

Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и  
радиоэлектроники»

Кафедра инженерной психологии и эргономики

**Пользовательские интерфейсы информационных систем**

Отчет по практическим занятиям на тему  
«Образовательный курс GitHowTo»

Выполнил:  
Студент гр. 210901  
Князева Е. С.

Проверил:  
Давыдович К. И.

Минск 2024

**Цель:** сформировать понимание и специфику работы с инструментом контроля версий «Git» и научиться пользоваться его основным функционалом.

Курс располагается по ссылке <https://githowto.com/ru>.

## **Введение**

В ходе прохождения курса на сайте githowto.com я получила глубокое понимание системы контроля версий Git. Этот курс стал отличной возможностью для освоения ключевых команд и принципов работы с Git, что является необходимым условием для эффективного управления проектами и отслеживания изменений в коде. В рамках обучения были охвачены как базовые концепции, так и продвинутые техники работы с Git, что позволяет максимально использовать его функциональные возможности.

## **Основные разделы и темы курса**

### **Введение в Git. Основные понятия и термины**

В этом разделе курс начинался с основ. Были рассмотрены такие базовые понятия, как репозиторий, коммит, ветка и слияние. Эти термины являются фундаментальными для работы с Git и их понимание необходимо для дальнейшего освоения системы.

### **Установка и настройка Git**

Далее был рассмотрен процесс установки Git на различные операционные системы, включая Windows, macOS и Linux. Я узнала, как правильно настроить Git, включая конфигурацию имени пользователя и электронной почты, что важно для ведения истории коммитов.

### **Пример команды для Git Bash:**

Установка имени пользователя.

```
git config --global user.name "Elizaveta"
```

Установка электронной почты.

```
git config --global user.email liza2005@gmail.com
```

Мы будем использовать main в качестве имени ветки по умолчанию. Чтобы установить его, выполните следующую команду:

```
git config --global init.defaultBranch main
```

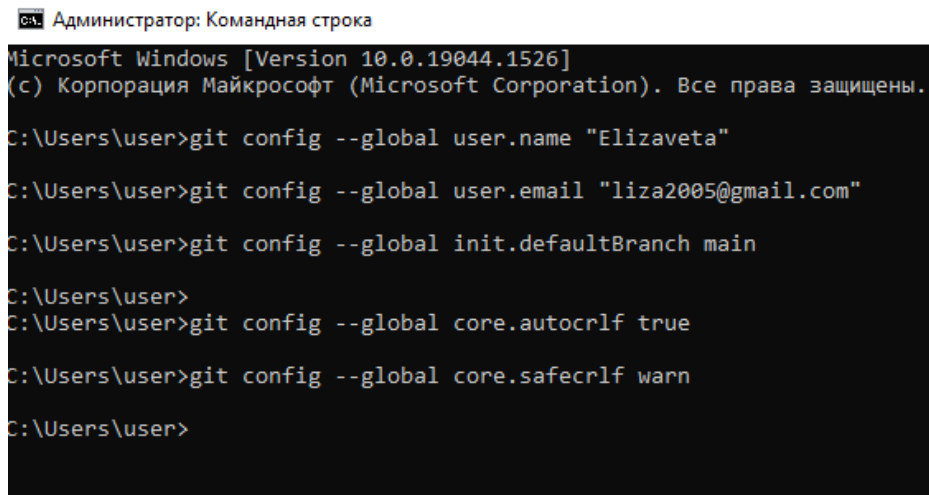
Также нужно выполнить данные команды для корректной обработки окончаний строк (для Windows):

```
git config --global core.autocrlf true
```

```
git config --global core.safecrlf warn
```

Проверка конфигурации.

git config –list



```
Администратор: Командная строка
Microsoft Windows [Version 10.0.19044.1526]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\user>git config --global user.name "Elizaveta"

C:\Users\user>git config --global user.email "liza2005@gmail.com"

C:\Users\user>git config --global init.defaultBranch main

C:\Users\user>
C:\Users\user>git config --global core.autocrlf true

C:\Users\user>git config --global core.safecrlf warn

C:\Users\user>
```

Рисунок 1 – начало работы с Git

## Инициализация и клонирование репозитория

### Команда git init

На этом этапе я изучила, как создать новый локальный репозиторий с помощью команды git init. Эта команда инициализирует новый репозиторий в текущей директории, создавая скрытую папку .git, где будет храниться вся история проекта.

### Пример команды для Git Bash:

Создание новой директории для проекта.

```
mkdir my_project
```

```
cd my_project
```

Инициализация нового репозитория.

```
git init
```

### Команда git clone

Также была рассмотрена команда git clone, которая позволяет клонировать удаленный репозиторий на локальную машину. Это полезно для работы в команде, когда требуется скопировать проект с удаленного сервера.

### Пример команды для Git Bash:

Клонирование удаленного репозитория.

```
git clone https://github.com/username/repository.git
```

## Основные команды Git

### Команда `git add`

Команда `git add` используется для добавления изменений в индекс (staging area). Этот процесс позволяет подготовить изменения для последующего коммита.

#### Пример команды для Git Bash:

Добавление всех изменений в индекс.

```
git add .
```

Добавление конкретного файла в индекс.

```
git add filename.txt
```

### Команда `git commit`

После добавления изменений в индекс, их можно зафиксировать в истории проекта с помощью команды `git commit`. Курс рассматривал, как правильно писать сообщения коммитов, чтобы они были информативными и полезными для команды.

#### Пример команды для Git Bash:

Создание коммита с сообщением.

```
git commit -m "Описание изменений"
```

### Команда `git status`

Для проверки текущего состояния репозитория используется команда `git status`. Она показывает, какие файлы были изменены, добавлены в индекс или удалены.

#### Пример команды для Git Bash:

Проверка статуса репозитория.

```
git status
```

### Команда `git log`

Команда `git log` позволяет просматривать историю коммитов в репозитории. Это важный инструмент для отслеживания изменений и анализа прошлого состояния проекта.

#### Пример команды для Git Bash:

Просмотр истории коммитов.

```
git log
```

## **Работа с ветками**

### **Команда `git branch`**

Ветвление в Git является мощным инструментом для работы над новыми функциями или исправлениями без риска нарушить основную ветку проекта. Команда `git branch` позволяет создавать новые ветки и управлять ими.

#### **Пример команды для Git Bash:**

Создание новой ветки.

```
git branch new_feature
```

Просмотр всех веток.

```
git branch
```

### **Команда `git checkout`**

Для переключения между ветками используется команда `git checkout`. Это позволяет легко переключаться между различными версиями проекта.

#### **Пример команды для Git Bash:**

Переключение на новую ветку.

```
git checkout new_feature
```

### **Команда `git merge`**

Для объединения изменений из одной ветки в другую используется команда `git merge`. Этот процесс иногда сопровождается конфликтами, которые необходимо разрешать вручную.

#### **Пример команды для Git Bash:**

Переключение на основную ветку.

```
git checkout master
```

Слияние изменений из ветки `new_feature`.

```
git merge new_feature
```

## **Удаленные репозитории**

### **Команда `git remote`**

Курс также рассматривал работу с удаленными репозиториями. Команда `git remote` позволяет управлять подключениями к удаленным серверам.

#### **Пример команды для Git Bash:**

Добавление удаленного репозитория.

```
git remote add origin https://github.com/kukidon/repository.git
```

Просмотр удаленных репозиторияев.

```
git remote -v
```

### **Команда git fetch и git pull**

Команда git fetch загружает изменения из удаленного репозитория без автоматического объединения их с текущей веткой. Команда git pull сочетает в себе fetch и merge, автоматически объединяя загруженные изменения с текущей веткой.

### **Пример команды для Git Bash:**

Загрузка изменений из удаленного репозитория.

```
git fetch origin
```

Загрузка и слияние изменений из удаленного репозитория.

```
git pull origin master
```

### **Команда git push**

Команда git push позволяет отправлять локальные изменения в удаленный репозиторий. Это важный этап для совместной работы над проектом, так как он синхронизирует изменения между участниками команды.

### **Пример команды для Git Bash:**

Отправка изменений в удаленный репозиторий.

```
git push origin master
```

### **Откат изменений**

### **Команда git reset**

Для отмены изменений и возврата к предыдущим состояниям репозитория используется команда git reset. Эта команда может быть очень мощной, но требует осторожности, так как изменения могут быть необратимыми.

### **Пример команды для Git Bash:**

Отмена последних изменений, но сохранение в рабочей директории.

```
git reset --soft HEAD~1
```

Отмена последних изменений без сохранения в рабочей директории.

```
git reset --hard HEAD~1
```

### **Команда git revert**

Команда git revert позволяет создать новый коммит, который отменяет изменения из предыдущего коммита. Это безопасный способ отката, так как история изменений сохраняется.

### **Пример команды для Git Bash:**

Создание коммита для отмены предыдущего коммита.

```
git revert HEAD
```

### **Практические задания и упражнения**

Курс включал множество практических заданий, которые позволяли закрепить теоретические знания на практике. Вот некоторые из них:

### **Инициализация нового репозитория**

В этом задании требовалось создать новый локальный репозиторий, добавив в него несколько файлов и закоммитив их.

### **Пример команды для Git Bash:**

Создание новой директории для проекта.

```
mkdir my_new_project
```

```
cd my_new_project
```

Инициализация нового репозитория.

```
git init
```

Создание файла и добавление его в репозиторий.

```
echo "Hello, Git!" > README.md
```

```
git add README.md
```

```
git commit -m "Добавил README файл"
```

### **Клонирование удаленного репозитория**

Для этого задания необходимо было клонировать существующий удаленный репозиторий и внести в него изменения, после чего отправить их обратно на сервер.

### **Пример команды для Git Bash:**

Клонирование удаленного репозитория.

```
git clone https://github.com/username/existing_repo.git
```

```
cd existing_repo
```

Внесение изменений и коммит.

```
echo "Новые изменения" >> README.md
```

```
git add README.md
git commit -m "Внес новые изменения в README"
Отправка изменений в удаленный репозиторий.
git push origin master
```

### **Работа с ветками и слияниями**

Задание включало создание новой ветки, внесение в нее изменений и последующее слияние с основной веткой, разрешая возникающие конфликты.

#### **Пример команды для Git Bash:**

Создание новой ветки.

```
git branch new_feature
git checkout new_feature
```

Внесение изменений в новой ветке.

```
echo "Новая функция" > feature.txt
git add feature.txt
git commit -m "Добавила новую функцию"
Переключение на основную ветку.
git checkout master
```

Слияние изменений из ветки new\_feature.

```
git merge new_feature
```

Разрешение конфликтов (если они возникнут).

Открытие конфликтного файла и редактирование его в соответствии с требованиями.

```
nano conflict_file.txt
```

После разрешения конфликтов добавление файла в индекс.

```
git add conflict_file.txt
```

Завершение слияния.

```
git commit -m "Разрешены конфликты и завершено слияние ветки new_feature"
```

### **Отправка изменений в удаленный репозиторий**

В этом упражнении необходимо было создать удаленный репозиторий, подключить его к локальному проекту и отправить в него изменения.

#### **Пример команды для Git Bash:**

Подключение удаленного репозитория.

```
git remote add origin https://github.com/username/new_repository.git
```

Отправка изменений в удаленный репозиторий.



```
git push -u origin master
```

### **Откат изменений**

В этом задании требовалось использовать команды `git reset` и `git revert` для отмены предыдущих коммитов и возврата к стабильной версии проекта.

### **Пример команды для Git Bash:**

Отмена последнего коммита, но сохранение изменений в рабочей директории.

```
git reset --soft HEAD~1
```

Полный откат последнего коммита без сохранения изменений.

```
git reset --hard HEAD~1
```

Создание коммита для отмены предыдущего коммита.

```
git revert HEAD
```

### **Дополнительные темы и советы**

Помимо основ, курс также рассматривал более продвинутые темы и давал полезные советы:

### **Работа с .gitignore**

Файл `.gitignore` позволяет исключить из отслеживания ненужные файлы, такие как временные файлы, конфигурационные файлы сред разработки и т.д. Это помогает поддерживать чистоту репозитория и избегать случайного коммита ненужных данных.

### **Пример команды для Git Bash:**

Создание файла `.gitignore`.

```
echo "node_modules/" > .gitignore
```

```
echo "*.log" >> .gitignore
```

Добавление `.gitignore` в репозиторий.

```
git add .gitignore
```

```
git commit -m "Добавила .gitignore для исключения временных файлов и директорий"
```

### **Рефакторинг истории с помощью git rebase**

Команда `git rebase` позволяет переписывать историю коммитов, что может быть полезно для упрощения дерева коммитов и объединения нескольких коммитов в один.

### **Пример команды для Git Bash:**

Перебазирование текущей ветки на master.

```
git checkout new_feature
```

`git rebase master`

Присоединение нескольких коммитов в один.

`git rebase -i HEAD~3`

В интерактивном режиме выбираем коммиты для слияния.

### **Использование тегов (git tag)**

Теги используются для маркировки определенных точек в истории проекта, таких как релизы версий. Это облегчает навигацию по репозиторию и управление выпусками.

#### **Пример команды для Git Bash:**

Создание аннотированного тега.

`git tag -a v1.0 -m "Релиз версии 1.0"`

Просмотр списка тегов

`git tag`

Отправка тега в удаленный репозиторий.

`git push origin v1.0`

### **Стратегии ветвления**

Курс также обсуждал различные стратегии ветвления, такие как Git Flow и Trunk Based Development, которые помогают организовать процесс разработки и интеграции изменений.

#### **Пример использования Git Flow:**

Установка Git Flow.

`brew install git-flow-avh`

Инициализация Git Flow в репозитории.

`git flow init`

Начало новой фичи.

`git flow feature start new_feature`

Завершение фичи и слияние ее с develop.

`git flow feature finish new_feature`

Выпуск релиза

`git flow release start v1.0`

`git flow release finish v1.0`

### **Заключение**

Курс на сайте githowto.com стал для меня важным шагом в освоении системы контроля версий Git, которая является неотъемлемой частью

современного программирования и разработки программного обеспечения. Git помогает разработчикам эффективно управлять проектами.

В ходе курса я познакомилась с основными концепциями Git, такими как репозитории, коммиты, ветвление и слияние. Эти понятия являются основополагающими для понимания работы системы и позволяют организовать процесс разработки на высоком уровне. Практические задания курса помогли мне закрепить полученные знания и применять их на практике, что особенно важно для формирования навыков работы с Git.

Курс также охватывает продвинутые возможности Git, такие как управление конфликтами при слиянии, работа с удалёнными репозиториями и использование различных стратегий ветвления. Эти темы раскрывают весь потенциал Git и позволяют мне более уверенно подходить к управлению проектами. Я научилась использовать команды для эффективного отслеживания изменений, создания резервных копий и совместной работы с другими разработчиками.

Кроме того, курс предоставляет полезные советы и рекомендации по лучшим практикам использования Git, что помогает минимизировать ошибки и повысить продуктивность. В результате я не только освоила теоретические аспекты, но и получила практический опыт, который смогу применять в будущих проектах.

Таким образом, курс на [githowto.com](https://githowto.com) стал для меня ценным ресурсом, который значительно расширил мои знания о Git и подготовил меня к успешной работе в команде, что является ключевым аспектом в современной разработке программного обеспечения.