Лабораторная работа №1

Работа с git

Шубнякова Дарья НКНбд-01-22

Содержание

# 1. Цель работы

Ознакомиться с работой git.

# 2. Задание

Реализовать на практике команды и понять, как они работают.

# 3. Теоретическое введение

Git — это распределённая система управления версиями (СКВ), созданная для отслеживания изменений в коде, координации работы множества разработчиков и управления различными версиями проектов. По сути, это как “машина времени” для вашего кода, позволяющая возвращаться к предыдущим состояниям проекта.

# 4. Выполнение лабораторной работы

Установка имени и электронной почты. Параметры установки окончаний строк. Установка отображения unicode. Создаем страницу «Hello,World»([рис. 1](#fig-001)).

|  |
| --- |
| Рисунок 1 |

Чтобы создать git репозиторий из этого каталога, выполняем команду git init. Добавим файл в репозиторий. Используем команду git status, чтобы проверить текущее состояние репозитория([рис. 2](#fig-002)).

|  |
| --- |
| Рисунок 2 |

Добавим кое-какие HTML-теги к нашему приветствию. Меняем содержимое файла hello.html([рис. 3](#fig-003)).

|  |
| --- |
| Рисунок 3 |

Сделаем коммит и проверим состояние. git commit Откроется редактор. В первой строке вводим комментарий: «Added h1 tag». Сохраняем файл и выходим из редактора (для этого в редакторе по-умолчанию (Vim) вам нужно нажать клавишу ESC, ввести :wq и нажать Enter). Теперь еще раз проверим состояние. ([рис. 4](#fig-004)).

|  |
| --- |
| Рисунок 4 |

Изменяем страницу «Hello, World», чтобы она содержала стандартные теги

и

.([рис. 5](#fig-005)).

|  |
| --- |
| Рисунок 5 |

Производим коммит проиндексированного изменения (значение по умолчанию), а затем еще раз проверяем состояние([рис. 6](#fig-006)).

|  |
| --- |
| Рисунок 6 |

Сделали коммит второго изменения([рис. 7](#fig-007)).

|  |
| --- |
| Рисунок 7 |

Затем проверяем содержимое файла hello.html. Создаем тег первой версии. Это версия c тегами

и

, но еще пока без

. Давайте сделаем ее версией v1-beta([рис. 8](#fig-008)).

|  |
| --- |
| Рисунок 8 |

Внесем изменение в файл hello.html в виде нежелательного комментария. ([рис. 9](#fig-009)).

|  |
| --- |
| Рисунок 9 |

Используем команду git checkout для переключения версии файла hello.html в репозитории.([рис. 10](#fig-010)).

|  |
| --- |
| Рисунок 10 |

Проиндексируем это изменение.([рис. 11](#fig-011)).

|  |
| --- |
| Рисунок 11 |

К счастью, вывод состояния показывает нам именно то, что мы должны сделать для отмены индексации изменения. git reset HEAD hello.html([рис. 12](#fig-012)).

|  |
| --- |
| Рисунок 12 |

Изменяем файл hello.html на следующий. Выполняем: git add hello.html git commit -m “Oops, we didn’t want this commit”([рис. 13](#fig-013)).

|  |
| --- |
| Рисунок 13 |

Чтобы отменить коммит, нам необходимо сделать коммит, который удаляет изме- нения, сохраненные нежелательным коммитом. Проверка лога показывает нежелательные и отмененные коммиты в наш репозиторий([рис. 14](#fig-014)).

|  |
| --- |
| Рисунок 14 |

Но прежде чем удалить коммиты, давайте отметим последний коммит тегом, чтобы потом можно было его найти([рис. 15](#fig-015)).

|  |
| --- |
| Рисунок 15 |

git tag oops([рис. 16](#fig-016)).

|  |
| --- |
| Рисунок 16 |

Создаем директорию lib.([рис. 17](#fig-017)).

|  |
| --- |
| Рисунок 17 |

Добавим в страницу комментарий автора (вставим свою фамилию)([рис. 18](#fig-018)).

|  |
| --- |
| Рисунок 18 |

Выполним: git add hello.html git commit -m “Add an author comment”. Выполним: ls -C .git/objects Мы должны увидеть набор каталогов, имена которых состоят из 2 символов. Имена каталогов являются первыми двумя буквами хэша sha1 объекта, хранящегося в git.([рис. 19](#fig-019)).

|  |
| --- |
| Рисунок 19 |

Выполним: git log –max-count=1 Эта команда должна показать последний коммит в репозиторий. SHA1 хэш в вашей системе, вероятно, отличается от моего, но вы увидите что-то наподобие этого([рис. 20](#fig-020)).

|  |
| --- |
| Рисунок 20 |

Давайте назовем нашу новую ветку «style». Выполним: git checkout -b style([рис. 21](#fig-021)).

|  |
| --- |
| Рисунок 21 |

Добавим файл стилей style.css([рис. 22](#fig-022)).

|  |
| --- |
| Рисунок 22 |

Теперь в вашем проекте есть две ветки. Выполним: git log –all([рис. 23](#fig-023)).

|  |
| --- |
| Рисунок 23 |

git checkout master cat lib/hello.html Сейчас мы находимся на ветке master. Это заметно по тому, что файл hello.html не использует стили style.css([рис. 24](#fig-024)).

|  |
| --- |
| Рисунок 24 |

Выполним: git checkout style cat lib/hello.html Содержимое lib/hello.html подтверждает, что мы вернулись на ветку style.([рис. 25](#fig-025)).

|  |
| --- |
| Рисунок 25 |

Выполним: git log –graph –all Добавление опции –graph в git log вызывает построение дерева коммитов с помощью простых ASCII символов.([рис. 26](#fig-026)).

|  |
| --- |
| Рисунок 26 |

Слияние переносит изменения из двух веток в одну. Давайте вернемся к ветке style и сольем master с style. Выполним: git checkout style git merge master git log –graph –all([рис. 27](#fig-027)).

|  |
| --- |
| Рисунок 27 |

Вернемся в ветку master и внесем изменения git checkout master([рис. 28](#fig-028)).

|  |
| --- |
| Рисунок 28 |

Используем команду rebase вместо команды merge. Мы вернулись в точку до пер- вого слияния и хотим перенести изменения из ветки master в нашу ветку style. На этот раз для переноса изменений из ветки master мы будем использовать команду git rebase вместо слияния([рис. 29](#fig-029)).

|  |
| --- |
| Рисунок 29 |

Выполним: git checkout master git merge style Поскольку последний коммит ветки master прямо предшествует последнему коммиту ветки style, git может выполнить ускоренное слияние-перемотку. При быстрой перемотке вперед git просто передвигает указатель вперед, таким обра- зом указывая на тот же коммит, что и ветка style. При быстрой перемотке конфликтов быть не может([рис. 30](#fig-030)).

|  |
| --- |
| Рисунок 30 |

Создадим клон репозитория. Выполним: git clone hello cloned\_hello ls([рис. 31](#fig-031)).

|  |
| --- |
| Рисунок 31 |

Мы увидели список всех файлов на верхнем уровне оригинального репозитория README.md, index.html и lib([рис. 32](#fig-032)).

|  |
| --- |
| Рисунок 32 |

git remote Мы видим, что клонированный репозиторий знает об имени по умолчанию удаленного репозитория. Давайте посмотрим, можем ли мы получить более подробную информацию об имени по умолчанию: Выполним: git remote show origin Удаленные репозитории обычно размещаются на отдельной машине, возможно, централизованном сервере.([рис. 33](#fig-033)).

|  |
| --- |
| Рисунок 33 |

Для того, чтобы увидеть все ветки, попробуем следующую команду: git branch -a([рис. 34](#fig-034)).

|  |
| --- |
| Рисунок 34 |

Внесем следующие изменения в файл README.md: Файл README.md This is the Hello World example from the git tutorial.([рис. 35](#fig-035)).

|  |
| --- |
| Рисунок 35 |

Выполним: git branch –track style origin/style git branch -a git log –max-count=2([рис. 36](#fig-036)).

|  |
| --- |
| Рисунок 36 |

Создаем чистый репозиторий cd .. git clone –bare hello hello.git ls hello.git([рис. 37](#fig-037)).

|  |
| --- |
| Рисунок 37 |

Файл README.md: This is the Hello World example from the git tutorial. (Changed in the original and pushed to shared) Выполним: git checkout master git add README git commit -m “Added shared comment to readme”([рис. 38](#fig-038)).

|  |
| --- |
| Рисунок 38 |

Теперь отправим изменения в общий репозиторий. Выполним: git push shared master Выполним: git remote add shared ../hello.git git branch –track shared master git pull shared master cat README.md([рис. 39](#fig-039)).

|  |
| --- |
| Рисунок 39 |

# 5. Выводы

Мы настроили git и ознакомились со всеми полезными командами для дальнейшей работы.