Оглавление

Английский язык

Английский язык

GIT

При разработке обычно используется удаленный сервер (как резервная копия). Если над проектом занимается команда (не один человек), то могут возникнуть проблемы. Например, как загружать изменения на удаленный сервер. В этой ситуации поможет GIT.

GIT берет на себя слияние разных версий файлов (merging) и является системой контроля версий (история изменений с возможностью вернуться любому моменту).

В системе контроля версий существует два подхода к хранению данных: централизированный и распределенный. При централизированном подходе проект храниться только на центральном сервере. При распределенном подходе проект храниться на центральном сервере плюс у каждого разработчика есть копия проекта. Второй подход имеет преимущества, т. к. разрабатывать можно офлайн и, если что-то случиться с центральным сервером, то в этом случае копия проекта останется у разработчиков. Git является распределенной системой.

В отличии от других систем контроль версий, которые хранят список изменений,

GIT хранит изменения снимков проекта во времени.

Статусы файлов

- untracked (неотслеживаемый) — файл создан;
- modified (измененный) — файл изменен;
- staged (подготовленный) — git add;
- committed (зафиксированный) — git commit.

Указатели

В GIT есть указатель HEAD. Обычно он указывает на последний (текущий) коммит. Этот указатель можно смещать: HEAD^ или HEAD~1 (1 коммит), HEAD^^ или HEAD~2 (2 коммита) и т. д. Помимо указателей для возврата проекта можно воспользоваться хэшом коммита.

Удаленный репозиторий

Удаленный репозиторий необходим для резервной копии проекта и для того, чтобы другие люди могли видеть наши коммиты.

Популярные удаленные репозитории: GitHub, BitBucket, GitLab. Они предоставляют всю инфраструктуру для хранения и управления GIT-репозиториев.

У одного проекта может существовать несколько удаленных репозиториев с разными именами и адресами (git remote).

SSH

SSH (от анг. «Secure Shell» — «безопасная оболочка») — сетевой протокол, позволяющий производить удаленное управление операционной системой. SSH позволяет безопасно передавать данные в незащищенной среде.

В простом представлении работу SSH можно представить наличием приватных ключей у клиента (локального компьютера) и сервера.

Ссылки для настройки SSH-ключа для github.com:

- проверка наличия SSH-ключа;
- генерация SSH-ключа;
- связка SSH-ключа.

Ветвление

- новые функции разрабатываются в отдельных ветках;
- ветка master содержит стабильную версию проекта, можем вернуться на master в любой момент;
- сразу несколько разработчиков могут работать в своих ветках над своими

задачами, после завершения работы над задачами эти ветки «сливаются» в ветку master.

Слияние веток (merge)

При окончании работы во время слияния текущей ветки в основную с момента ответвления от основной ветки могут быть следующие ситуации:

- Fast-Forward
  - изменений в основной ветке не было,
  - конфликты не могут возникнуть,
  - слияние происходит автоматически,
  - коммит не создается;
- No-Fast-Forward
  - в основную ветку был добавлен коммит,
  - могут возникнуть конфликты,
  - слияние будет происходить автоматически или вручную,
  - создается коммит.

Конфликт происходит при изменении одного и того же файла в разных ветках.
Он решается вручную, т. к. GIT не может самостоятельно слить ветки.

rebase

rebase — альтернатива merge:

- обе команды делают одно и то же — сливают ветки;
- команда merge может создавать merge commit при слиянии (в случае не fast-forward), команда rebase merge commit'а не создает;
- команда merge безопасней, чем rebase — есть отдельный commit, отображающий слияние;
- плюс merge — достоверная полная история commit'ов;
- плюс rebase — лаконичная линейная история без лишних коммитов;
- если в ветке долго велась работа и произошло много изменений лучше использовать merge;
- если ветка была недолгая и произошло мало изменений — можно использовать rebase;
- используйте merge, если вас не просят о rebase.

Команда rebase работает так, будто мы только сделали git pull и сразу добавили в нее изменения. Можно сказать, что новая ветка «перебазировалась» на последний коммит. Или в новую ветку был добавлен

последний коммит из master, а затем, поверх него были добавлены коммиты текущей ветки. Теперь можно делать fast-forward слияние без merge commit'a.

После совершения данной команды коммиты текущей ветки помещаются во временную зону, далее в текущую ветку добавляются все коммиты из ветки master, позже поочередно добавляются все коммиты из временной зоны.

Также можно сделать все наоборот. Можно перейти в ветку мастер и совершить текущую команду из нее. Таким образом сначала добавятся коммиты из новой ветки, а затем новый коммит из мастера.

Разрешение конфликта такое же, как в случае с merge.

Интерактивный rebase

- обычный rebase нужен для манипуляций с ветками, интерактивный rebase работает на одной ветке;
- обычный rebase берет коммиты из другой ветки, перемещает их в нашу ветку и поверх этих коммитов по одному применяет коммиты из временной зоны;
- интерактивный rebase не берет коммиты из другой ветки, он помещает некоторые коммиты из текущей ветки во временную

зону и потом применяет эти коммиты опять к текущей ветке (в момент применения мы можем изменить коммиты);

- несмотря на то, что название команд одинаковое, обычный rebase сильно отличается от интерактивного rebase (разная логика).

Интерактивный rebase работает с коммитами, которые идут после того коммита, который вы указали.

Что можно делать с помощью интерактивного rebase:

- поменять коммиты местами;
- поменять название коммита(ов);
- объединить два коммита в один;
- добавить изменения в существующий коммит;
- разделить коммит на несколько коммитов;
- ...

Команды

Информационные команды:

- `git help` (помощь, документация);
- `git help название_команды` (документация конкретной команды).

Конфигурация:

- `git config --global user.name "имя фамилия"`;
- `git config --global user.email "email"`;
- `git config --global color.ui true`.

Создание нового проекта:

- `mkdir название_проекта` (создание каталога);
- `cd название_проекта` (перейти к данному каталогу);
- `git init` (инициализация репозитория git).

Базовые команды:

- `git status` (узнать текущий статус репозитория);
- `git add` (подготовить файлы к коммиту):
  - `git add .` (все файлы в текущей папке),
  - `git add *.java` (все файлы в текущей папке с расширением .java),
  - `git add someDir/*.java` (все файлы в папке someDir с расширением .java),
  - `git add someDir/` (все файлы в папке someDir),
  - `git add "*.java"` (все файлы в проекте с расширением .java);

- git commit (сделать коммит):
  - git commit -m "сообщение",
  - git commit -a -m "сообщение" (в отличии от примера выше такая вариация позволяет не использовать команду git add),
  - git commit --amend -m "сообщение" (дополняет последний коммит, добавляя в него «свежие» изменения, меняет сообщение последнего коммита, новый коммит не создается);
- git log (история коммитов).

Другие команды:

- git diff (разница между статусами untracked и committed),
  - git diff --staged (staged и committed),
  - git diff COMMIT_ID (текущим состоянием репозитория и указанным коммитом);
- git reset (git reset --mixed HEAD) (отмена изменений, откат к комиту),
  - git reset --hard (git reset --hard HEAD) (возвращает проект к указанному коммиту, при этом полностью удаляет все коммиты после указанного безвозвратно, файлы в статусе untracked остаются без изменений),

- `git reset --mixed` (`git reset --mixed HEAD`) (возвращает проект к указанному коммиту, при этом переводит все коммиты после указанного в неотслеживаемую (unstaged) зону),
- `git reset --soft` (`git reset --soft HEAD`) (возвращает проект к указанному коммиту, при этом переводит все коммиты после указанного в отслеживаемую (staged) зону),
- `git reset HEAD~2`,
- `git reset --soft HEAD^^`,
- `git reset --hard хеш_коммита`,
- файлы в статусе untracked нельзя удалить командой `git reset --hard`, но их можно перевести в любой другой статус, а затем удалить при помощи текущей команды, или воспользоваться следующей командой;

- `git clean` (удаление untracked файлов):
  - `git clean -n` (посмотреть какие файлы будут удалены),
  - `git clean -f` (удалить untracked файлы);

- `git checkout` (перемещения между коммитами, версиями отдельных файлов и ветками):
  - `git checkout хеш_коммита/указатель` (между коммитами, совершать изменения и делать новые коммиты нельзя),
  - `git checkout название_текущей_ветки` (переход обратно к актуальному коммиту);
  - `git checkout хеш_коммита/указатель -- путь_до_файлов` (между версиями файлов в разных коммитах),
  - `git checkout HEAD~3 -- .` (все файлы),
  - `git checkout HEAD^ -- file1 file2` (для файлов file1 и file2),
  - `git checkout -- путь_до_файлов` (`git checkout HEAD -- путь_до_файлов`);
  - `git checkout название_ветки` (между ветками),
  - две черты указывают, что после них идет обычный текст (в нашем случае путь до файла), а не команда или параметр для команды (например, когда наименование ветки совпадет со значением коммита);

- git remote (настройка и просмотр удаленных репозиториев, на компьютере хранится только ссылка на удаленный репозиторий, origin — название этой ссылки):
  - git remote -v (просмотр списка существующих удаленных репозиториев),
  - git remote add название_репозитория URL_репозитория (добавить новый удаленный репозиторий, на компьютере к удаленному репозиторию мы будем обращаться по его названию),
  - git remote remove название_репозитория (удалить репозиторий (ссылку на него)),
  - git remote show название_репозитория (позволяет сравнить актуальность веток локальных и удаленных);
- git push (отправка локального репозитория на удаленный),
  - git push название_репозитория ветка;
  - git push origin master (отправка удаленный репозиторий с именем origin ветку master).
  - git push --delete origin master (удаление ветки в удаленном репозитории);

- git pull (git fetch, git merge) (получения обновлений (новые коммиты) с удаленного репозитория);
  - git pull origin (получить все удаленные ветки и обновления в них),
  - git pull origin название_ветки (получить обновления по определенной ветке).
- git clone URL_репозитория (загрузить репозиторий);
- git branch (работа с ветками),
  - git branch (список веток и ветка с указателем HEAD),
  - git branch -r (список удаленных веток),
  - git branch название_ветки (создание новой ветки),
  - git branch -d название_ветки (удаление ветки),
  - git branch -D название_ветки (удаление ветки, даже если в ней был коммит);
- git merge ветка (слияние текущую и указанную веток);
- git rebase ветка (слияние текущую и указанную веток, разница команд описана выше),

- `git rebase --continue` (принять команду после исправления вручную конфликтов),
- `git rebase --skip` (пропустить коммит, который вызывает конфликт слияния),
- `git rebase --abort` (прекратить слияние),
- `git rebase -i HEAD~3` (интерактивный rebase);

- `git cherry-pick` («взять» коммиты из другой ветки),
  - `git cherry-pick коммит` («взять» один или несколько коммитов из другой ветки, у коммита будет другой хэш),
  - `git cherry-pick --edit коммит` (перенести коммит из другой ветки, но при этом хотим поменять сообщение коммита),
  - `git cherry-pick --no-commit коммит` (хотим перенести изменения из коммита из другой ветки, но при этом не хотим делать коммит в нашей ветке (изменения просто попадут в отслеживаемую зону); это бывает полезно, если мы хотим внести небольшие правки в тот коммит, который мы забираем из другой ветки или слияние двух коммитов из другой ветки в один коммит),

- `git cherry-pick -x коммит` (указывает в сообщении коммита хэш того коммита, из которого мы сделали cherry-pick),
- `git cherry-pick --signoff коммит` (указывает в сообщении коммита имя того пользователя, кто совершил cherry-pick).

Источники

- Курс на Udemy «Git: Полный курс для начинающих и не только», Наиль Алишев, 03.2019.

Операционная система

Linux (bash)

Windows (bat)

Паттерны и алгоритмы

Паттерны

asdf

Алгоритмы

Алгоритм — набор инструкций для выполнения некоторой задачи.

O:

- $O(\log n)$ или $O(\log_2 n)$ — логарифмическое время (бинарный поиск);
- $O(n)$ — линейное время (простой поиск);
- $O(n \log n)$ — эффективные алгоритмы сортировки (быстрая сотрировка);
- $O(n^2)$ — медленные алгоритмы сортировки (сортировка выбором);
- $O(n!)$ — очень медленные алгоритмы (задача о коммивояжере).

Скорость алгоритма измеряется не в секундах, а в темпе роста количества операций. По сути формула описывает, насколько быстро возрастает время выполнения алгоритма с увеличением размера исходных данных.

Бинарный поиск

Работает с отсортированной структурой, делит ее каждый раз на два, время логарифмическое.

asdf

Java Core

Java Core

Инструменты

Инструменты

Фреймворки

Spring

Spring Framework

sdfds

String Core

sdfsd

String Data

sdfds

String Cloud

sdfsd

Spring Security

sdfsdf

Обработка исключений в контроллерах

asdasd

REST vs SOAP

Enterprise Application (корпоративные приложения) и его проблемы:

- объемы данных;
- устаревшие приложения;
- монолитность систем и интеграций;
- внешняя интеграция.

Интеграция — обмен данными между двумя системами.

Интеграция и ее история:

- интеграция через БД;
- интеграция через вызов методов:
  - DCOM, RPC, RMI;
  - CORBA (Common Object Request Broker Architecture, общая архитектура брокера объектных запросов);
  - SOAP, REST;
- SOA.

Интеграция через БД

Преимущества:

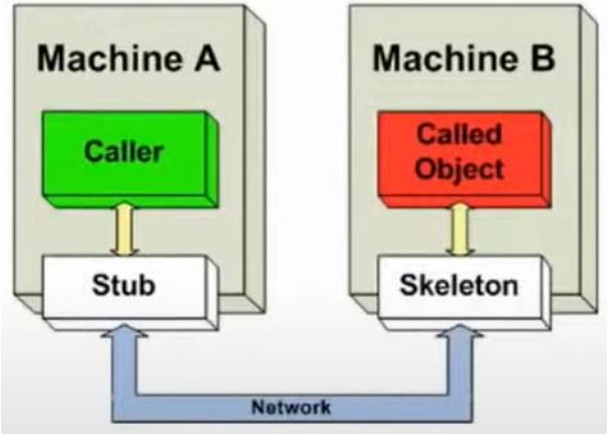
- простота;
- дешевизна.

Недостатки:

- разрастание;

- отсутствие масштабирования;
- нарушения целостности.

RPC (Remote Procedure Call, вызов удаленных процедур)



Machina A вызывает Stub локально, а Stub дальше уже через сеть обращается к Skeleton. Skeleton локально вызывает код на сервере (Machina B). Можно сказать на обеих машинах совершается локальных вызов, но они невидимым образом превращаются в сетевые вызовы.

К подвиду RPC можно отнести Java RMI (Remote Method Invocation, удаленный вызов метода). В отличие от RPC поддерживал объектно-ориентированный подход, но использовал большое количество портов.

DCOM (Distributed Component Object Model, объектный режим распределенных компонентов)

Преимущества:

- технология Microsoft;
- HTTP поддержка.

SOAP, REST

В SOAP интеграция происходит через шину, а не напрямую между приложениями. Таким образом, программист не пишет интеграцию для каждого приложения, а делает только одну интеграцию — это шина. А потом ее средствами перенаправляет эти данные на вход другого сервиса.

REST

REST — это не протокол и не стандарт, а архитектурный стиль.

Принципы стиля:

- Единый интерфейс
  Ресурсы должны быть однозначно идентифицированы посредством одного URL-адреса и только с помощью базовых методов сетевого протокола (DELETE, PUT, GET, HTTP).
- Клиент-сервер

Должно быть четкое разграничение между клиентом и сервером: пользовательский интерфейс и вопросы сбора запросов — на стороне клиента; доступ к данным, управление рабочей нагрузкой и безопасность — на стороне сервера.

- Сохранение состояния
  Все клиент-серверные операции должны быть без сохранения состояния. Любое необходимое управление состоянием должно осуществляться на клиенте, а не на сервере.
- Кэширование
  Все ресурсы должны разрешать кэширование, если явно не указано, что оно невозможно.
- Многоуровневая система
  REST API допускает архитектуру, которая состоит из нескольких уровней серверов.
- Запрос кода
  В большинстве случаев сервер отправляет обратно статические представления ресурсов в формате XML или JSON. Однако при необходимости серверы могут отправлять исполняемый код непосредственно клиенту.

Сравнение

- SOAP — это целое семейство протоколов и стандартов;
- SOAP использует HTTP как транспортный протокол, в то время как REST базируется на нем;
- есть мнение, что разработка RESTful сервисов намного проще;
- REST может быть представлен в различных форматах, а SOAP привязан к XML;
- «REST vs SOAP» можно перефразировать в «Простота vs Стандарты»;
- обработка ошибок;
- SOAP работает с операциями, а REST — с ресурсами.

Выводы

Где REST лучше использовать и почему:

- В сервисах, которые будут использоваться из javascript. Тут и говорить нечего, javascript хорошо работает с json, поэтому именно его и надо предоставлять.
- В сервисах, которые будут использоваться из языков, в которых нет возможности сгенерировать прокси клиента. Это Objective-C, например. Не нужно парсить вручную SOAP-конверт, это незачем.

- Когда существуют очень высокие требования к производительности. Это, как правило, очень интенсивно используемые API, вроде Twitter API или Google API.

Где SOAP лучше использовать и почему:

- Во внешней интеграции между большими (Enterprise) системами.
- В случае использования при пересылке сложных (от сотни полей) объектов, требующих автоматической валидации (например, имеющих небольшое количество консистентных состояний).
- В случае, когда технический диалог с командой, поддерживающей другую часть интеграции, затруднен (например, гос органы).
- Короче — в сложных случаях.

REST

Ни HTTP методы (GET, POST, PUT DELETE), ни коды ответов никак не сопоставляются с бизнесом.

Транзакция REST:

- метод запроса (GET);
- путь запроса (/object/list);
- тело запроса (форма);

- код ответа (200 OK);
- тело ответа (данные в формате JSON).

HATEOAS

HATEOAS (Hypermedia As The Engine Of Application State) — архитектурные ограничения для REST-приложений.

С помощью HATEOAS клиент взаимодействует с сетевым приложением, сервер которого обеспечивает динамический доступ через гипермедиа. REST-клиенту не требуется заранее знать, как взаимодействовать с приложением или сервером за пределом гипермедиа.

JSON Schema

- один из языков описания структуры JSON-документа;
- использует синтаксис JSON;
- базируется на концепциях XML Schema, RelaxNG, Kwalify.

Преимущества RESTful API:

- простота;
- скорость;
- легкость в написании.

Недостатки RESTful API:

- до сих пор нет общего согласования того, что такое RESTful API;
- словарь REST поддерживается не полностью;
- словарь REST недостаточно насыщен и не расширяем;
- RESTful API очень трудно дебажить;
- RESTful API привязан к протоколу.

SOAP

SOAP (Simple Object Access Protocol);

- SOAP (Simple Object Access Protocol, простой протокол доступа к объектам). Основан на XML.
- Рекомендует к использованию W3C (World Wide Web Consortium, Консорциум Всемирной паутины);
- SOAP не зависит ни от платформы, ни от языка.

XML-RPC

- XML-RPC — стандарт/протокол вызова удаленных процедур, использующий XML для кодирования своих сообщений и HTTP в качестве транспортного механизма.
- Является прародителем SOAP, отличается исключительной простотой в применении.

- Был отвергнут Microsoft, как излишне простой.
- Существует по сей день и даже набирает популярность.

WSDL (Web Services Description Language, язык описания веб-сервисов)

- Для описания используется документ формата XML. В нем описываются технические детали: URL-адреса, порт, имена методов, аргументы и типы данных.
- Поскольку WSDL представляет собой XML, он читается человеком и может использоваться машиной, что помогает динамически вызывать службы и привязываться к ним.

UDDI (Universal Description, Discovery and Integration, универсальное описание, обнаружение и интеграция)

- Это служба каталогов.
- Веб-сервис может зарегистрироваться в UDDI и сделать себя доступным через него для обнаружения.

Завершение

- В простых случаях и когда у нас критична скорость работы — REST.
- В сложных случаях, когда у нас не критична скорость, но критична

автоматическая поддержка от технологии — SOAP.

Источник

- Видео на YouTube «Rest web-services vs SOAP Services», Sergey Nemchinskiy, 05.2020.

Сервер

Проект Spring (https://start.spring.io) с зависимостями Spring Web и Spring Web Services.

К текущему проекту добавить зависимости

```
<dependency>
  <groupId>wsdl4j</groupId>
  <artifactId>wsdl4j</artifactId>
</dependency>
```

а также

```
<dependency>
  <groupId>javax.activation</groupId>
  <artifactId>activation</artifactId>
  <version>1.1.1</version>
</dependency>
<dependency>
  <groupId>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
  <version>2.3.1</version>
</dependency>
<dependency>
  <groupId>org.glassfish.jaxb</groupId>
  <artifactId>jaxb-runtime</artifactId>
  <version>2.3.4</version>
</dependency>
```

и плагин:

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>jaxb2-maven-plugin</artifactId>
  <version>2.5.0</version>
  <executions>
    <execution>
      <id>xjc</id>
      <goals>
        <goal>xjc</goal>
```

```
        </goals>
      </execution>
    </executions>
    <configuration>
      <sources>
        <source>
          ${project.basedir}/src/main/resources
              /countries.xsd
        </source>
      </sources>
    </configuration>
  </plugin>
```

Ссылка на POM файл.

В ресурсы поместить файл схемы XML (XSD), из которого при помощи плагина будет создан WSDL: ссылка.

Ссылка на проект. Компилируем его и переходим по ссылке.

Клиент

Ссылка на проект.

Сначала компилируем, потом запускаем.

Источники

- Видео YouTube «SOAP Spring Boot Web Service на примере с нуля. SOAP UI тестирование», Artemy, 10.2021;
- Официальная документация Spring «Producing a SOAP web service»;
- Официальная документация Spring «Consuming a SOAP web service»;

Quarkus

dsfds

Vert.X

sdfsf

Micronaut

sdsdf

Desktop (JavaFX, Swing AWT)

sdfsdf

Web basic

Web basic

Тестирование

Тестирование

Utils

Utils

База данных

База данных

Java Performance

Java Performance