная используемая версия JDK 11
6. Создать личный репозиторий в github
   1. Создайте личный репозиторий  
      
   2. Задайте ему имя в формате mts-homework-{фамилия}. Убедитесь, что репозиторий публичный.  
      
   3. Отправьте ссылку на репозиторий преподавателю

## Теория

Git – это распределенная система контроля версий, которая позволяет. Это означает, что каждый пользователь хранит у себя локально полную историю всех изменений.  
 

Git хранит историю изменений в виде снапшотов, то есть каждая версия содержит не изменения произошедшие с прошлой версии, а новый файл целиком. Если файл не был изменен, то хранится ссылка на предыдущую версию этого файла



Основные понятия:

* repository: Репозиторий в Git - это место, где хранятся файлы вашего проекта, включая все его версии и историю изменений. Он представляет собой центральное хранилище, в котором фиксируются все изменения, сделанные вами и предоставляет механизм для управления версиями вашего кода. Репозиторий в Git может быть локальным (на вашем компьютере), удаленным (на сервере) или обоими одновременно. Он позволяет пользователям отслеживать изменения в коде, вносить исправления, создавать новые функции и сотрудничать с другими участниками проекта. Каждый коммит в репозитории сохраняет состояние всех файлов проекта на определенный момент времени, что обеспечивает точный контроль версий и возможность отката к предыдущим состояниям кода в случае необходимости.
* index: В Git индекс, также известный как стейджинг-область, это промежуточное хранилище, где вы подготавливаете изменения перед их фиксацией в репозитории. При работе с Git вы вносите изменения в файлы вашего проекта, а затем добавляете эти изменения в индекс, чтобы подготовить их к коммиту. Индекс позволяет контролировать, какие изменения будут включены в следующий коммит, и обеспечивает возможность составления более логических коммитов, разделяя изменения на логические части. После добавления изменений в индекс вы можете зафиксировать их с помощью команды git commit, сохраняя текущее состояние файлов в вашем репозитории. Использование индекса в Git обеспечивает более гибкое и структурированное управление версиями вашего кода.

В следующий коммит попадут только staged файлы.

* commit: Коммит в Git представляет собой фиксацию изменений в репозитории. Это действие сохраняет текущее состояние всех файлов, которые вы добавили в индекс, в истории репозитория. Каждый коммит обычно сопровождается сообщением, которое кратко описывает внесенные изменения, что помогает другим разработчикам и вам самим понять, что именно было изменено в этом коммите. Коммиты играют ключевую роль в управлении версиями кода, позволяя вам возвращаться к предыдущим состояниям проекта, просматривать историю изменений и проводить анализ различий между версиями. Кроме того, коммиты могут быть объединены, перемещены или удалены, что делает их мощным инструментом для управления разработкой программного обеспечения.
* branch: В Git ветвь (branch) представляет собой параллельную линию разработки, которая позволяет создавать и работать над изменениями в коде независимо от остальных.. Создание ветвей позволяет разработчикам проводить эксперименты, добавлять новые функции или вносить исправления, не затрагивая основную ветвь. После завершения работы ветви можно слить (merge) обратно в основную ветвь для интеграции изменений. Этот подход позволяет проводить параллельную разработку, управлять версиями кода и упрощает сотрудничество нескольких разработчиков над проектом, минимизируя конфликты и обеспечивая безопасность основной кодовой базы.
* merge: Слияние веток в Git — это процесс объединения изменений из одной ветки в другую. Обычно это делается для интеграции новых функций или исправлений ошибок, которые были разработаны в отдельных ветках, обратно в основную ветку (обычно это мастер или develop ветка).  
  Основные шаги для слияния веток в Git:
  + 1. **Переключение на целевую ветку**: Сначала необходимо переключиться на ту ветку, в которую вы хотите внести изменения. Допустим, это мастер или develop ветка git checkout master
    2. **Слияние ветки**: Затем вы можете выполнить команду слияния, указав имя ветки, из которой вы хотите взять изменения. Например, если вы хотите слить ветку feature-branch в текущую ветку:  
       sql git merge feature-branch
    3. **Разрешение конфликтов (по необходимости)**: Если Git обнаруживает конфликты при слиянии изменений, он помечает их в файлах. Вам нужно будет вручную разрешить эти конфликты, выбрав нужные изменения. После решения конфликтов вы должны добавить измененные файлы с помощью git add и выполнить git commit для завершения слияния.
    4. **Завершение слияния**: После разрешения всех конфликтов и сохранения изменений, завершите слияние, создав новый коммит.  
       sql git commit
    5. **Проверка результатов**: После завершения слияния хорошей практикой является тестирование вашего кода, чтобы убедиться, что все работает правильно.
* Merge Request (в GitLab) или Pull Request (в GitHub и Bitbucket) - это механизм, который позволяет разработчикам предложить изменения из одной ветки в другую в репозитории Git. Это часто используется в ситуациях, когда у разработчика есть новый функционал или исправление ошибок, которые он хочет внести в основную ветку проекта.

1. **Создание новой ветки**: Разработчик создает новую ветку на основе целевой ветки (чаще всего это ветка master или main) для работы над определенной задачей или функционалом.
2. **Работа над изменениями**: Разработчик вносит необходимые изменения в код в своей ветке, тестирует их и убеждается, что все работает корректно.
3. **Открытие Merge Request/Pull Request**: Когда разработчик завершает свои изменения и готов предложить их для слияния, он открывает Merge Request (в GitLab) или Pull Request (в GitHub и Bitbucket). В этом запросе он указывает, какие изменения он предлагает внести и в какую ветку их нужно слить.
4. **Обсуждение и рецензирование**: Другие участники команды могут просматривать изменения, оставлять комментарии, задавать вопросы и обсуждать любые возможные изменения. Также они могут выполнить код-ревью, чтобы убедиться в качестве изменений.
5. **Принятие и слияние**: После того как изменения прошли все обсуждения и рецензии и все проблемы были решены, администратор проекта или владелец репозитория принимает Merge Request/Pull Request и производит слияние изменений в основную ветку проекта.
6. **Удаление ветки (по желанию)**: После успешного слияния ветку, которая была использована для внесения изменений, можно удалить, если она больше не нужна.

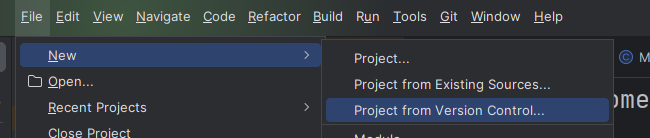
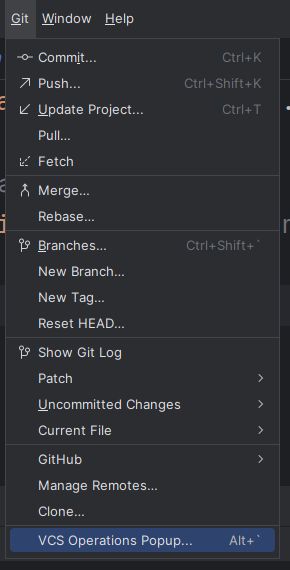
## Практика

### Работа через консоль

1. Конфигурация git
2. Скачивание удаленного репозитория:
   1. git clone <адрес репозитория>
3. Создать новый локальный репозиторий:
   1. git init
4. Привязать код к существующему репозиторию:
   1. git remote add <имя> <адрес репозитория>
5. Работа с index:
   1. Просмотр состояний файлов: git status
   2. Добавление файлов в index: git add
   3. Удалить файлы из индекса: git restore --staged <файл>
   4. Просмотр отличий рабочей директории и последнего коммита: git diff --cached
6. Создать commit
   1. git commit -m <сообщение коммита>
7. Отправить локальные коммиты в удаленный репозиторий Push
   1. git push <репозиторий> <ветка>
8. Pull
   1. git pull <репозиторий> <ветка>
9. Работа с branch
   1. Создать новую локальную ветку из текущей git branch -b <наименование новой ветки>
   2. Удаление ветки git branch -d <наименование ветки>
   3. Переименовать ветку git branch -m <наименование старой ветки> <наименование новой ветки>

### 

### Работа через Intellij Idea

1. Скачивание удаленного репозитория:
   1. 
2. Основное меню и хоткеи
   1. 

### 

### Небольшое задание

1. Клонируйте созданный ранее репозиторий на свой локальный компьютер
2. Создайте в нем Java проект, использующий gradle или maven
3. Закоммитьте и запушьте созданный проект в удаленный репозиторий
4. Создайте новую ветку “feature-branch”
5. Внесите любые изменения
6. Слейте новую ветку с основной вручную, следуя инструкциям приведенным ранее
7. Создайте новую ветку “bugfix-branch”
8. Внесите любые изменения
9. Merge Request (в GitLab) или Pull Request (в GitHub), но не вливайте изменения