МІНІСТЕРСТВО ОСВІТИ І НАУКИ

ХАРКІВСЬКИЙ КОМП′ЮТЕРНО-ТЕХНОЛОГІЧНИЙ КОЛЕДЖ

НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ

«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

**ЗВІТ**

**ПРО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ № 2**

з дисципліни «Соціальні мережі»

|  |  |
| --- | --- |
| Виконав  студент гр. ОПК-315  Дробот Олександр | Перевірив  викладач  Фесенко Д.В. |

Харків 2017**ЛАБОРАТОРНА РОБОТА №2**

«Основи галуження у Git. Робота з гілками»

**1. Тема роботи:** Основи використання галуження при створенні проектів.

**2 . Мета роботи:** Опанувати навичками роботи з гілками.

**Хід роботи:**

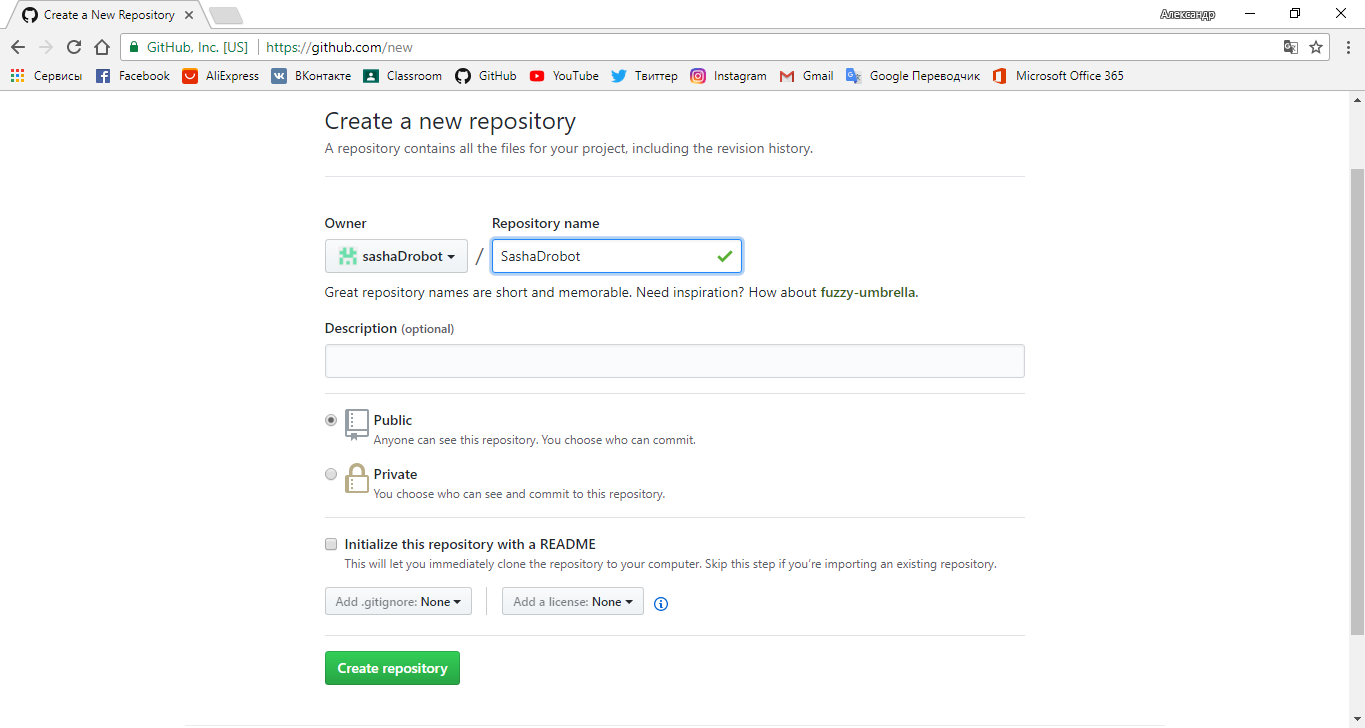


Рисунок 2.1 – Створення власного репозиторію на Гіт.

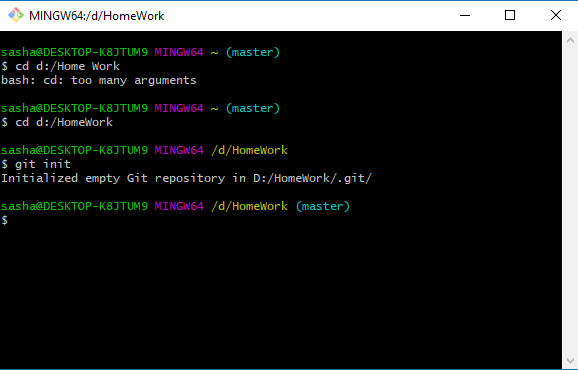


Рисунок 2.2 – Створення локального репозиторію на комп’ютері.

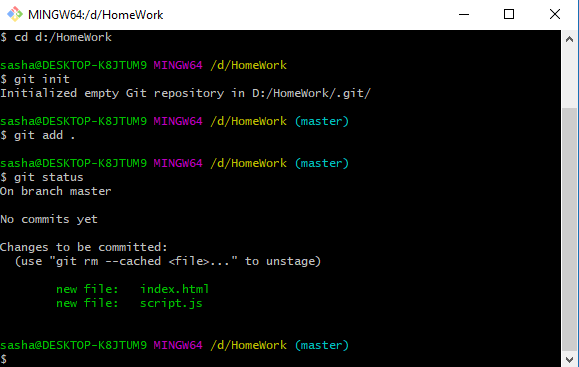


Рисунок 2.3 – Індексування проекту в робочому каталозі.

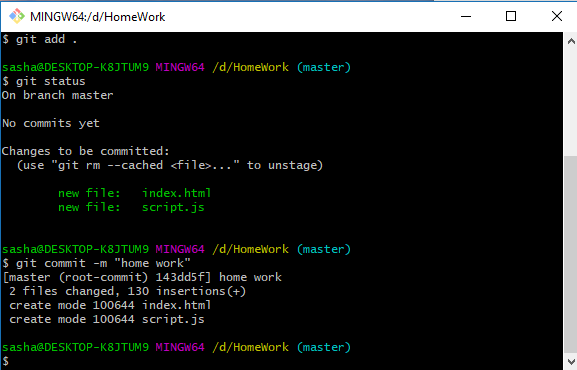


Рисунок 2.4 – Фіксування змін (коміт).

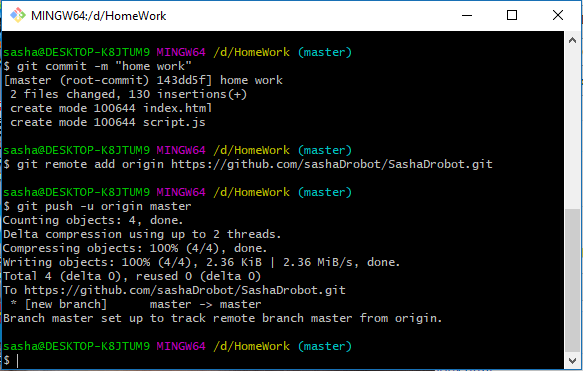


Рисунок 2.5 – Підключення до віддаленого репозиторію та закачування файлів на сервер.

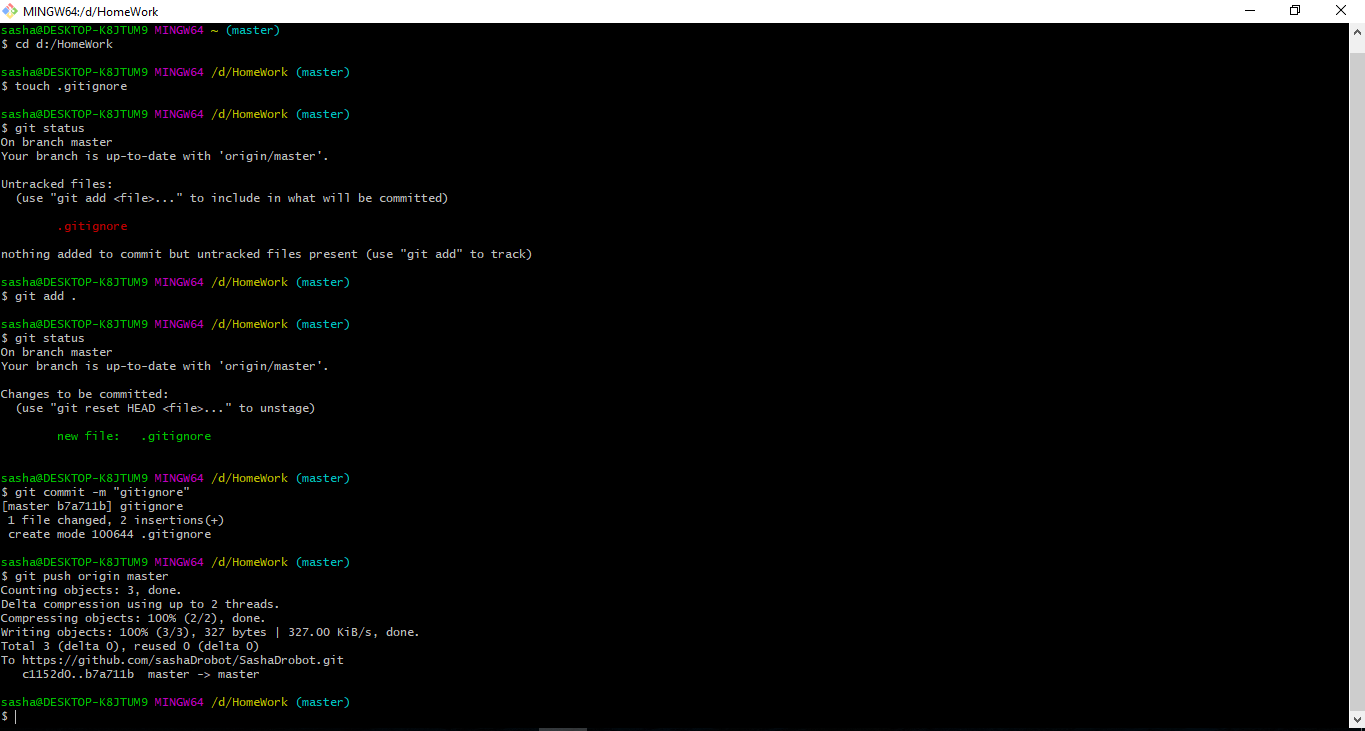


Рисунок 2.6 – Добавлення файлу .gitignore.

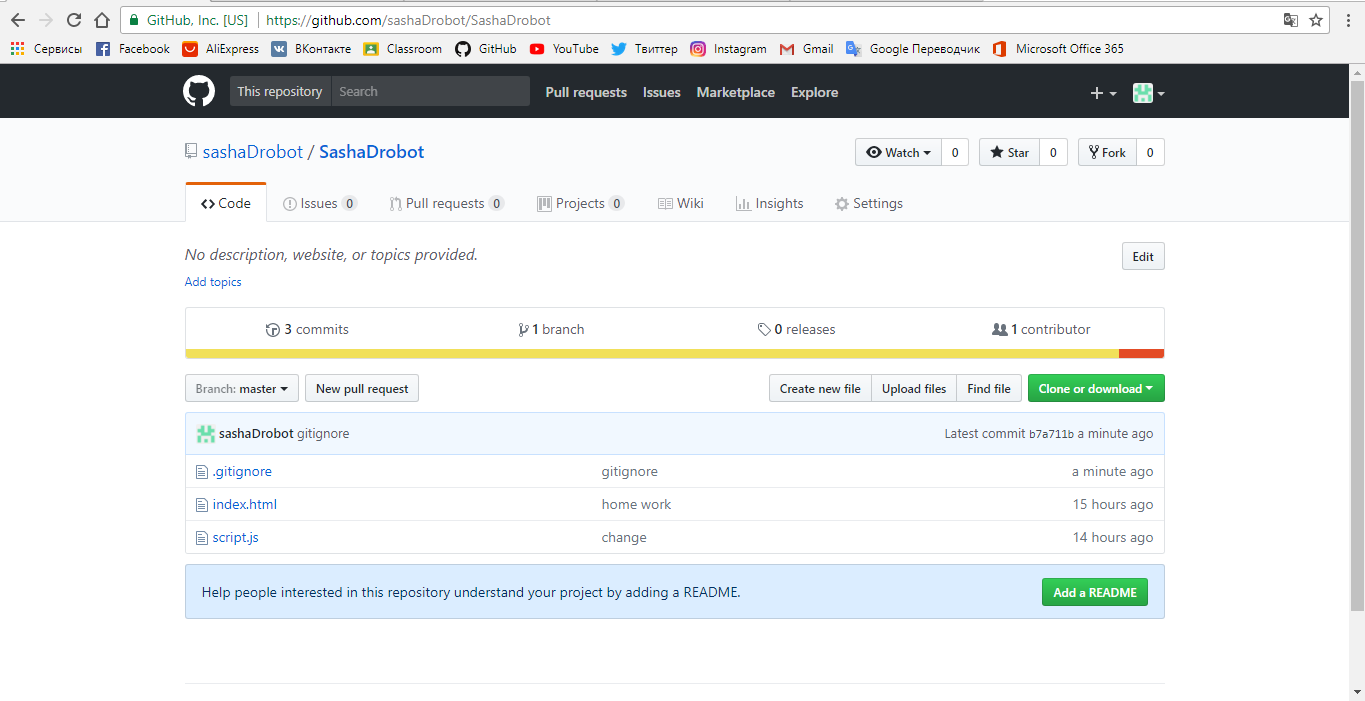


Рисунок 2.7 – Віддалений репозиторій з файлами проекту.

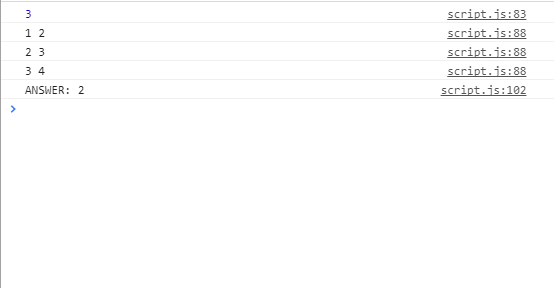


Рисунок 2.8 – Приклад роботи програми.

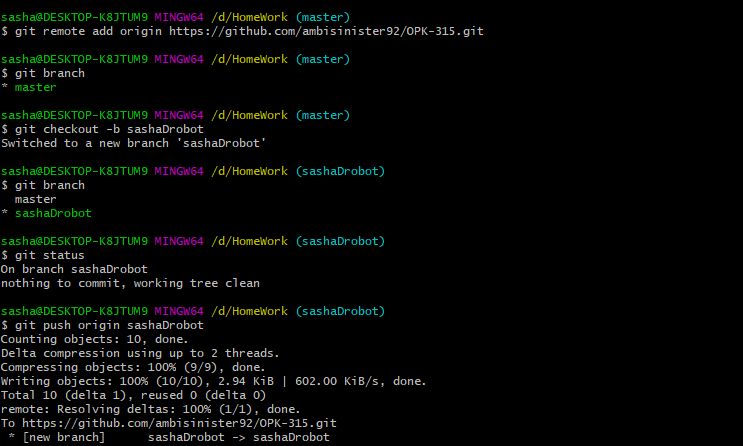


Рисунок 2.9 – Підключення до віддаленого репозиторію, створення гілки та відправлення проекту на неї.

Код програми: (JavaScript)

*// TV OBJECT***var** *TV* = **function** (li, ri) {  
 **this**.**li** = li;  
 **this**.**ri** = ri;  
 **this**.**on** = **true**;  
  
 **return this**;  
}  
  
*//функция которая принимает импрешшион - массив обьектов типа типа {li: n, ri: m}, по сути массив интервалов  
//функция преобразовывает этот массив соединяя интервалы друг с другом и выдает красивый массив интервалов по порядку и без повторов***function** *merge*(imp) {  
 *// в каждой итерации каждый элемент вбирает в себя интервалы следующих элементов  
 // т.е если в первой ячейке массива (1; 2), а во второй (2; 4), то при первой итерации в первой ячейке окажется (1;4)  
 //условия в этих циклах основное что надо понять: сначала проверка входит ли проверяемый элемент j в элемент i, то есть  
 // входит ли один из краев интервала j в интервал i, чтобы интервал i можно было расширить  
 // изменения записываются в элемент i, то есть по ходу работы циклов изменения будут накапливаться ближе к началу массива  
 //сначала проверяется находится интервал j левее интервала i (но все еще входит в него правой границой или касается края)  
 //потом то же самое для правой стороны* **var** flag = **true**;  
 **while** (flag)  
 {  
 flag = **false**;  
 **for** (**var** i = 0; i < (imp.**length** - 1); i++) {  
 **for** (**var** j = i + 1; j < imp.**length**; j++) {  
 **if** ((imp[j].**li** <= imp[i].**li**) && (imp[j].**ri** >= (imp[i].**li** - 1))) {  
 **if** (imp[i].**li** != imp[j].**li**) {  
 imp[i].**li** = imp[j].**li**;  
 flag = **true**;  
 }  
 }  
 **if** ((imp[j].**ri** >= imp[i].**ri**) && (imp[j].**li** <= (imp[i].**ri** + 1))) {  
 **if** (imp[i].**ri** != imp[j].**ri**) {  
 imp[i].**ri** = imp[j].**ri**;  
 flag = **true**;  
 }  
 }  
 }  
 }  
 }  
 *//этот цикл сортирует: интервалы с меньшей левой границей передвигаются к началу массива  
 //второе условие: если левые границы одинаковые, то ближе к началу становится тот, у которого больше правая граница* **for** (**var** i = 0; i < imp.**length**; i++) {  
 **for**(**var** j = i + 1; j < imp.**length**; j++) {  
 **if** ((imp[j].**li** < imp[i].**li**) || (imp[j].**li** === imp[i].**li** && imp[j].**ri** > imp[i].**ri**))  
 [imp[i], imp[j]] = [imp[j], imp[i]]; *//swap* }  
 }  
 *//удаление лишнего:  
 //этот цикл сравнивает один элемент i с каждым другим, и если находится элемент с более широким интервалом, то интервал i удаляется  
 // таким образом остается чистый красивый список интервалов* **for** (**var** i = 0; i < imp.**length**; i++) {  
 **loop**:  
 **for** (**var** j = 0; j < imp.**length**; j++) {  
 **if** (i!=j) {  
 **if** (imp[i].**li** >= imp[j].**li** && imp[i].**ri** <= imp[j].**ri**)  
 {  
 imp.splice(i, 1);  
 i--;  
 **break loop**;  
 }  
 }  
 }  
 }  
 **return** imp;  
}  
  
*//принимает массив телевизоров и на основе его данных строит выражение: массив интервалов времени  
//считаются только включенные тв***function** *makeImpression*(tvs) {  
 **var** impression = [];  
 tvs.forEach(**function** (item) {  
 **if** (item.**on**)  
 impression.push({**li**: item.**li**, **ri**: item.**ri**});  
 });  
 **return** impression;  
}  
*//возвращает true если массивы интервалов равны***function** *isEqual*(imp1, imp2) {  
 **if** (imp1.**length** != imp2.**length**)  
 **return false**;  
 **for** (**var** i = 0; i < imp1.**length**; i++) {  
 **if** ((imp1[i].**li** != imp2[i].**li**) || (imp1[i].**ri** != imp2[i].**ri**))  
 **return false**;  
 }  
 **return true**;  
}  
**var *TVs*** = []; *// массив телевизоров***var *n*** = +prompt(**"Enter a number of TVs:"**); *//ввод количества телевизоров, <2\*105  
//+ перем промтом - явное преобразование в числовое значение. нужно число, а промпт возвращает строку***while** (***n*** <= 0 || ***n*** > 109) {  
 ***n*** = +prompt(**"Number of TVs should be more then 0 and less then 110. Re-enter:"**);  
}  
**console**.log(***n***);  
**var *Li***, ***Ri***; *// < 109***for** (**var *i*** = 0; ***i*** < ***n***; ***i***++) {  
 ***Li*** = +prompt(**"Enter li for TV #"** + (***i*** + 1));  
 **while** (***Li*** < 0 || ***Li*** >= 2 \* ***Math***.pow(10, 5)) {*//проверка на соответствие ограничениям* ***Li*** = +prompt(**"Interval boundary should be >= 0 and < 2\*10^5. Re-enter li:"**);  
 }  
  
 ***Ri*** = +prompt(**"Enter ri for TV #"** + (***i*** + 1));  
 **while** (***Ri*** < 0 || ***Ri*** >= 2 \* ***Math***.pow(10, 5)) { *//проверка на соответствие ограничениям* ***Ri*** = +prompt(**"Interval boundary should be >= 0 and < 2\*10^5. Re-enter ri:"**);  
 }  
  
 **while** (***Ri*** <= ***Li***) {*//проверка на то чтобы левый интервал был меньше правого* ***Ri*** = +prompt(**"RI must be > then LI, re-enter ri for TV #"** + (***i*** + 1));  
 }  
  
 **console**.log(***Li*** + **" "** + ***Ri***);  
 ***TVs***.push(**new** *TV*(***Li***, ***Ri***));  
}  
  
**var *impression*** = *merge*(*makeImpression*(***TVs***)); *//создание массива интервалов***var *checking***;  
**var *noAnswer*** = **true**;  
*//цикл по очереди выключает каждый телевизор, и проверяем изменится ли интервал***cycle**:  
 **for** (**var *i*** = 0; ***i*** < ***TVs***.**length**; ***i***++)  
 {  
 ***TVs***[***i***].**on** = **false**;  
 ***checking*** = *merge*(*makeImpression*(***TVs***));  
 **if** (*isEqual*(***impression***, ***checking***)) {  
 **console**.log(**"ANSWER: "** + (***i***+1));  
 ***noAnswer*** = **false**;  
 *//break cycle;* }  
 ***TVs***[***i***].**on** = **true**;  
 }  
**if** (***noAnswer***) **console**.log(**"-1"**);

**Питання до захисту:**

1. **Чим є насправді гілки у Git?**

Гілка в Гіт це просто легкий двигаємий вказівник, що може пересуватися на одну з фіксацій (створених за допомогою git commit). Створення нової гілки створює новий вказівник, щоб можна було пересуватися. Особливий вказівник під назвою HEAD вказує на локальну гілку, на якій ви знаходитесь.

1. **Розкажіть про способи злиття у Git?**

Злиття буває двох типів: звичайне (трьохпозиційне) – коли у гілок спільний предок та нащадки зливаються в одну гілку; злиття-перемотка - якщо вітка, в яку відбувається злиття є прямим батьком вітки, яку зливають, то Git просто переміщає вказівник вперед, фізично нічого не відбувається.

1. **Що відбувається у робочій директорії при перемиканні гілок?**

При перемиканні гілки, файли у робочій папці змінюються. Якщо переключитися до старшої гілки, робоча папка буде повернута до того стану, який був на момент останнього фіксування у тій гілці. Якщо Гіт не зможе це зробити без проблем, він не дасть переключитися взагалі.

Роботу виконав: Дробот О.О. Роботу перевірив: Фесенко Д.В.