**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ХАРКІВСЬКИЙ КОМП′ЮТЕРНО-ТЕХНОЛОГІЧНИЙ**

**КОЛЕДЖ НТУ «ХПІ»**

**ЛАБОРАТОРНА РОБОТА №2**

з дисципліни

«Соціальні мережі»

Варіант №1

Виконав Перевірив

студент гр. ОПК-325 викладач

Сириця Владислав Фесенко Д.В.

Харків 2017

**ЛАБОРАТОРНА РОБОТА №2**

**«Основи галуження у Git. Робота з гілками»**

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

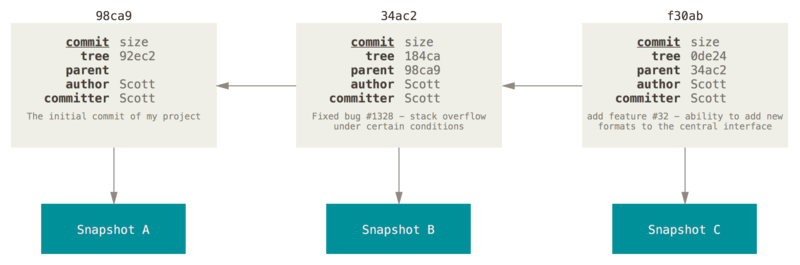
**2 . Мета роботи:** Опанувати навичками роботи з гілками.

**3. Теоретичний матеріал**

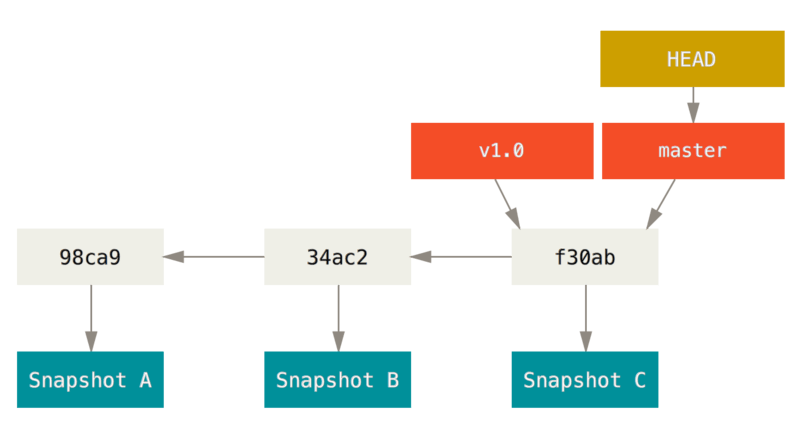
Майже кожна система контролю версій підтримує гілки (branches) в певній мірі. Галуження - це відмежування від основної лінії розробки для продовження своєї частини роботи та уникнення конфліктів з основною лінією. Коли ви фіксуєте зміни, Git зберігає об’єкт фіксації, що містить вказівник на знімок змісту, який ви додали. Цей об’єкт також містить ім’я та поштову адресу автора, набране вами повідомлення та вказівники на фіксацію або фіксації, що були прямо до цієї фіксації (батько чи батьки): нуль для першої фіксації, одна фіксація для нормальної фіксації, та декілька фіксацій для фіксацій, що вони є результатом злиття двох чи більше гілок.

Коли ви створили фіксацію за допомогою git commit, Git обчислив контрольну суму кожної теки та зберігає ці об’єкти дерева в сховищі Git. Потім Git створює об’єкт фіксації, що зберігає метадані та вказівник на корінь дерева проекту, щоб він міг відтворити цей знімок, коли потрібно.

Якщо ви зробите якісь зміни та зафіксуєте знову, наступна фіксація буде зберігати вказівник на попередню.

 Рисунок 2.1 – Фіксації та їх батьки

Гілка в Git це просто легкий вказівник, що може пересуватись, на одну з цих фіксацій. Коли ви почнете робити фіксації, вам надається гілка master, що вказує на останню зроблену фіксацію. Щоразу ви фіксуєте, вона переміщується вперед автоматично.

 Рисунок 2.2 – Гілка та її історія фіксацій

Створення нової гілки створює новий вказівник, щоб ви могли пересуватися. Це робиться за допомогою команди git branch.

Особливий вказівник під назвою HEAD вказує на локальну гілку, на якій ви знаходитесь.

Щоб переключитися на існуючу гілку, треба виконати команду git checkout.

Важливо зауважити, що коли ви переключаєте гілки в Git, файли у вашій робочій теці змінюються. Якщо ви переключаєтесь до старшої гілки, ваша робоча тека буде повернута до того стану, який був на момент останнього фіксування у тій гілці. Якщо Git не може зробити це без проблем, він не дасть вам переключитися взагалі.

Оскільки гілки в Git це насправді простий файл, що містить 50 символів контрольної суми SHA-1 фіксації, на яку вказує, гілки дешево створювати та знищувати. Створити гілку так же швидко, як записати 41 байт до файлу(40 символів та символ нового рядку).

Основні команди для роботи з гілками у Git:

git branch – вивід списку існуючих гілок;

git branch -v – вивід списку існуючих гілок та їх останніх коммітів;

git branch -vv – вивід списку існуючих гілок, що відстежуються;

git branch --merged – вивід списку гілок, що злиті із поточною;

git branch --no-merged – вивід списку гілок, що не злиті із поточною;

git branch -u [remote]/[remotebranch] – прив’язка поточної локальної гілки до віддаленої;

git branch [branchname] – створення гілки;

git branch -d [branchname] – видалення гілки;

git branch -D [branchname] – форсоване видалення гілки;

git checkout [branchname] – переключення на гілку;

git checkout -b [branchname] – створення гілки та переключення на неї;

git checkout -b [branchname] [remote]/[remotebranch] – створення гілки, що відстежує віддалену гілку(з можливістю зміни імені локальної гілки);

git checkout --track [remote]/[remotebranch] – створення гілки, що відстежує віддалену гілку;

git merge [branchname] – злиття заданої гілки з поточною;

git push [remote] [localbranch] – відправлення локальної гілки на сервер;

git push --delete [remotebranch] – видалення гілки з сервера;

git rebase [branch] – перебазування поточної гілки на задану;

git rebase [basebranch] [topicbranch] – перебазування гілок;

git pull --rebase – автоматичне перебазування при злитті.

**4. Вихідні дані роботи**

Варіант 1: Team1

Варіант 2: Team2

Варіант 3: Team3

**5. Хід виконання практичної роботи**

Робота проводиться на операційній системі Windows 7 Ultimate x64. На сам перед, треба зареєструватись на GitHub. Далi у пошуку меню «Пуск» знайти програму *Git Bash* та запустити її.

1. У D:\network створити репозиторій, або перейти до вже існуючого.

2. Підключити віддалений репозиторій (https://github.com/ambisinister92/laba2).

3. Отримати данні з віддаленого сховища та злити їх з вашою роботою.

4. Створити нову гілку(ім’я гілки згідно з варіантом), що відгалужується від гілки laba2/dev.

5. Внести зміни до створеної гілки(під змінами мається на увазі обов’язкове створення файлу, ім’я якого містить прізвище та групу студента, та не обов’язкова зміна вже існуючих файлів).

6. Настроїти локальну гілку dev так, щоб вона відстежувала віддалену laba2/dev.

7. Відправити зміни на сервер.

Задача:

#include<vector>

#include<cstdio>

#include<cstring>

#include<iostream>

#include<algorithm>

using namespace std;

#define N 500010

#define M 20

#define ll long long

ll Ans[N];

vector<int>g[N],e[N];

int i,j,k,n,m,p,l;

int a[N],b[N],c[N];

int x,Rt,d[N],f[N][M];

inline void Dfs(int x){

g[d[x]].push\_back(x);

for(int i=0;i<e[x].size();i++)

d[e[x][i]]=d[x]+1,Dfs(e[x][i]);

}

inline int Lca(int x,int y){

if(d[x]<d[y])swap(x,y);

for(int i=M-1;i>=0;i--)if(d[f[x][i]]>=d[y])x=f[x][i];

if(x==y)return x;

for(int i=M-1;i>=0;i--)if(f[x][i]!=f[y][i])x=f[x][i],y=f[y][i];

return f[x][0];

}

inline void Solve(){

ll sum=0;

for(int j=0;j<g[i].size();j++){

int x=g[i][j];

if(!j)a[l=1]=x,b[1]=c[1]=0;else{

while(1){

int L=Lca(a[l],x);

if(b[l]<=d[L]){

a[++l]=x;b[l]=d[L]+1;c[l]=j;

break;

}

sum-=1ll\*(c[l]-c[l-1])\*b[l];

l--;

}

sum+=1ll\*(c[l]-c[l-1])\*b[l];

}

Ans[x]+=sum;

}

}

int main(){

scanf("%d",&n);

for(i=1;i<=n;i++){

scanf("%d",&x);

if(!x)Rt=i,f[i][0]=i;else f[i][0]=x,e[x].push\_back(i);

}

Dfs(Rt);

for(j=1;j<M;j++)

for(i=1;i<=n;i++)

f[i][j]=f[f[i][j-1]][j-1];

for(i=1;i<n;i++){

if(!g[i].size())break;

for(j=0;j<g[i].size();j++)Ans[g[i][j]]+=Ans[f[g[i][j]][0]]+i;

Solve();reverse(g[i].begin(),g[i].end());Solve();

}

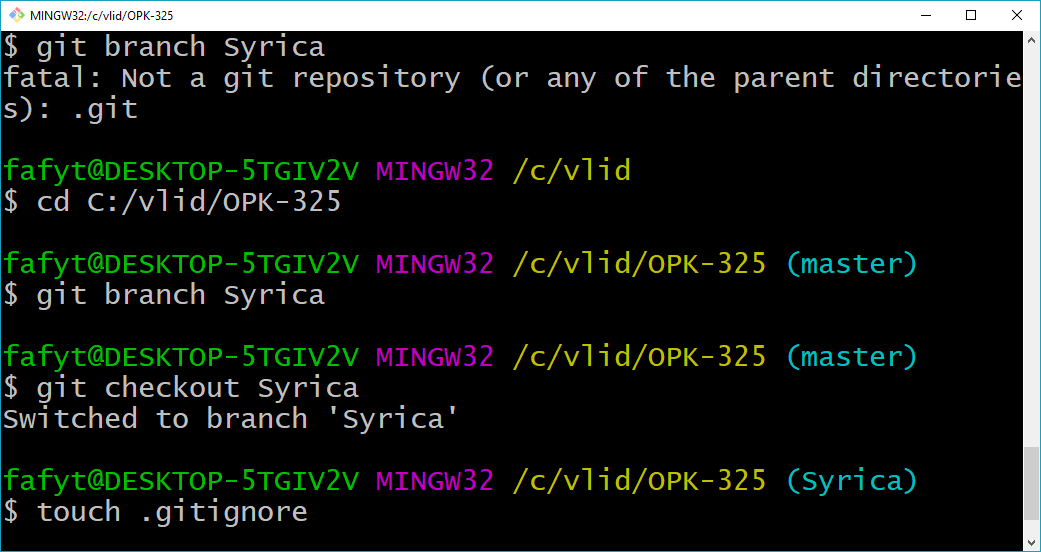
for(i=1;i<=n;i++)printf("%I64d ",Ans[i]);

return 0;

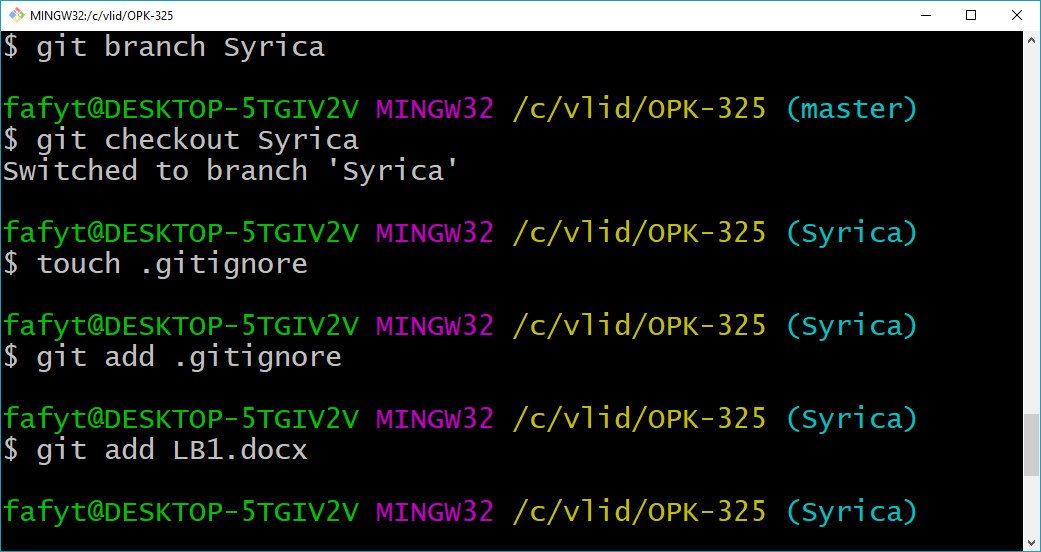
}



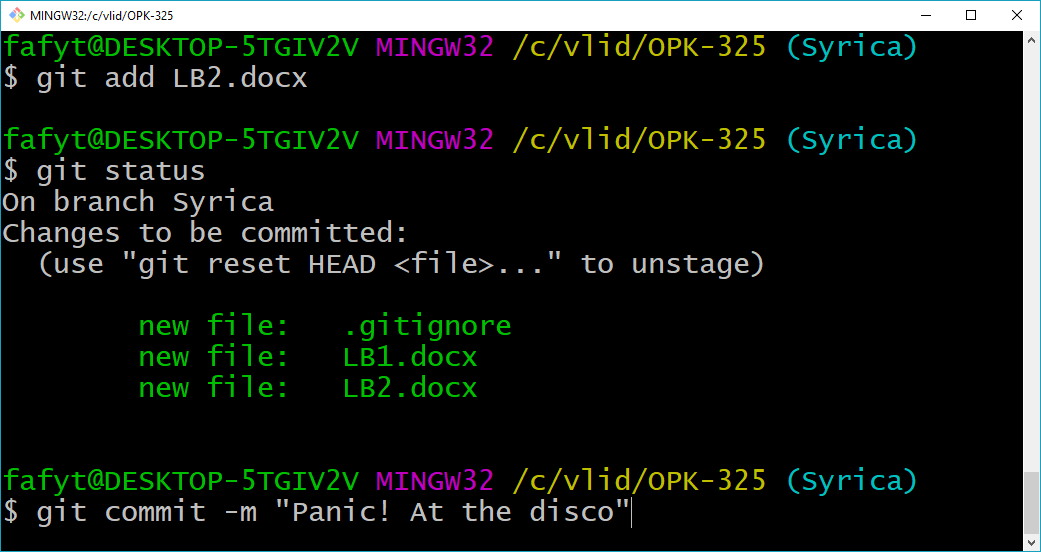
Клонував репозиторiй у створену папку.



Створив свою гiлку та прейшов неї. Далі створив файл gitignore.



Проіндексував зміни.



Відправив файли на сервер.

**Питання до захисту :**

**Чим на є насправді гілки у Git?**

Відповідь: Гілка в Git це просто легкий вказівник, що може пересуватись, на одну з цих фіксацій. Коли ви почнете робити фіксації, вам надається гілка master, що вказує на останню зроблену фіксацію. Щоразу ви фіксуєте, вона переміщується вперед автоматично.

**Розкажіть про способи злиття у Git?**

Відповідь:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Відповідь: Коли ви переключаєте гілки в Git, файли у вашій робочій теці змінюються. Якщо ви переключаєтесь до старшої гілки, ваша робоча тека буде повернута до того стану, який був на момент останнього фіксування у тій гілці. Якщо Git не може зробити це без проблем, він не дасть вам переключитися взагалі.

Роботу виконав Роботу перевірив

Сириця В.О. Фесенко Д.В.