Системы управления версиями Git и GitHub

Системы управления версиями (*Version control system, VCS*) – программное обеспечение, предназначенное для управления быстро меняющейся информацией. Системы управления версиями позволяют сохранять нужное количество предыдущих версий информации (обычно – текстового документа, в том числе с программным кодом, или набора таких документов – проекта), обращаться к этим версиям при необходимости, а также позволяют организовать редактирование одного и того же документа разными пользователями.

Существуют централизованные и распределенные системы управления версиями. В централизованных *VCS* функции по хранению версий документа и предоставлению доступа к ним берет на себя сервер. Для внесения изменений в какой-либо документ пользователю в первую очередь обычно требуется скачать на свой компьютер «рабочую копию» нужной ему версии документа или обновить уже имеющуюся версию до последней (update). После внесения изменений пользователь закачивает новую версию документа на сервер (*commit*), причем предыдущая версия не удаляется, а также сохраняется на сервере. Такой подход имеет ряд недостатков, в том числе риски потери информации.

В децентрализованных *VCS* пользователи скачивают на свои рабочие компьютеры не только нужные им версии нужных им файлов, а полностью весь репозиторий (хранилище файлов). В этом случае при выходе из строя сервера вся информация может быть восстановлена из рабочих компьютеров пользователей. Системы управления версиями хранят информацию о том, кто, когда и какие изменения выполнил, позволяет сравнивать две версии документа, а также реализует функцию ветвления. Создание «ветви» (*branch* или *fork*) позволяет создать две копии одного и того же набора документов, развивать их параллельно и при необходимости снова слить в один проект (*merge*). Ветвление позволяет, например, поддерживать несколько версий одного и того же продукта и зачастую выполняется перед началом работы над достаточно объемным обновлением для продукта.

Система *Git* – распределенная кроссплатформенная система управления версиями, созданная в 2005 году для управления разработкой ядра *Linux*. Благодаря своему удобству и скорости работы является в настоящий момент одной из самых популярных *VCS*. С *Git* можно работать как посредством командной строки, так и при помощи любого из множества графических интерфейсов. Шпаргалка по командам git может быть найдена найдена, например, на официальном сайте GitHub https://training.github.com/downloads/ru/github-git-cheat-sheet/.

Задание 1.

1. Установка *git*:

   «*sudo apt install git*».

2. Прежде чем пользоваться Git необходимо настроить свой профиль в системе.

   Для настройки профиля можно воспользоваться командой

   «git config --global user.name "ваше имя"».

   Также необходимо ввести эл. почту:

   «git config --global user.email "..."».

   В примере выше указана опция *--global*. Это значит, что такие данные будут сохранены для всех действий в *Git* в любых проектах. Если необходимо менять эту информацию для разных проектов, то в директории проекта нужно будет вводите эти же команды, только без опции *--global*.

3. Создайте проект, в котором должен быть один или несколько файлов либо директория. Репозиторий должен храниться в отдельной директории.

4. Для того, чтобы папка стала репозиторием *Git*, в папке надо написать команду

   «git init».

5. Проверьте наличие папки «*.git*» в репозитории.

   В директории «*.git*» будет собираться вся информация о дальнейшей работе.

6. Проиндексировать содержимое проекта проекта командой

   «git add .».

7. Создайте первый коммит репозитория комодной

   «git commit -m "First Commit"».

   Ключ «*-m*» позволяет сохранить сообщение (*message*). Также можно производить коммит и индексацию в одно действие с использованием дополнительного ключа *«-a».*

8. Для изменений проекта рекомендуется использовать отдельные ветки. Для создания новой ветки используйте команду

   «git branch <branch name>».

9. Переключитесь на новую ветку командой

   «git checkout <branch name>»

10. Создайте новый фал в репозитории.
11. Проверьте статус ветки командой

    «git status».

12. Проиндексируйте новый файл в текущей ветке.
13. Проверьте статус ветки.

14. Добавьте коммит к ветке и проверьте ее статус. Опишите изменения статусов.
15. Проверьте какие сейчас есть ветки командой

    «*git brunch*».

    Сравните результат с содержимым каталога

    «ls ./.git/refs/heads/».
16. Для слияния вновь созданной ветки и основной необходимо переключиться на основную ветку. Проверьте, что вы переключились.
17. Для слияния используйте команду

    «git merge <branch name>».
18. Удалите созданный файл находясь в основной ветке.
19. Перейдите в ветку, где файл был создан. Проверьте историю ветку при помощи команды

    «*git log*».

    Данную команду можно использовать с ключами, например, *«--oneline»* выдает только базовую информацию; ключ *«-p»* позволяет вывести различие между каждым коммитом, а ключ *«-<число>»* – ограничивает вывод количеством коммитов, заданных числом.
20. Создайте метку последнего коммита комодной

    «git tag <tag name>»
21. Проверьте логи.
22. Откатитесь к последнему коммиту командой

    «git reset --hard HEAD» (или номер коммита).
23. Проверьте состояние проекта.
24. Вновь удалите созданный файл и создайте соответствующий коммит с названием тест.
25. Восстановите состояние системы на момент коммита когда удаленный файл существовал.
26. Проверьте лог ветки, опишите изменения.
27. Создайте новый файл и проиндексируйте его (не коммит).
28. Посмотрите изменения по сравнению с последним коммитом при помощи команды

    «git diff HEAD»

    Посмотрите изменения по сравнению с предпоследним коммитом при помощи команды

    «*git diff HEAD^*»

    Посмотрите изменения по сравнению с веткой мастер при помощи команды

«*git diff master*»

29. Сделайте коммит созданного файла и затем запишите в него информацию и проиндексируйте изменения. Проверьте изменения файла командой

    «*git diff blame <file name>*»

30. Откатите репозиторий на коммит с созданной меткой.
31. Посмотрите текущее состояние репозитория командой

    «*git show*»

    https://csc-software-development.readthedocs.io/ru/latest/01-intro.html

32. Создайте псевдоним новой команде *git*, например

    git config --global alias.last-commit "log –oneline -p -1"

    Проверьте работу псевдонима.

33.

    Часто в проекте существуют файлы. которые нет необходимости индексировать. К таким файлам обычно относятся автоматически генерируемые файлы (различные логи, результаты сборки программ и т.п.). Для этих случаев в системе *Git* предусмотрен файл шаблонов «*.gitignore*». В данном файле можно указывать какие шаблоны файлов не подлежат индексации. К шаблонам применяются следующие правила:

    - пустые строки, а также строки, начинающиеся с символа # (символ комментария), игнорируются.
    - можно использовать стандартные glob шаблоны (упрощенные регулярные выражения). Примеры таких шаблонов могут быть найдены, например, тут https://devacademy.ru/article/ignorirovanie-faylov-i-katalogov-v-git.
    - можно заканчивать шаблон символом слэша (/) для указания каталога.
    - можно инвертировать шаблон, использовав восклицательный знак (!) в качестве первого символа.

Задание 2.

1. Продолжить работу с созданным репозиторием на первой лабораторной работе. Проведите коммит всех изменений на данный момент.
2. Создать папку *temp* в своем репозитории. Создать папку *log* и добавить в нее 2 файла: *main.html* и *some.tmp*.
3. Создать файл *.gitignore*.
4. Добавьте в игнорирование папку *temp* и файлы с расширением «*.tmp*» (в т.ч. из папки *log*).
5. Проиндексируйте репозиторий и проверьте его статус.

6. Добавьте в репозиторий temp новый файл и вновь проверьте статуст.
7. Объясните изменения.

Часто в больших проектах требуется не только работа на своем ПК, но и удаленная работа. Например, это характерно если работа ведется командой разработчиков. Для поддерживание согласованного состояния репозиториев команды обычно используется сервер, который хранить центральный репозиторий. Система *git* содержит команды для согласования работы репозиториев (клонирование, синхронизация изменений и др.). Кроме того, специальные серверные системы на основе *git*, такие как, например, как *github.com* или *gitlab.com*, предоставляют дополнительные возможности по управлению рабочим процессом создания проекта.

Задание 3.

1. Создайте новый проект и инициализируйте его с помощью *git.*
2. Клонируйте репозиторий *https://github.com/MVRonkin/operation-systems.git* при помощи команды

   «git clone <url> <name of dir>».

   Система *git* поддерживает различные способы ссылки на удаленный репозиторий. Два наиболее простых способа доступа к удаленному репозиторию: протоколы *HTTP* и *SSH*. Протокол *HTTP* — простой способ разрешить к репозиторию анонимный доступ только для чтения. Для подключения по протоколу *HTTP* репозиториев предусмотрены url страницы *.git.*
3. Проверьте создание репозитория. И зайдите в созданную директорию.
4. Проверьте URL подключенные к директории командой

   «git remote -v».

   По сути, команда *git remote* — это интерфейс для управления списком записей об удаленных подключениях, которые хранятся в файле */.git/config* репозитория.
5. Найдите имя подключения при помощи команды

   «git remote».
6. Переименуйте подключение с имени по умолчанию на другое командой

   «git remote rename <old-name> <new-name>»
7. Посмотрите информацию о подключённом репозитории комодной

   «git remote show <name>».
8. Удалите подключение при помощи команды

   «git remote rm <name>».

Задание 4.

1. Пройдите регистрацию на *github.com* или любом другом сервисе поддержки git.

   Если вы впервые на *github.com* то также необходимо создать токен, при помощи следующей инструкции:

   https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token

   В настройках токена необходимо указать разрешения на совершения действий с репозиториями.
2. Создайте тестовый репозиторий в сервисе.
3. Создайте папку проектов. Папка должна быть инициализирована системой *git*.
4. Попробуйте подключиться удаленно к репозиторию при помощи команды

   «git remote add <name> <url> ».

   Данная команда позволяет управлять подключениями к удаленным репозиториям. Такого же результата можно достичь, напрямую отредактировав файл */.git/config* с помощью текстового редактора.

   Стандартное название *remote* подключения *origin.*
5. Проверьте, что вы подключились к репозиторию.
6. Создайте файл с расширением «*.md*», например стандартный файл *README.md.*
7. Создайте соответствующий коммит. Проверьте коммит, ветку в которой вы работаете и название репозитория.
8. Создайте запрос на внесение изменений комадной

   «git push <remote_name> <branch_name>»

   Команда *git push* чаще всего используется для публикации выгружаемых локальных изменений в центральном репозитории. Команда создает локальную ветку в репозитории назначения. Если необходимо выполнить слияние веток принудительно, то используйте команду с ключом *-f (--force)*. Если необходимо опубликовать все ветки, используйте ключ *--all*. Также можно использовать ключ *--tags* для публикации только помеченных коммитов.
9. Перейдите в свой репозиторий, например на *GitHub* и найдите свой тестовый репозиторий. Найдите кнопку «*Compare & pull request*» проделайте действия для слияния репозиториев.
10. Клонируйте репозиторий в текущем состоянии. Проверьте, что у вас есть ветка *main.* Внесите изменения в основную ветку. Проведите слияние с удаленным репозиторием. Проверьте состояние удаленного репозитория по каждой из веток.
11. Отредактируйте созданный файл удаленно, например, с вебсайта GitHub.com в основной ветке.

12. Внесите изменения в свой локальный репозиторий комодной *«git pull <remote_name> <branch-name>».*
13. Проверьте результат.
14. Сравните методы *clone, pull* и *remote add.*