AIT LMS

lms.ait-tr.de

Задачи на урок:

1. Установка git
2. Проверка IntelIJ
3. Создание локальных репозиториев при помощи IntelIJ
4. Создание удаленных репозиториев
5. Подключение remote через интрефейс IntelIJ
6. push, pull, add, commit через интрефейс IntelIJ

Зачем нужна система контроля версий?

Каждому из нас хотя бы раз в жизни при написании программы или редактировании текста было необходимо откатиться назад, например, потому, что вы случайно удалили важный параграф.

Хорошо, если это изменение, которое произошло только что — многие редакторы поддерживают отмену изменений, например, сочетанием клавиш `Ctrl`+`Z`.

Однако трудно обратить действия, если изменения произошли давно, а ошибка была обнаружена спустя какое-то время. В таком случае самое простое решение — раз в день создавать копию документа и помещать в другую папку, помечая ее датой, в которую эти изменения были созданы.

Это самое простое в реализации решение, однако при восстановлении легко ошибиться и перепутать папки. Ситуация еще больше усложняется, если над этим документом или кодом программы работают несколько человек, каждый из которых решает свои собственные задачи.

Система контроля версий предназначена для сохранения истории изменений.

История представляет собой снимки проекта, следующие друг за другом в хронологическом порядке. В любой момент можно откатиться к любому состоянию системы в прошлом.

Таким образом, можно восстановить поврежденные или случайно удаленные файлы, а также выяснить, кто автор внесенных в код изменений.

Другое назначение системы контроля версий — организация командной работы над проектом.

Система контроля версий в команде разработчиков позволяет корректно слить изменения от нескольких участников, не перезаписывая результаты работы друг друга.

При возникновении конфликтной ситуации, когда разработчики правят один и тот же участок кода, система обязательно просигнализирует об этом. Добавить новый код без устранения конфликта не получится.

Настройка `git`

Перед началом работы необходимо сообщить `git`, кто вы и как вас представлять другим участникам распределенной системы контроля версий. Пока вы не подпишетесь, система не даст вам регистрировать снимки проекта, коммиты.

Все ваши изменения должны быть подписаны вашим именем и электронным адресом, чтобы другие участники проекта знали, чьи это правки и как с вами связаться. Сделать это нужно один раз, как правило, сразу после установки git, если вы переустановите систему, процедуру потребуется повторить.

Чтобы задать ваши имя и электронный адрес, следует воспользоваться командой `git config`:

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

Убедиться в том, что настройки успешно установлены, можно, запросив их список при помощи команды `git config --list`.

Она выдает множество настроек, в том числе и только что установленные значения.

```
user.name=John Doe
user.email=johndoe@example.com
```

Файл `.gitignore`

Файл `.gitignore` для работы с проектом в `IntelliJ Idea` должен включать в себя следующие строчки:

```
.idea/
*.iml
```

Работа с GitHub и IntelliJ Idea

Работа с GitHub в нужном нам количестве подробно описана на видеозаписи занятия.

Документация по настройке IntelliJ Idea

Базовые понятия

- **проект** - каталог с файлами исходного кода (кодовая база)
- **репозиторий** - хранилище истории разработки проекта
- **клонирование (репо)** - скачивание репо на компьютер

Базовый порядок работы

1. **Инициализация нового репо**
    1. `git init`
    2. создается “скрытое хранилище” - каталог `.git/`
2. **Сохранение**
    - Индексация файлов (добавить в очередь на сохранение)
      ```
      git add .
      ```
    - Выполнить сохранение
      ```
      git commit -m 'update'
      ```
3. **Привязка репо**
4. **Выгрузка ветки на GitHub**
    1. `git push -u origin main` (`main` или другое название)
    2. `git push` (если ветку уже выгружал)

Файл `README.md`

- использует формат `Markdown`
- описание репо на GitHub
- помещается в корень проекта, как правило

Пример

```
## Test Project
```

Работа с привязкой

Удалить старую привязку

```
git remote rm origin
```

Добавить привязку

```
git remote add origin скопированная_ссылка
```

Просмотр текущей привязки

```
git remote -v
```

Ветвление в Git

Ветка - еще одна версия проекта (изолированный поток разработки)

Стратегии ветвления в Git

1. **Git Flow**
    1. `main/master/stable` - long-term (только для проверенного, оттестированного кода - “**священный грааль**”)
    2. `develop/current` - long-term (для тестирования, текущая разработка)
    3. `login/bugfix1/payments` - short-term
2. **GitHub Flow**
    1. `main/master/stable` - long-term
    2. `login/bugfix1/payments` - short-term

Базовые команды по работе с ветками в Git

- `git branch` просмотр веток
    - `git branch -avv` подробный вывод
    - выйти из просмотра - `q`
- `git branch новая_ветка` создать ветку
  `git branch новая_ветка старая_ветка`
- `git checkout ветка` переключиться на ветку
    - `git checkout -` переключиться на предыдущую ветку
    - **ПЕРЕКЛЮЧАТЬСЯ НЕОБХОДИМО С “ЧИСТЫМ СТАТУСОМ”**
- `git checkout -b новая_ветка` создать и переключиться
  `git checkout -b новая_ветка старая_ветка`
- `git branch -D ветка` удалить ветку (локально)
- `git merge название_ветки` слияние веток

- `git push origin --delete ветка` удалить ветку (дистанционную)
- `git branch -m новое_название` переименовать ветку (локально)

Слияние веток

- перенос (интеграция) изменений из одной ветки в другую
- выполняется командой

    `git merge название_ветки`
- перед слиянием необходимо переключиться в целевую ветку

Пример

```
git checkout -b login
# внести правки
git checkout master
git merge login
git branch -D login
```

Клонирование репо

1. Открыть репо на **GitHub**
2. Решить, куда его скачать
3. Скопировать SSH-ссылку
4. Выполнить команду
    1. `git clone скопированная_ссылка`

Ссылки

Homework

Задание 1

1. Создать локальный репозиторий, создать удаленный репозиторий. Подключить локальный репозиторий к удаленному.

2. Внести изменения в локальном репозитории и отправить эти изменения в удаленный.
3. Прислать ссылку на удаленный репозиторий мне в личном сообщении. (содержание у репозитория может быть любым)

./code/example_01/src/Main.java

```
public class Main {
  public static void main(String[] args) {
    int y = 5;
    Orange orange = new Orange();
    int x = 10;
    System.out.println(x);
    System.out.println(y);
    // git - программа, система контроля версий.
    // Сама по себе ничего не знает про ваши облачные хранилища
    // и может работать даже без доступа в Интернет.

    // Git работает с репозиториями - папками, в которых следит за
историей и версиями файлов.
    // В git могут существовать разные параллельные версии одного
и того же репозитория - ветки.
    // С ними вы познакомитесь позже, в нашем случае вся работа
будет происходить в одной основной
    // ветке - main (в старых версиях - master).

    // Эти репозитории можно хранить в облачном хранилище. Самое
популярное такое хранилище - GitHub.
    // Это надёжнее, и так гораздо легче организовать совместную
работу над проектом.

    // На сайте GitHub создаём ПУСТОЙ репозиторий, и подключаем
его как remote к нашему существующему,
    // созданному при помощи IntelliJ Idea.
    // Любая другая последовательность действий может привести к
проблеме, которую будет сложно
    // решить.
  }
}
```

./code/example_01/src/Apple.java

```
public class Apple {
}
```

./code/example_01/src/Orange.java

```
public class Orange {
}
```