МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ КОМП’ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра

комп’ютерних наук

Комплексне практичне індивідуальне завдання

з дисципліни

«Конструювання програмного забезпечення»

**Виконав:**

студент групи ПЗАС-32

Петрик А. І.

**Перевірив:**

викладач

Сирник О. Й.

Тернопіль

2015

**Вступ**

Контроль версій - це система, котра веде протокол змін в файлі або кількох файлах увесь час для того, щоб ви мали можливість викликати певні версії пізніше. Незважаючи на те, що в якості прикладів файлів під контролем версій в цій книзі використовується вихідний код програмного забезпечення, насправді, будь-який тип файлів на ПК може бути розташований під контролем версій.

Якщо ви графічний або веб-дизайнер і ви хочете зберегти кожну версію малюнку або шару (, що безсумнівно вам хочеться,), то використання системи контролю версій (надалі - СКВ) буде дуже розсудливим в цьому випадку. СКВ надає можливість робити вам наступне: повертати файли до попереднього вигляду, повертати цілий проект до попереднього вигляду, переглядати зміни, зроблені за увесь час, дивитися, хто востаннє змінював щось, що могло призвести до проблеми, хто вирішив задачу та коли й багато іншого. Використання СКВ також означає що, якщо ви щось зламаєте або загубите файли, ви зможете легко усе відновити. До того ж, ви все це отримуєте за дуже маленькі накладні витрати.

1. **Що таке Git?**

Так що ж таке Git? Git це система контролю версій. Головна відмінність Git від будь-яких інших СКВ (наприклад, Subversion і їй подібних) - це те, як Git дивиться на свої дані. У принципі, більшість інших систем зберігає інформацію як список змін (патчів) для файлів. Ці системи (CVS, Subversion, Perforce, Bazaar та інші) відносяться до збережених даних як до набору файлів і змін, зроблених для кожного з цих файлів в часі, як показано на рис. 1.1.



Рис. 1.1. Інші системи зберігають дані як зміни до базової версії для кожного файлу.

Git не зберігає свої дані в такому вигляді. Замість цього Git вважає збережені дані набором зліпків невеликої файлової системи. Кожен раз, коли ви фіксуєте поточну версію проекту, Git, по суті, зберігає зліпок того, як виглядають всі файли проекту на поточний момент. Заради ефективності, якщо файл не змінювався, Git не зберігається файл знову, а робить посилання на раніше збережений файл. Те, як Git підходить до зберігання даних, схоже на рис. 1.2.



Рис. 1.2 Git зберігає дані як зліпки станів проекту в часі.

Це важлива відмінність Git від практично всіх інших систем контролю версій. Через нього Git змушений переглянути практично всі аспекти контролю версій, які інші системи перейняли від своїх попередниць. Git більше схожий на невелику файлову систему з неймовірно потужними інструментами, що працюють поверх неї, ніж на просто ВКВ.

1. **Створення Git-репозиторію**

Для створення Git-репозиторію існують два основні підходи. Перший підхід - імпорт в Git вже існуючого проекту або каталогу. Другий - клонування вже існуючого репозиторію з сервера.

Для того, щоб почати використовувати Git для існуючого проекту, то вам необхідно перейти в проектний каталог і в командному рядку ввести

$ git init

Ця команда створює в поточному каталозі новий підкаталог з ім'ям .git містить всі необхідні файли репозиторію - основу Git-репозиторію.

Щоб додати під версійній контроль існуючі файли (на відміну від порожнього каталогу), вам варто проіндексувати ці файли і здійснити перші фіксацію змін. Здійснити це ви можете за допомогою декількох команд git add вказують індексовані файли, а потім commit:

$ git add \*.c

$ git add README

$ git commit -m 'initial project version'

Якщо ви бажаєте отримати копію існуючого репозиторію Git, наприклад, проекту, в якому ви хочете взяти участь, то вам потрібна команда git clone. Якщо ви знайомі з іншими системами контролю версій, такими як Subversion, то помітите, що команда називається clone, а не checkout. Це важлива відмінність - Git отримує копію практично всіх даних, що є на сервері. Кожна версія кожного файлу з історії проекту забирається (pulled) з сервера, коли ви виконуєте git clone. Фактично, якщо серверний диск вийде з ладу, ви можете використовувати будь-який з клонів на будь-якому з клієнтів, для того щоб повернути сервер в той стан, в якому він перебував у момент клонування (ви можете втратити частину серверних перехоплювачів (server-side hooks) і т.п., але всі дані, поміщені під версійність контроль, будуть збережені.

Клонування репозиторію здійснюється командою git clone. Наприклад, якщо ви хочете клонувати бібліотеку Ruby Git, відому як Grit, ви можете зробити це таким чином:

$ git clone git://github.com/schacon/grit.git

Ця команда створює каталог з ім'ям grit, ініціалізує в ньому каталог .git, викачує всі дані для цього репозиторію і створює (checks out) робочу копію останньої версії. Якщо ви зайдете в новий каталог grit, ви побачите в ньому проектні файли, придатні для роботи і використання.

1. **Робота зі змінами**

Отже, у вас є справжній Git-репозиторій і робоча копія файлів для деякого проекту. Вам потрібно робити деякі зміни і фіксувати "знімки" стану (snapshots) цих змін у вашому репозиторії кожен раз, коли проект досягає стану, який вам хотілося б зберегти.

Запам'ятайте, кожен файл у вашому робочому каталозі може перебувати в одному з двох станів: під версійність контролем (відстежувані) і ні (невідстежувані). Відстежувані файли - це ті файли, які були в останньому зліпку стану проекту (snapshot); вони можуть бути незміненими, зміненими або підготовленими до коммітов (staged). Невідстежувані файли - це все інше, будь-які файли у вашому робочому каталозі, які не входили до вашого останнього зліпку стану і не підготовлені до коммітов. Коли ви вперше клонуєте репозиторій, всі файли будуть відстежувати і незміненими, тому що ви тільки взяли їх зі сховища (checked them out) і нічого поки не редагували.

Як тільки ви відредагуєте файли, Git буде розглядати їх як змінені, тому ви змінили їх з моменту останнього коммітов. Ви індексуєте (stage) ці зміни і потім фіксуєте все індексовані зміни, а потім цикл повторюється.

Тепер ви можете робити зміни у проекті. Коли ви будете вважати, що проект потрібно зберегти, вам потрібно використати команду

$ git commit –m “message”

Замість “message” слід ввести повідомлення яке описує зміни, які ви зробили, або завдання, яке ви виконали. Після цього цей комміт можна надіслати до віддаленого репозиторію, тобто зробити push, за допомогою команди

$ git push

1. **Лістинги програми**

Нижче приведені лістинги програми для імітації роботи мікрохвильової печі

Файл «Form1.h»:

#include "MicroVaveOven.h"

#include <Windows.h>

#include <stdio.h>

namespace KPIZ\_OOP\_PE3K {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

/// <summary>

/// Summary for Form1

/// </summary>

public ref class Form1 : public System::Windows::Forms::Form

{

public:

MicroVaveOven oven;

Form1(void)

{

InitializeComponent();

//

//TODO: Add the constructor code here

//

}

protected:

/// <summary>

/// Clean up any resources being used.

/// </summary>

~Form1()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::TextBox^ textBox2;

private: System::Windows::Forms::Button^ button1;

private: System::Windows::Forms::TextBox^ textBox1;

private: System::Windows::Forms::Timer^ timer1;

private: System::ComponentModel::IContainer^ components;

protected:

private:

/// <summary>

/// Required designer variable.

/// </summary>

#pragma region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

void InitializeComponent(void)

{

this->components = (gcnew System::ComponentModel::Container());

this->textBox2 = (gcnew System::Windows::Forms::TextBox());

this->button1 = (gcnew System::Windows::Forms::Button());

this->textBox1 = (gcnew System::Windows::Forms::TextBox());

this->timer1 = (gcnew System::Windows::Forms::Timer(this->components));

this->SuspendLayout();

//

// textBox2

//

this->textBox2->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Right));

this->textBox2->Font = (gcnew System::Drawing::Font(L"Comic Sans MS", 8.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->textBox2->Location = System::Drawing::Point(4, 415);

this->textBox2->MaximumSize = System::Drawing::Size(673, 20);

this->textBox2->MinimumSize = System::Drawing::Size(673, 20);

this->textBox2->Name = L"textBox2";

this->textBox2->Size = System::Drawing::Size(673, 23);

this->textBox2->TabIndex = 0;

this->textBox2->KeyDown += gcnew System::Windows::Forms::KeyEventHandler(this, &Form1::textBox2\_KeyDown);

//

// button1

//

this->button1->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Bottom | System::Windows::Forms::AnchorStyles::Right));

this->button1->Font = (gcnew System::Drawing::Font(L"Comic Sans MS", 8.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->button1->Location = System::Drawing::Point(683, 413);

this->button1->MaximumSize = System::Drawing::Size(114, 23);

this->button1->MinimumSize = System::Drawing::Size(114, 23);

this->button1->Name = L"button1";

this->button1->Size = System::Drawing::Size(114, 23);

this->button1->TabIndex = 1;

this->button1->Text = L"Enter";

this->button1->UseVisualStyleBackColor = true;

this->button1->Click += gcnew System::EventHandler(this, &Form1::button1\_Click);

//

// textBox1

//

this->textBox1->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((((System::Windows::Forms::AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Bottom)

| System::Windows::Forms::AnchorStyles::Left)

| System::Windows::Forms::AnchorStyles::Right));

this->textBox1->Font = (gcnew System::Drawing::Font(L"Comic Sans MS", 8.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->textBox1->Location = System::Drawing::Point(1, 1);

this->textBox1->Multiline = true;

this->textBox1->Name = L"textBox1";

this->textBox1->ReadOnly = true;

this->textBox1->ScrollBars = System::Windows::Forms::ScrollBars::Vertical;

this->textBox1->Size = System::Drawing::Size(796, 411);

this->textBox1->TabIndex = 2;

//

// timer1

//

this->timer1->Interval = 1000;

this->timer1->Tick += gcnew System::EventHandler(this, &Form1::timer1\_Tick);

//

// Form1

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(798, 439);

this->Controls->Add(this->button1);

this->Controls->Add(this->textBox2);

this->Controls->Add(this->textBox1);

this->MaximumSize = System::Drawing::Size(814, 477);

this->MinimumSize = System::Drawing::Size(814, 477);

this->Name = L"Form1";

this->Text = L"MicroVaveOven Simulator v1.0";

this->Load += gcnew System::EventHandler(this, &Form1::Form1\_Load);

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

private: void Add(System::String ^text){

this->textBox1->AppendText(text + "\n");

}

private: System::Void timer1\_Tick(System::Object^ sender, System::EventArgs^ e) {

if(oven.TimeLeft())

{

this-> oven.setTime(oven.TimeLeft() - 1);

Add(System::Convert::ToString(oven.TimeLeft()) + " seconds left");

}

else

{

Add("Now You can take Your food!");

timer1->Stop();

}

}

private: System::Void Form1\_Load(System::Object^ sender, System::EventArgs^ e) {

Add("MicrovaveOven Simulator v0.1");

Add("© Pe3k Andriy");

Add("Type \"help\" for more info");

}

private: System::Void textBox2\_KeyDown(System::Object^ sender, System::Windows::Forms::KeyEventArgs^ e) {

if(e->KeyValue == 13)

{

if(!System::String::IsNullOrEmpty(textBox2->Text) && !System::String::IsNullOrWhiteSpace(textBox2->Text))

{

Add(textBox2->Text);

array<System::String ^>^ strs = gcnew array<System::String ^>(3);

String^ delimStr = " ";

array<Char>^ delimiter = delimStr->ToCharArray( );

System::String ^str = gcnew System::String(textBox2->Text);

strs = str->Split(delimiter);

if(strs[0]->ToLower() == "help")

{

Add("Commands:");

Add(" clear -- clear the console");

Add(" start -- launch the oven");

Add(" stop -- stop the oven");

Add(" temperature <temperature> -- set the temperature of heating");

Add(" time <time> -- set the time of heating");

Add(" quit -- exit the program");

Add(" help -- view this help window");

this->textBox2->Text = "";

return;

}

if(strs[0]->ToLower() == "clear")

{

textBox1->Text = "";

Add("Cosole cleared!");

this->textBox2->Text = "";

return;

}

if(strs[0]->ToLower() == "start")

{

if(oven.TimeLeft() > 0)

{

if(!oven.IsRunning())

{

oven.Start(timer1);

}

}

else

{

Add("Please, set the time using command \"time <time>\"");

}

this->textBox2->Text = "";

return;

}

if(strs[0]->ToLower() == "stop")

{

if(oven.IsRunning())

{

oven.Stop(timer1);

}

this->textBox2->Text = "";

return;

}

if(strs[0]->ToLower() == "time")

{

try

{

if(!System::String::IsNullOrEmpty(strs[1]))

{

if(System::Convert::ToInt32(strs[1]) > 0)

{

oven.setTime(System::Convert::ToInt32(strs[1]));

Add("Heating time is now " + strs[1] + " seconds");

this->textBox2->Text = "";

return;

}

else

{

Add("Invalid input data");

this->textBox2->Text = "";

return;

}

}

}

catch(System::Exception ^ex)

{

Add("Time: " + oven.TimeLeft());

this->textBox2->Text = "";

}

return;

}

if(strs[0]->ToLower() == "temperature")

{

try

{

if(System::Convert::ToInt32(strs[1]) > 0)

{

if(!System::String::IsNullOrEmpty(strs[1]))

{

oven.setTemperature(System::Convert::ToInt32(strs[1]));

Add("Temperature: " + strs[1]);

this->textBox2->Text = "";

return;

}

}

else

{

Add("Invalid input data");

this->textBox2->Text = "";

return;

}

}

catch(System::Exception ^ex)

{

Add("Temperature: " + oven.getTemperature());

this->textBox2->Text = "";

return;

}

}

if(strs[0]->ToLower() == "quit")

{

System::Windows::Forms::Application::Exit();

}

Add(strs[0] + ": unknow command");

}

textBox2->Text = "";

}

}

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

textBox2\_KeyDown(gcnew System::Object(), gcnew System::Windows::Forms::KeyEventArgs(System::Windows::Forms::Keys::Enter));

}

};

}

Файл «MicroVaveOven.h»:

ref class MicroVaveOven

{

private:

int temperature, timeLeft;

bool isClosed, isRunning;

public:

int getTemperature();

void setTemperature(int);

bool IsRunning();

int TimeLeft();

void setTime(int);

void Start(System::Windows::Forms::Timer ^timer);

void Stop(System::Windows::Forms::Timer ^timer);

MicroVaveOven(void);

MicroVaveOven(int, int);

MicroVaveOven(int, int, bool);

~MicroVaveOven(void);

};

**Висновки**

Під час виконання даного комплексного практичного індивідуального завдання я закріпив свої навички в констроюванні програмног забезпечення, а також попрактикувався в користанні системою контролю версій, а саме Git, навчився створювати власний репозиторій, додавати та робити коміти змін, а також додавати їх на віддалений репозиторій.