Введение в системы контроля версий и основы Git

Введение

Системы контроля версий (СКВ) играют ключевую роль в современном программировании и разработке программного обеспечения. Они позволяют разработчикам отслеживать изменения в коде, управлять версиями и эффективно работать в команде. В этой лекции мы познакомимся с основными концепциями СКВ, сосредоточив внимание на Git — одной из самых популярных систем контроля версий.

1. Понятие системы контроля версий

Что такое система контроля версий?

Система контроля версий (СКВ) — это программное обеспечение, которое позволяет отслеживать изменения в файлах и управлять разными версиями этих файлов. СКВ особенно полезны при разработке программного обеспечения, когда множество разработчиков работают над одним и тем же проектом и необходимо координировать изменения, чтобы избежать конфликтов.

Зачем нужны системы контроля версий?

СКВ позволяют:

- **Отслеживать историю изменений:** Вы можете увидеть, кто и когда вносил изменения в код, а также просмотреть, что именно было изменено.
- **Откат изменений:** Если что-то пошло не так, вы всегда можете вернуться к предыдущей версии кода.
- **Параллельная работа:** Разработчики могут работать над разными частями проекта одновременно, не мешая друг другу.
- **Создание веток:** Вы можете создавать параллельные версии проекта для экспериментов или разработки новых функций.

Примеры популярных СКВ:

- **Git** — самая популярная система контроля версий, разработанная Линусом Торвальдсом для управления разработкой ядра Linux.
- **Subversion (SVN)** — центральная система контроля версий, широко использовавшаяся до появления Git.
- **Mercurial** — децентрализованная система контроля версий, схожая с Git.

Основные преимущества использования СКВ:

- **История проекта:** Полная история всех изменений, что позволяет легко восстановить предыдущие версии.
- **Работа в команде:** СКВ облегчает командную работу, позволяя разработчикам легко интегрировать свои изменения.
- **Резервное копирование:** Репозиторий служит резервной копией всего проекта.

2. Основные концепции Git

Локальный и удаленный репозиторий

Git управляет проектами через репозитории:

- **Локальный репозиторий** — это версия проекта, находящаяся на вашем компьютере. Вы можете вносить изменения в код, коммитить их и создавать новые ветки.
- **Удаленный репозиторий** — это версия проекта, находящаяся на сервере (например, на GitHub). Разработчики отправляют свои изменения в удаленный репозиторий, чтобы другие участники команды могли их видеть и использовать.

Коммиты, ветки и слияния

- **Коммит (commit)** — это сохранение изменений в истории репозитория. Каждый коммит содержит описание изменений, автора, дату и хэш-сумму (уникальный идентификатор).
- **Ветка (branch)** — это параллельная версия проекта, в которой можно вести разработку независимых функций. Основная ветка обычно называется master или main.
- **Слияние (merge)** — это процесс объединения изменений из одной ветки в другую. Например, можно объединить изменения из экспериментальной ветки в основную ветку.

3. Установка и настройка Git

Установка Git на Windows/Mac/Linux

1. **Windows:**
   - Скачайте установочный файл с официального сайта Git.
   - Запустите установку, следуя инструкциям на экране.
   - Во время установки выберите параметры по умолчанию, если у вас нет специфических требований.
2. **Mac:**
   - Если у вас установлен Homebrew, просто выполните команду в терминале: brew install git.
   - Также можно скачать установочный файл с официального сайта Git.
3. **Linux:**
   - Для Ubuntu/Debian: sudo apt-get install git.
   - Для Fedora: sudo dnf install git.
   - Для других дистрибутивов используйте соответствующие команды для установки пакетов.

Настройка имени пользователя и электронной почты

После установки Git, нужно настроить ваше имя пользователя и электронную почту, чтобы ваши коммиты были правильно подписаны:

```
git config --global user.name "Ваше Имя"
git config --global user.email "ваш.email@example.com"
```

Настройка редактора по умолчанию

По умолчанию Git использует встроенный редактор для ввода сообщений коммитов. Если вы предпочитаете другой редактор (например, VS Code или Vim), вы можете настроить его:

```
git config --global core.editor "code --wait"  # VS Code
git config --global core.editor "vim"          # Vim
```

Основные команды: git init, git config

- **git init** — команда для инициализации нового локального репозитория. Она создаст скрытую папку .git, где будут храниться все данные о версии проекта.
- **git config** — команда для настройки различных параметров Git. Например, имя пользователя, электронную почту, редактор и т.д.

4. Практическая часть

Создание нового репозитория

1. Создайте новую папку на вашем компьютере:

   mkdir my_project

   cd my_project

2. Инициализируйте новый репозиторий Git:

   git init

Первый коммит

1. Создайте файл README.md с кратким описанием проекта:

   echo "# My Project" > README.md

2. Добавьте файл в индекс (stage):

   git add README.md

3. Сделайте первый коммит:

   git commit -m "Initial commit"

Настройка .gitignore

Создайте файл .gitignore, чтобы указать Git, какие файлы и папки не нужно отслеживать. Например, если у вас есть папка с временными файлами:

```
echo "tmp/" > .gitignore
git add .gitignore
git commit -m "Add .gitignore"
```

5. Работа с ветками

Что такое ветка и зачем она нужна?

Ветка (branch) в Git — это независимая линия разработки, которая позволяет вам работать над различными функциями или исправлениями ошибок параллельно, не затрагивая основную (главную) ветку проекта. Ветки облегчают экспериментирование и упрощают интеграцию изменений.

Создание и переключение между ветками

- **Создание новой ветки:**

Для создания новой ветки используется команда git branch. Например, чтобы создать ветку с именем feature-xyz:

```
git branch feature-xyz
```

- **Переключение на ветку:**

Для переключения на существующую ветку используется команда git checkout:

```
git checkout feature-xyz
```

Или можно создать и переключиться на новую ветку одновременно с помощью команды git checkout -b:

```
git checkout -b feature-xyz
```

- **Просмотр списка веток:**

Чтобы увидеть все локальные ветки, используйте:

```
git branch
```

Для просмотра всех веток, включая удаленные:

```
git branch -a
```

Работа с командами git merge и git rebase

- **Слияние веток (git merge):**

Слияние позволяет объединить изменения из одной ветки в другую. Например, чтобы слить ветку feature-xyz в main:

```
git checkout main
git merge feature-xyz
```

При слиянии Git автоматически объединяет изменения, если они не конфликтуют. В случае конфликтов необходимо вручную разрешить их.

- **Перебазирование (git rebase):**

Перебазирование позволяет переместить последовательность коммитов на новую базу. Это полезно для поддержания линейной истории проекта.

Например, чтобы перебазировать ветку feature-xyz на main:

```
git checkout feature-xyz
git rebase main
```

После успешного перебазирования можно слить изменения в main.

Разрешение конфликтов при слиянии веток

Конфликты возникают, когда изменения в разных ветках затрагивают одни и те же строки в одном файле. Git уведомит вас о наличии конфликтов, и вам потребуется вручную их разрешить:

1. **Идентификация конфликтов:**

После попытки слияния Git покажет файлы с конфликтами.

2. **Разрешение конфликтов:**

Откройте конфликтующие файлы и вручную выберите, какие изменения сохранить. Конфликтные участки будут помечены специальными маркерами:

diff

Копировать код

<<<<<<< HEAD

Текущие изменения в основной ветке

=======

Изменения из ветки feature-xyz

>>>>>>> feature-xyz

Удалите маркеры и оставьте только нужный код.

3. **Завершение слияния:**

После разрешения всех конфликтов добавьте исправленные файлы в индекс и завершите слияние:

```
git add <файл>
git commit
```

6. Работа с удаленными репозиториями

Подключение удаленного репозитория (GitHub, GitLab, Bitbucket и т.д.)

Удаленные репозитории позволяют хранить проект на сервере и сотрудничать с другими разработчиками. Наиболее популярными платформами являются GitHub, GitLab и Bitbucket.

- **Добавление удаленного репозитория:**

После создания удаленного репозитория на выбранной платформе, подключите его к вашему локальному репозиторию:

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

Здесь origin — это стандартное имя для основного удаленного репозитория.

Команды для работы с удаленными репозиториями

- **Клонирование удаленного репозитория:**

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

- **Получение изменений из удаленного репозитория:**

  git fetch

- **Слияние изменений из удаленного репозитория в локальную ветку:**

  git pull

- **Отправка изменений в удаленный репозиторий:**

  git push

Настройка SSH-ключей для работы с удаленными репозиториями

Использование SSH-ключей позволяет аутентифицироваться на сервере без ввода пароля.

1. **Создание SSH-ключа:**

   ssh-keygen -t rsa -b 4096 -C "ваш.email@example.com"

Следуйте инструкциям, чтобы сохранить ключ в стандартном месте (~/.ssh/id_rsa).

2. **Добавление SSH-ключа в SSH-агент:**

   eval "$(ssh-agent -s)"

   ssh-add ~/.ssh/id_rsa

3. **Добавление публичного ключа в GitHub/GitLab/Bitbucket:**

- Скопируйте содержимое файла ~/.ssh/id_rsa.pub.
- Перейдите в настройки вашего аккаунта на выбранной платформе.
- Найдите раздел SSH-ключей и добавьте новый ключ, вставив скопированный текст.

Введение в Pull Requests (PR) и Merge Requests (MR)

Pull Requests (в GitHub) и Merge Requests (в GitLab и Bitbucket) позволяют предложить изменения для включения их в основной проект. Это основной механизм для код-ревью и совместной работы.

- **Создание Pull Request:**

После отправки изменений в удаленный репозиторий, перейдите на платформу (например, GitHub) и создайте Pull Request, выбрав ветку с вашими изменениями и целевую ветку для слияния.

- **Обзор и обсуждение:**

Другие участники проекта могут просматривать изменения, оставлять комментарии и предлагать улучшения.

- **Слияние Pull Request:**

После одобрения изменений, Pull Request можно слить в целевую ветку.

7. Практическая часть

Создание новой ветки и внесение изменений в ней

1. **Создание и переключение на новую ветку:**

```
git checkout -b feature-new-feature
```

2. **Внесение изменений:**

Добавьте или измените файлы в проекте.

3. **Добавление и коммит изменений:**

```
git add .
git commit -m "Добавлена новая функция"
```

Слияние ветки с основной веткой (master/main)

1. **Переключение на основную ветку:**

```
git checkout main
```

2. **Слияние изменений из ветки feature-new-feature:**

```
git merge feature-new-feature
```

3. **Удаление ветки после слияния (опционально):**

```
git branch -d feature-new-feature
```

Подключение к удаленному репозиторию и отправка изменений

1. **Добавление удаленного репозитория (если еще не добавлен):**

```
git remote add origin git@github.com:username/repository.git
```

2. **Отправка локальной ветки в удаленный репозиторий:**

```
git push -u origin main
```

Для последующих отправок достаточно использовать git push.

3. **Отправка новой ветки в удаленный репозиторий:**

```
git push -u origin feature-new-feature
```