Lesson 27

05.07.2021

```java
public class Test1 {
    public static void main(String[] args) {
        int sum = 0;
        for (int i = 0, j = 0; i < 6 & j < 5; ++i, j = i + 1) {
            sum = +i;
            System.out.println(sum);
        }
    }
}
```

```java
public class Test2 {
    public static void main(String[] args) {
        int arr[] = {11, 22, 33};
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i] + " ");

        int arrr[] = new int[3];
        arrr[] = {11, 22, 33};
        for (int i = 0; i < arrr.length; i++)
            System.out.println(arrr[i] + " ");
    }
}
```

```java
public class Test3 {
    public static void main(String[] args) {
        int[][] arr1 = new int[2][3];
        int[][] arr2 = new int[2][];
        int[][] arr3 = new int[][];
        int[][] arr4 = new int[][3];
    }
}
```

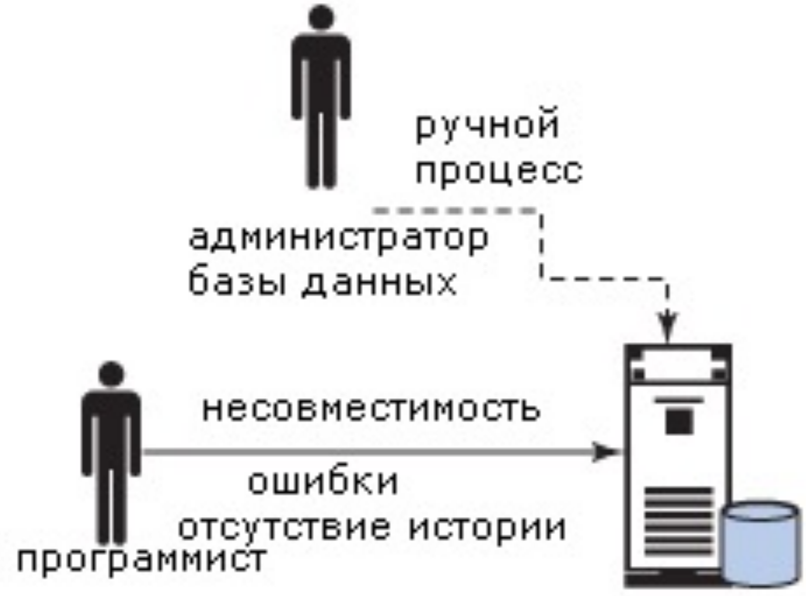
```
public class Test4 {
    public static void main(String[] args) {
        for (int i = 0; i < 1; System.out.println("Java")) {
            System.out.println("Scala");
        }
    }
}
```

Миграция баз данных

Разработка программного обеспечения приводит к следующим проблемам:

- ручное внесение изменений в БД;
- разные версии БД у разных участников команды разработчиков;
- непоследовательные подходы к внесению изменений (в базу данных или данные);
- неэффективные механизмы ручного управления изменениями при переходах между версиями баз данных.

Ручное внесение изменений в таблицу базы данных



Автоматизация миграции базы данных

Управление изменениями в базе данных с помощью LiquiBase (www.liquibase.org)

Liquibase — это независимая от базы данных библиотека для отслеживания, управления и применения изменений схем базы данных. Для того, чтобы внести изменения в БД, создается файл миграции (*changeset*), который подключается в главный файл (*changeLog*), который контролирует версии и управляет всеми изменениями. В качестве описания структуры и изменений базы данных используется XML, YAML, JSON и SQL форматы.

Основные функциональные возможности

- Обновление базы данных до текущей версии
- Откат последних изменений в базе данных / на определенную дату / время / тега
- SQL для обновлений баз данных и откатов можно сохранить для ручного просмотра
- «Контексты» для включения / исключения наборов изменений для выполнения
- Отчет о различиях базы данных
- Генерация изменений в базе данных
- Возможность создания журнала изменений для создания существующей базы данных
- Создание документации по изменениям базы данных
- Проверка СУБД, проверка пользователя и предварительные условия проверки SQL
- Возможность разбивать журнал изменений на несколько файлов для упрощения управления
- Исполняется через командную строку, Apache Ant, Apache Maven, servlet container или Spring Framewor

Чтобы начать работать с LiquiBase, нужно выполнить четыре шага:

- Создать файл с журналом изменений в базе данных (**change log**).
- Определить набор изменений (change set) внутри этого файла.
- Применить набор изменений к базе данных через командную строку или сценарий сборки.
- Проверить изменения в базе данных.

Подготовленное состояние файла

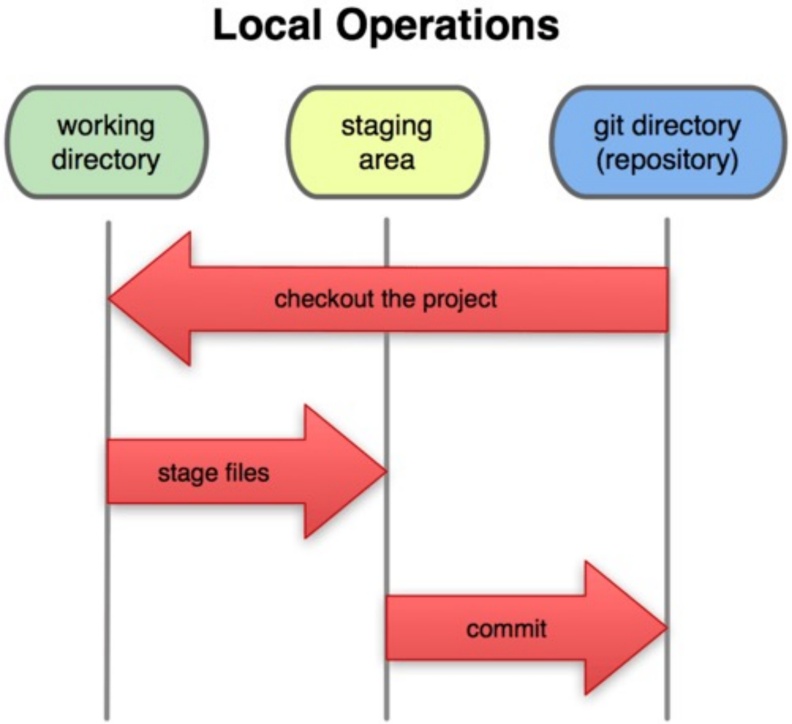
Это изменённые файлы, отмеченные для включения в следующий коммит.

Измененное состояние файла

К изменённым относятся файлы, которые поменялись, но ещё не были зафиксированы.

Зафиксированное состояние файла

Значит, что файл уже сохранён в вашей локальной базе.



Файл ~/.gitconfig

Хранит настройки конкретного пользователя. Этот файл используется при указании параметра --global.

В системах семейства Windows Git ищет файл `.gitconfig` в каталоге `$HOME` (`C:\$USER` или `C:\Users\$USER`).

Первое, что вам следует сделать после установки Git'a, — указать ваше имя и адрес электронной почты. Это важно, потому что каждый коммит в Git'e содержит эту информацию, и она включена в коммиты, передаваемые вами, и не может быть далее изменена:

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

Если вы хотите проверить используемые настройки, можете использовать команду git config - -list

Создание репозитория в существующем каталоге

Если вы собираетесь начать использовать Git для существующего проекта, то вам необходимо перейти в проектный каталог и в командной строке ввести git init.

Эта команда создаёт в текущем каталоге новый подкаталог с именем ***.git*** содержащий все необходимые файлы репозитория — основу Git-репозитория. На этом этапе ваш проект ещё не находится под версионным контролем.

Файл .gitignor

создаем txt фаил = имя .gitignore
Команды игнора для файлов папок
(Имя папки)
txt files
docs/*.txt// Все файлы .txt в этой папке, будут проигнорированы
показать не отслеживаемые файлы - $ git status --untracked-files=all

Добавления всех файлов проекта в Git

Используем команду “git add .” и добаляем все файлы под версионный контроль
Варианты команды:
git *.расширение – все файлы одного расширения
Git Имя_файла. Расширение

Удаление файла из Git
$ git rm --cached ИмяФайла.разширение

Добавление всех файлов в коммит
$ git commit -a -m"init"(-a = all, -m = коментарий)

Определение состояния файлов

Основной инструмент, используемый для определения, какие файлы в каком состоянии находятся — это команда git status

```
$ git status
# On branch master
  nothing to commit, working directory clean
```

Это означает, что у вас чистый рабочий каталог, другими словами — в нём нет отслеживаемых изменённых файлов. И наконец, команда сообщает вам на какой ветке (branch) вы сейчас находитесь. Пока что это всегда ветка master — это ветка по умолчанию.

Просмотр истории комитов

Наиболее простой и в то же время мощный инструмент для этого — команда git log. Один из наиболее полезных параметров — это –p, который показывает дельту (разницу/diff), привнесенную каждым коммитом. Вы также можете использовать -2 что ограничит вывод до 2-х последних записей.
Наиболее интересный параметр — это format
$ git log --pretty=format:"%h - %an, %ar : %s"

ca82a6d - Scott Chacon, 11 months ago : changed the version number 085bb3b - Scott Chacon, 11 months ago : removed unnecessary test code a11bef0 - Scott Chacon, 11 months ago : first commit

Параметр	Описание выводимых данных
%H	Хеш коммита
%h	Сокращённый хеш коммита
%T	Хеш дерева
%t	Сокращённый хеш дерева
%P	Хеши родительских коммитов
%p	Сокращённые хеши родительских коммитов
%an	Имя автора
%ae	Электронная почта автора
%ad	Дата автора (формат соответствует параметру --date=)
%ar	Дата автора, относительная (пр. "2 мес. назад")
%cn	Имя коммитера
%ce	Электронная почта коммитера
%cd	Дата коммитера
%cr	Дата коммитера, относительная
%s	Комментарий

Создание ветвлений в Git

Команды:

$ git checkout -b "name"// Имя новой ветки, создать и сразу переключится

Посмотреть ветки

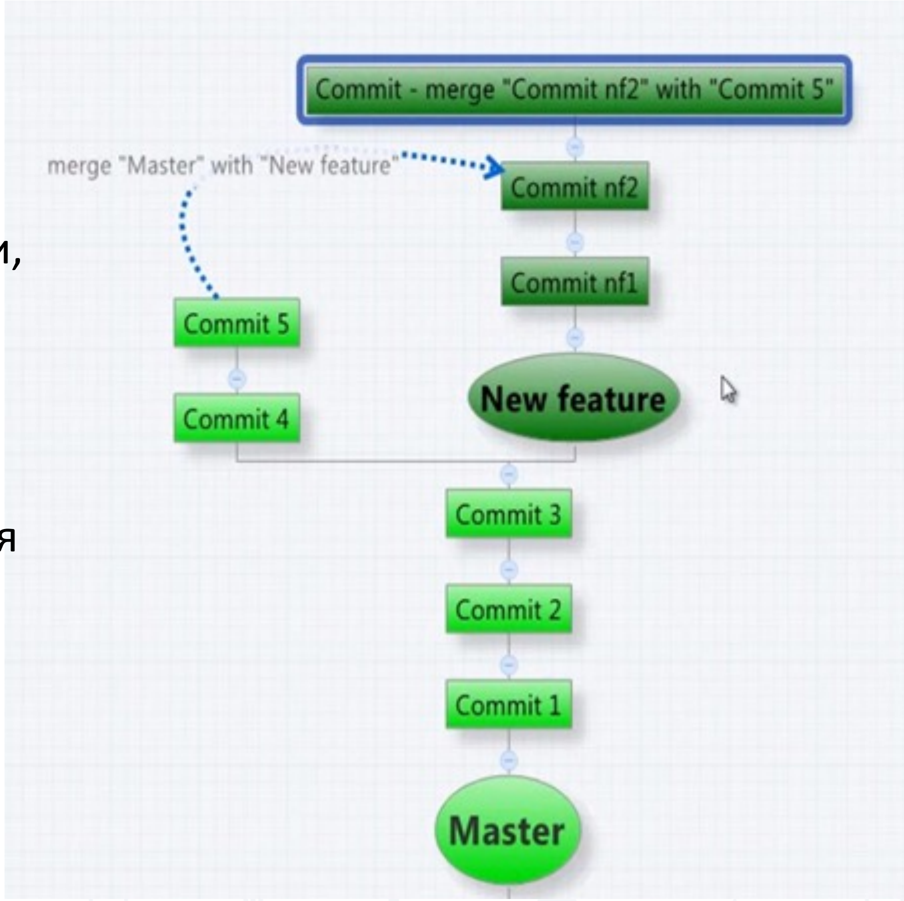
$ git branch (-v доп. инфа и последние комиты)

- Просто создание ветки без переключения

  $ git branch “namebranch”

- Переключение на другую ветку

  $ git checkout "namebranch"



Слияние веток. Разрешение конфликтов при слиянии

Указание утилиты для слияния: $ git config --global merge.tool kdiff3
Команда на слияние (заливаем в свою ветку другую ветку) $ git merge master
Запускаем утилиту для мержа(которую мы прописали в конфиг) $ git mergetool
Надо скачать kdiff3 https://sourceforge.net/projects/kdiff3/files/
И добавляем адресс в Git
$ git config --global mergetool.kdiff3.cmd '"F:\\KDiff3\\kdiff3" $BASE $LOCAL $REMOTE -o $MERGED'
теперь можно мержить
$ git mergetool