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: | Проверил: |
| cтудент гр. 210901  Романов К.В. | Давыдович К. И. |
|  |  |

Минск 2024

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

**Отчет по прохождению курса по обучению в Git**

**Введение**

В ходе прохождения курса на платформе githowto.com я получил всестороннее представление о системе контроля версий Git, который является неотъемлемым инструментом для современного программирования и разработки программных проектов. Этот курс предоставил детальное объяснение как базовых, так и более сложных концепций, что позволило мне не только разобраться в базовых командах, но и освоить продвинутые методики работы с репозиториями и ветками.

**Основные разделы курса**

**Основные разделы курса**

1. **Введение в Git: ключевые понятия и термины**

В самом начале курса были рассмотрены ключевые концепции и термины, которые являются основой для работы с Git. Это такие понятия, как:

**Репозиторий** — место хранения проекта, где Git сохраняет всю историю изменений.

**Коммит** — зафиксированное изменение в репозитории, которое сохраняет состояние файлов.

**Ветка (Branch)** — параллельная линия разработки, которая позволяет работать над новыми функциями или исправлениями, не затрагивая основную (master) ветку.

**Слияние (Merge)** — процесс объединения изменений из одной ветки в другую.

1. **Установка и настройка Git**

Курс подробно объяснил, как правильно установить Git на различных операционных системах (Windows, macOS и Linux) и настроить его для работы. На этом этапе я узнал, как настроить имя пользователя и электронную почту, что важно для отслеживания авторства коммитов. Пример команд для Git Bash:

bash

Копировать код

git config --global user.name "Peter"

git config --global user.email "kukidon820@gmail.com"

git config --list

Эти настройки позволяют Git фиксировать информацию о том, кто сделал коммит, что особенно важно при работе в команде.

1. **Инициализация и клонирование репозитория**

На этом этапе я научился создавать новый локальный репозиторий с помощью команды **git init**. Эта команда инициализирует новый репозиторий в текущей директории, создавая скрытую папку .git, где хранятся все метаданные и история проекта. Пример:

bash

Копировать код

mkdir my\_project

cd my\_project

git init

Также я изучил команду **git clone**, которая позволяет клонировать удаленный репозиторий на локальную машину, что особенно полезно для совместной работы в команде. Пример:

bash

Копировать код

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

1. **Основные команды Git**

Курс подробно объяснил работу с самыми часто используемыми командами Git:

**git add** — добавляет изменения в индекс (staging area), готовя их к коммиту. Пример:

bash

Копировать код

git add .

git add filename.txt

**git commit** — фиксирует изменения в репозитории с сообщением, которое объясняет суть изменений. Пример:

bash

Копировать код

git commit -m "Описание изменений"

**git status** — проверка текущего состояния репозитория. Показаны все измененные файлы, файлы, готовые к коммиту, и те, которые еще не добавлены в индекс. Пример:

bash

Копировать код

git status

**git log** — просмотр истории изменений проекта. Пример:

bash

Копировать код

git log

1. **Работа с ветками**

Ветки являются основой параллельной разработки в Git. Курс обучил меня работе с ветками с помощью таких команд, как **git branch**, **git checkout** и **git merge**:

**git branch** — позволяет создавать новые ветки и просматривать существующие:

bash

Копировать код

git branch new\_feature

git branch

**git checkout** — переключает текущую рабочую ветку:

bash

Копировать код

git checkout new\_feature

**git merge** — слияние изменений из одной ветки в другую. Этот процесс может потребовать разрешения конфликтов, если изменения в сливаемых ветках противоречат друг другу. Пример:

bash

Копировать код

git checkout master

git merge new\_feature

1. **Работа с удаленными репозиториями**

Курс также охватывает работу с удаленными репозиториями, что позволяет синхронизировать изменения между несколькими разработчиками:

**git remote** — для управления удаленными репозиториями (например, добавление нового удаленного репозитория).

bash

Копировать код

git remote add origin https://github.com/username/repository.git

git remote -v

**git fetch** — загружает изменения из удаленного репозитория, но не сливает их с локальной веткой.

bash

Копировать код

git fetch origin

**git pull** — комбинированная команда, которая загружает изменения из удаленного репозитория и сразу сливает их с текущей веткой.

bash

Копировать код

git pull origin master

**git push** — отправка изменений из локального репозитория в удаленный:

bash

Копировать код

git push origin master

1. **Откат изменений**

Бывает необходимо отменить изменения, и для этого курс объяснил команды **git reset** и **git revert**:

**git reset** — отменяет изменения в репозитории. Есть несколько вариантов использования этой команды:

Сохранить изменения в рабочей директории:

bash

Копировать код

git reset --soft HEAD~1

Полностью отменить изменения:

bash

Копировать код

git reset --hard HEAD~1

**git revert** — безопасный способ отката изменений, создавая новый коммит, который отменяет изменения предыдущего коммита:

bash

Копировать код

git revert HEAD

1. **Дополнительные темы и советы**

В процессе обучения я также освоил более сложные аспекты работы с Git:

**Работа с файлом .gitignore** — позволяет исключить из отслеживания временные файлы, логи и другие ненужные данные:

bash

Копировать код

echo "node\_modules/" > .gitignore

echo "\*.log" >> .gitignore

git add .gitignore

git commit -m "Добавлен .gitignore"

**Git rebase** — переписывание истории коммитов, позволяющее сделать ее более линейной и понятной:

bash

Копировать код

git rebase master

git rebase -i HEAD~3

**Теги (git tag)** — используются для маркировки важных моментов в истории проекта, например, для выпуска версий:

bash

Копировать код

git tag -a v1.0 -m "Релиз версии 1.0"

git push origin v1.0

**Стратегии ветвления** — Git Flow и Trunk Based Development — помогают организовать процесс разработки и управления ветками в проекте. Пример с Git Flow:

bash

Копировать код

git flow init

git flow feature start new\_feature

git flow feature finish new\_feature

git flow release start v1.0

**Заключение**

Курс на сайте githowto.com стал важным этапом в освоении Git, который теперь является неотъемлемой частью моего рабочего процесса. Я научился работать с репозиториями, управлять ветками, сливать изменения и синхронизировать проект с удаленными репозиториями. Полученные знания и навыки, такие как правильное использование команд Git, умение разрешать конфликты и управлять историей изменений, значительно улучшат мою способность эффективно работать в команде и поддерживать порядок в проекте.

Этот курс не только расширил мои технические умения, но и значительно улучшил мою способность управлять проектами, следить за их развитием и справляться с различными ситуациями, которые могут возникнуть в процессе работы с репозиториями.