Введение в контроль версий (лекции)

Урок 3. Углубляемся в контроль версий
Просим вас вести конспекты на лекциях и семинарах.
Это поможет вам:

- лучше запомнить информацию;
- структурировать данные, полученные на лекции;
- грамотно пополнять базу знаний, ориентируясь на план конспекта и дополняя полученные сведения новыми деталями;
- сразу ухватить самое главное из изучаемой темы.

В это окно для сдачи нужно приложить все ваши конспекты по курсу "Введение в контроль версий ".
Мы соберем лучшие приложенные материалы и выложим их в одно место, чтобы вы и ваши одногруппники могли их скачать и пользоваться для лучшего восприятия информации.

1. Лекции прослушал, вот у меня какие мысли по этому поводу:
2. Конспект лекции "Git init add fetch, и т.д.":

Git - это распределенная система контроля версий, которая позволяет отслеживать изменения в исходном коде и управлять ими. Git имеет множество команд, которые позволяют работать с репозиториями. Рассмотрим несколько основных команд: git init, git add, git fetch и другие.

1. Команда git init - используется для создания нового локального репозитория. Например, если вы хотите создать новый репозиторий с именем myrepo, необходимо выполнить следующую команду:
```
git init myrepo
```
2. Команда git add - используется для добавления изменений в индекс для последующего коммита. Например, если вы хотите добавить изменения в файл index.html, необходимо выполнить следующую команду:
```
git add index.html
```
3. Команда git fetch - используется для получения изменений из удаленного репозитория без автоматического объединения с локальным репозиторием. Например, если вы хотите получить изменения из ветки origin/master, необходимо выполнить следующую команду:
```
git fetch origin master
```
4. Команда git commit - используется для сохранения изменений в репозитории. Например, если вы внесли изменения в файл index.html, необходимо выполнить следующие команды:
```
git add index.html
git commit -m "Добавлены изменения в index.html"
```
5. Команда git push - используется для отправки изменений в удаленный репозиторий. Например, если вы хотите отправить изменения в ветку origin/master, необходимо выполнить следующую команду:
```
git push origin master
```

6. Команда git clone - используется для создания локальной копии удаленного репозитория. Например, если вы хотите склонировать удаленный репозиторий с именем myrepo, необходимо выполнить следующую команду:

```
git clone https://github.com/user/myrepo.git
```

    1. Git init, git add, git fetch, git commit, git push и git clone - это основные команды Git, которые позволяют работать с репозиториями и управлять изменениями в исходном коде. Они являются необходимыми для эффективной работы с Git и могут быть использованы для работы с удаленными и локальными репозиториями.
    2. Как я могу создать новую ветку в Git?

   Чтобы создать новую ветку в Git, выполните следующие шаги:

1. Откройте терминал или командную строку и перейдите в локальный репозиторий Git, в котором вы хотите создать новую ветку.
2. Выполните команду git branch, чтобы увидеть список всех веток в вашем репозитории:

```
git branch
```

3. Выберите ветку, от которой вы хотите создать новую ветку, с помощью команды git checkout. Например, если вы хотите создать новую ветку на основе ветки "master", выполните следующую команду:

```
git checkout master
```

4. Создайте новую ветку с помощью команды git branch и укажите имя новой ветки. Например, если вы хотите создать новую ветку с именем "new-feature", выполните следующую команду:

```
git branch new-feature
```

5. Переключитесь на новую ветку с помощью команды git checkout:

```
git checkout new-feature
```

   Теперь вы находитесь в новой ветке, и все изменения, которые вы вносите в ваш репозиторий, будут сохранены в этой ветке. Чтобы отправить изменения в удаленный репозиторий, нужно выполнить команду git push и указать имя новой ветки:

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

   Таким образом, вы создали новую ветку в вашем репозитории Git и можете начать работу в ней.

Команда `git push -f` используется для принудительной отправки изменений в удаленный репозиторий, перезаписывая любые конфликтующие изменения, которые могут существовать на удаленной ветке. Эта команда полезна в ситуациях, когда разработчику необходимо внести значительные изменения в свою локальную ветку, например, переписать историю коммитов или отменить предыдущие изменения. Однако важно заметить, что `git push -f` следует использовать с осторожностью, так как это может иметь непредвиденные последствия. Если другие разработчики внесли изменения в ту же ветку,

принудительная отправка может перезаписать их изменения и вызвать конфликты в репозитории.
Чтобы избежать этих проблем, рекомендуется использовать вместо `git push -f` команду `git push --force-with-lease`. Опция `--force-with-lease` позволяет разработчику принудительно отправлять свои изменения, но только если удаленная ветка соответствует его локальной ветке. Это помогает предотвратить конфликты и гарантирует, что разработчик не перезаписывает изменения других разработчиков.
В целом, всегда хорошей практикой является общение с другими разработчиками и убедиться, что каждый знает о значительных изменениях, вносимых в репозиторий. Это помогает предотвратить конфликты и гарантирует, что все находятся на одной волне.

Как мне удалить ветку в Git?
Чтобы удалить ветку в Git, выполните следующие шаги:

1. Откройте терминал или командную строку и перейдите в локальный репозиторий Git, в котором вы хотите удалить ветку.
2. Выполните команду `git branch`, чтобы увидеть список всех веток в вашем репозитории:
```
git branch
```
3. Выберите ветку, которую вы хотите удалить, с помощью команды `git branch -d` и укажите имя ветки. Например, если вы хотите удалить ветку с именем "my-feature", выполните следующую команду:
```
git branch -d my-feature
```
4. Если ветка, которую вы хотите удалить, не была слита с основной веткой, команда может не сработать и вы увидите предупреждение. В этом случае, чтобы удалить ветку, используйте команду `git branch -D` вместо `git branch -d`. Например:
```
git branch -D my-feature
```
5. После выполнения команды ветка будет удалена из вашего локального репозитория Git.
6. Если вы хотите удалить ветку также и на удаленном сервере, выполните команду `git push` с флагом `--delete` и указанием имени ветки. Например:
```
git push origin --delete my-feature
```
Теперь ветка удалена как локально, так и на удаленном сервере.
3.
В Git есть множество полезных команд, которые помогают управлять репозиториями и эффективно работать с кодом. Вот несколько других наиболее часто используемых команд Git на русском языке:

1. git status - позволяет увидеть текущий статус вашего репозитория, включая измененные, добавленные или удаленные файлы.
2. git branch - позволяет создавать, просматривать, переименовывать или удалять ветки в вашем репозитории.
3. git checkout - позволяет переключаться между ветками или на определенный коммит.

4. git log - позволяет просмотреть историю коммитов, включая автора, дату и сообщение коммита.
5. git reset - позволяет отменить изменения, откатить коммиты или переместить указатель HEAD на другой коммит.
6. git revert - позволяет создать новый коммит, который отменяет изменения, внесенные в предыдущий коммит.
7. git stash - позволяет временно сохранить изменения, которые еще не готовы к коммиту, чтобы вы могли переключиться на другую ветку или получить изменения из удаленных репозиториев.
8. git tag - позволяет пометить определенный коммит номером версии или именем выпуска.
9. git merge --no-ff - позволяет объединить ветку в текущую ветку, сохраняя историю веток и создавая коммит объединения.
10. git cherry-pick - позволяет применить определенный коммит из одной ветки в другую.

    Это только несколько примеров из множества команд Git, которые помогают управлять репозиториями и эффективно работать с кодом. Важно изучить и понять эти команды, чтобы использовать все возможности Git и улучшить ваш рабочий процесс.

    1. Конспект лекции "Git commands merge pull commit remote":

    Git – система контроля версий, которая позволяет отслеживать изменения в коде и управлять ими. Git имеет множество команд, которые помогают в работе с репозиториями.

1. Команда git merge - используется для объединения двух или более веток в одну. Например, если вы хотите объединить ветку feature с веткой master, необходимо выполнить следующие команды:
```
git checkout master
git merge feature
```
2. Команда git pull - используется для получения обновлений из удаленного репозитория и объединения их с текущей веткой. Например, если вы хотите получить обновления из ветки origin/master, необходимо выполнить следующую команду:
```
git pull origin master
```
3. Команда git commit - используется для сохранения изменений в репозитории. Например, если вы внесли изменения в файл index.html, необходимо выполнить следующие команды:
```
git add index.html
git commit -m "Добавлены изменения в index.html"
```
4. Команда git remote - используется для управления удаленными репозиториями. Например, если вы хотите добавить удаленный репозиторий с именем origin и адресом https://github.com/user/repo.git, необходимо выполнить следующую команду:
```
git remote add origin https://github.com/user/repo.git
```

Команды git merge, git pull, git commit и git remote - это основные команды Git, которые помогают в управлении изменениями и совместной работе над проектами. Каждая из этих команд выполняет определенную функцию, и их правильное использование позволяет работать с Git более эффективно.

Конспект лекции "Git rebase и другие команды":
Git rebase - команда, используемая для перебазирования коммитов на другую ветку. Она позволяет перенести коммиты с одной ветки на другую, изменить порядок коммитов и объединить несколько коммитов в один. Рассмотрим несколько других полезных команд Git:

1. Команда git diff - используется для показа различий между файлами, коммитами или ветками. Например, если вы хотите увидеть различия между текущим коммитом и предыдущим коммитом, необходимо выполнить следующую команду:
```
git diff HEAD~1 HEAD
```
2. Команда git rebase - используется для переноса коммитов с одной ветки на другую с сохранением истории коммитов. Например, если вы хотите перенести коммиты с ветки feature на ветку master, необходимо выполнить следующие команды:
```
git checkout feature
git rebase master
```
3. Команда git cherry-pick - используется для применения указанного коммита к текущей ветке. Например, если вы хотите применить коммит с хеш-кодом abc123, необходимо выполнить следующую команду:
```
git cherry-pick abc123
```
4. Команда git bisect - используется для поиска коммита, который внес ошибку. Например, если вы заметили ошибку в вашем проекте, но не знаете, в каком коммите она появилась, вы можете использовать команду git bisect для поиска:
```
git bisect start
git bisect bad HEAD
git bisect good HEAD~10
```
5. Команда git clone - используется для создания локальной копии удаленного репозитория. Например, если вы хотите склонировать удаленный репозиторий с именем myrepo, необходимо выполнить следующую команду:
```
git clone https://github.com/user/myrepo.git
```

Это только некоторые из многих команд, которые доступны в Git. Но понимание этих команд поможет вам лучше управлять вашими репозиториями и улучшить ваш рабочий процесс. Вы можете использовать эти команды, чтобы быстро переключаться между ветками, переносить коммиты, искать ошибки и многое другое.
Введение в контроль версий (семинары)
Урок 1. Первое использование контроля версий
Дооформить инструкцию по работе с Git, используя возможности Markdown (цитаты, картинки, ссылки и т.п.) и приложить свой проект в заархивированном виде (всю папку целиком).

Введение в контроль версий с использованием Git

Цель урока

Научиться использовать контроль версий на примере Git и понять, как он может облегчить работу с проектами.

Шаги по работе с Git

1. Установить Git на свой компьютер (если еще не установлен).
2. Создать локальный репозиторий Git в папке проекта.
3. Добавить файлы проекта в индекс Git с помощью команды `git add`.
4. Сделать коммит с помощью команды `git commit`. В комментарии к коммиту описать изменения.
5. Опционально: создать ветку Git для экспериментов или новых функций.
6. Опубликовать локальный репозиторий на удаленный сервер с помощью команды `git push`.
7. Получить изменения, внесенные другими участниками проекта, с помощью команды `git pull`.

Пример использования Git

Предположим, мы создаем простой проект в папке `my_project`. Для начала, мы инициализируем локальный репозиторий и добавляем в него все файлы проекта:

```
cd my_project
git init
git add .
git commit -m "Initial commit"
```

После этого мы можем создать удаленный репозиторий на GitHub или другом сервисе и опубликовать наши изменения:

```
git remote add origin https://github.com/my_username/my_project.git
git push -u origin master
```

Теперь мы можем работать над проектом вместе с другими участниками. Если кто-то внес изменения в удаленный репозиторий, мы можем получить их с помощью команды `git pull`:

```
git pull
```

Проект в заархивированном виде

Мой проект находится в папке `my_project`. Я приложил ее в заархивированном виде, чтобы вы могли скачать и изучить ее:
my_project.zip

Дополнительные ресурсы

- Официальный сайт Git
- Книга "Pro Git"

2.

Введение в контроль версий (семинары)
Урок 2. Второе использование контроля версий
Продолжить работу с файлом, начатую на Семинаре 1. Создать и слить как минимум 4 ветки. Обязательно создать конфликт и разрешить его. Архив с репозиторием и проделанной работой приложить к уроку.:
Контроль версий - это система, которая позволяет отслеживать изменения в файлах и папках, хранить различные версии файлов и возвращаться к предыдущим версиям в случае необходимости. Это очень полезно при работе в команде, когда несколько человек работают над одним проектом и вносят изменения в один и тот же файл.

Для продолжения работы с файлом, начатой на Семинаре 1, необходимо выполнить следующие шаги:

1. Создать репозиторий на любом из веб-хостингов контроля версий, например, GitHub или GitLab.
2. Клонировать репозиторий на свой компьютер с помощью команды `git clone <url репозитория>`.
3. Создать новую ветку с помощью команды `git branch <название ветки>`.
4. Переключиться на новую ветку с помощью команды `git checkout <название ветки>`.
5. Внести изменения в файл.
6. Закоммитить изменения с помощью команды `git commit -m "<описание изменений>"`.
7. Повторить шаги 3-6 еще три раза, создавая новые ветки, внося изменения и коммитя их.
8. Изменить одну строку в файле в одной из веток.
9. Закоммитить изменения в этой ветке.
10. Переключиться на другую ветку и изменить ту же строку, но на другое значение.
11. Закоммитить изменения в этой ветке.
12. Попытаться слить эти две ветки с помощью команды `git merge <название ветки>`.
13. Увидеть сообщение об ошибке о том, что произошел конфликт.
14. Разрешить конфликт вручную, выбрав нужное значение строки.
15. Закоммитить изменения после разрешения конфликта.
16. Запушить все ветки и изменения в удаленный репозиторий с помощью команды `git push origin --all`.
17. Проверить, что все изменения и ветки успешно загрузились в удаленный репозиторий.
18. Создать архив с репозиторием и проделанной работой и приложить его к уроку.

Таким образом, мы продолжили работу с файлом, начатую на Семинаре 1, создали несколько веток, внесли изменения и разрешили конфликт, используя контроль версий. Это позволило нам эффективно работать над проектом в команде и сохранять историю изменений.

continue Введение в контроль версий (семинары)

Урок 3. Работа с удалёнными репозиториями

Дополнить файл с инструкцией по работе с git (второе домашнее задание) и направить pull request в репозиторий преподавателя.

(https://github.com/MarzipanCoin/git_seminars)

Файл с инструкцией необходимо дополнить информацией о работе с удаленными репозиториями.

В системе подгрузить скриншот отправленного pull request.:

Для работы с удаленными репозиториями в Git необходимо выполнить следующие шаги:

1. Создать удаленный репозиторий на любом из веб-хостингов контроля версий, например, GitHub или GitLab.
2. Связать локальный репозиторий с удаленным репозиторием с помощью команды `git remote add <название репозитория> <url репозитория>`.
3. Загрузить локальные изменения в удаленный репозиторий с помощью команды `git push <название репозитория> <название ветки>`.

4. Получить изменения из удаленного репозитория с помощью команды `git pull <название репозитория> <название ветки>`.
5. Просмотреть список удаленных репозиториев с помощью команды `git remote -v`.
6. Удалить связь с удаленным репозиторием с помощью команды `git remote remove <название репозитория>`.
7. Изменить имя удаленного репозитория с помощью команды `git remote rename <старое название> <новое название>`.
8. Внести изменения в удаленный репозиторий через pull request. Для этого необходимо выполнить следующие шаги:

   8.1. Создать ветку в локальном репозитории с помощью команды `git branch <название ветки>`.

   8.2. Переключиться на созданную ветку с помощью команды `git checkout <название ветки>`.

   8.3. Внести изменения в файлы.

   8.4. Закоммитить изменения с помощью команды `git commit -m "<описание изменений>"`.

   8.5. Отправить ветку в удаленный репозиторий с помощью команды `git push -u <название репозитория> <название ветки>`.

   8.6. Создать pull request на сайте удаленного репозитория.

   8.7. Ждать подтверждения и слияния изменений в удаленный репозиторий.
   Для дополнения файла с инструкцией по работе с Git информацией о работе с удаленными репозиториями, необходимо добавить описанные выше шаги. После этого необходимо отправить pull request в репозиторий преподавателя. Для этого нужно выполнить следующие шаги:

1. Сделать fork репозитория преподавателя на GitHub.
2. Склонировать свой fork репозитория на свой компьютер с помощью команды `git clone <url репозитория>`.
3. Добавить изменения в файл с инструкцией.
4. Закоммитить изменения с помощью команды `git commit -m "<описание изменений>"`.
5. Отправить изменения в свой fork репозиторий с помощью команды `git push origin <название ветки>`.
6. Создать pull request на сайте GitHub, выбрав свой fork репозитория и ветку с изменениями.
7. Ожидать подтверждения и слияния изменений в репозиторий преподавателя.
   Чтобы подгрузить скриншот отправленного pull request в систему, необходимо выполнить следующие шаги:

1. Открыть сайт GitHub и перейти в свой fork репозитория с инструкцией.
2. Найти созданный pull request и открыть его.
3. Сделать скриншот экрана с открытым pull request.

4. Сохранить скриншот в удобном формате (например, PNG или JPEG).
5. Загрузить скриншот в систему, используя соответствующую функцию загрузки файла или перетащив его в окно браузера.
6. Убедиться, что скриншот успешно загружен и отображается в системе.
   Таким образом, мы дополнили файл с инструкцией по работе с Git информацией о работе с удаленными репозиториями и отправили pull request в репозиторий преподавателя. Также мы подгрузили скриншот отправленного pull request в систему для дальнейшей проверки.
   1.