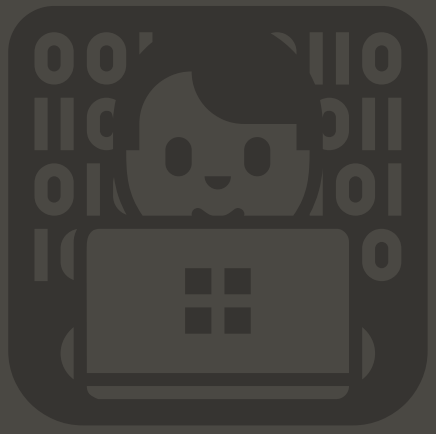


Professional Programming

Nástroje programátora

Roderik Ploszek

1.4.2019

Význam tejto prednášky

- predstavenie tém, ktoré úzko súvisia s programovaním a v budúcnosti ich budete používať
- za 100 minút sa nenaučíte nástroje dokonale ovládať, ale budete ich poznať a vedieť, kam sa pozrieť pre viac informácií

Obsah

Programovacie prostredie

- Jazyk
  - prvá časť orientovaná na C, zvyšok aplikovateľný na ľubovoľný jazyk
- Systém
  - orientované na UNIX
  - Windows 10 obsahuje Linux subsystém, UNIX prostredie sa dá tiež získať cez projekty MSYS2, Cygwin a MinGW

.h

časť prvá

modularizácia kódu

Motivácia

- na malý projekt stačí jeden `*.c` súbor
- pri väčších projektoch by to bolo neprehľadné

Riešenie

- Rozdeliť kód projektu do logických celkov – **modulov**
- Funkcie, ktoré spolu súvisia budú v jednom zdrojovom súbore
- *Ako zavolať funkciu z iného súboru?*

Princíp modularizácie I

- Kód modulu pripraví jeho implementátor (***.c**)
- Užívateľov jeho kódu zaujíma, čo funkcia robí, aké má vstupy (argumenty) a aké má výstupy (návratová hodnota) – teda **rozhranie** – to je definované v hlavičkovom súbore (***.h**)
  - Užívatelia nepotrebujú vedieť **ako** je funkcia implementovaná

Hlavičkové súbory

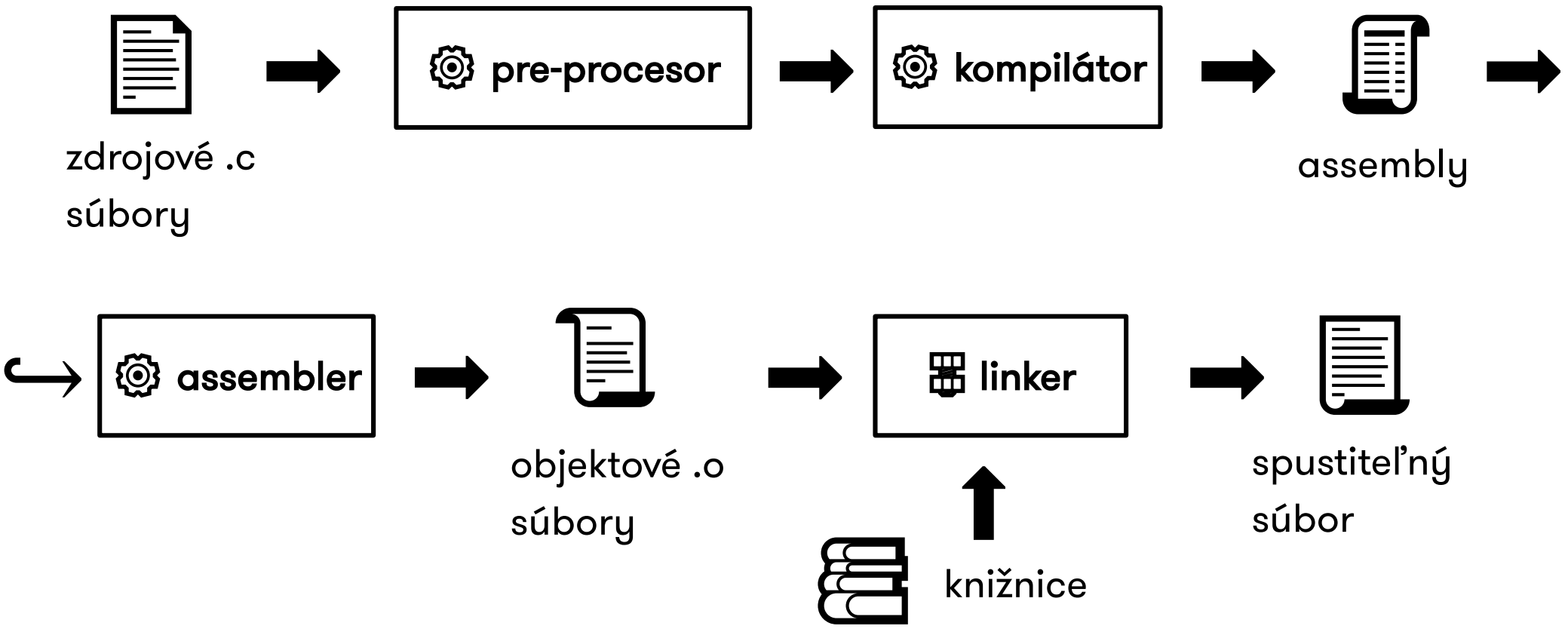
- Implementácie funkcií v **`*.c`** súbore
- Deklarácie funkcií (prototypy) v **`*.h`** súbore
- Pre zavolanie funkcií z iného súboru ich najprv deklarujem direktívou **`#include`**
  - **Preprocesor** na miesto **`#include`** vloží obsah odkazovaného súboru

Zahrnutie súborov

- **`#include <stdio.h>`**
  - hľadá hlavičkový súbor v systémových priečinkoch
- **`#include "todo.h"`**
  - hľadá hlavičkový súbor v aktuálnom priečinku

bližšie vysvetlenie v manuáli GCC

Kompilácia


pre-procesor
kompilátor
zdrojové .c súbory
assembly
assembler
objektové .o súbory
linker
knižnice
spustiteľný súbor

Princíp modularizácie II

- Nie všetky funkcie z `*.c` súborov je nutné prepísať do hlavičkových súborov
- V hlavičkových súboroch sú deklarované funkcie, ktoré umožňuje autor modulu volať (*verejné funkcie*)
- Interné funkcie **pre potreby modulu** sa nedeklarujú v hlavičkových súboroch – užitočné je označiť ich kľúčovým slovom **`static`**
  - také funkcie nie je možné volať z iných súborov (sú *privátne*)

Ukážkový projekt

- V prezentácii budeme pracovať s malým programom na správu todo zoznamov
- Zdrojový kód je dostupný na github.com/Programator2/ctodo

Využitie modularizácie v ctodo

- **`date.c`** práca s dátumami
- **`file.c`** práca so súbormi
- **`list.c`** podpora zreťazených zoznamov
- **`main.c`** obsahuje hlavnú (main) funkciu
- **`todo.c`** manupulácia s todo zoznamami
- **`tui.c`** textové užívateľské rozhranie

každý **`.c`** súbor má svoj hlavičkový súbor, v ktorom definuje svoje **rozhranie** pre iné zdrojové súbory

do hlavičkových súborov *neuvádzame* funkcie pre interné použitie v module

Verejné vs. privátne funkcie

 file.c práca so súbormi

```
FILE* open_todo_file(void);
char* read_line(FILE* file);
```

funkcie užitočné pre iné moduly, t.j. **verejné funkcie** sú uvedené v hlavičkovom súbore

```
static char* _check_buffer(void);
```

Interná funkcia používaná iba vo **`file.c`**. Nechceme, aby k nej mali iné moduly prístup – kľúčové slovo **`static`**. Taktiež viď. podtržník na začiatku – dohoda pre označovanie interných funkcií.

Kompilácia

- **`gcc -Wall -Wextra date.c file.c list.c main.c todo.c tui.c -o todo`**
- Čo keď bude v projekte oveľa viac súborov?
  - Je nepraktické všetky písať do príkazového riadku

make

časť druhá

zostavovanie programov

Motivácia

- Kompilácia veľkých projektov trvá dlho
- Programátor často upravuje len niekoľko súborov – stačí skompilovať iba tie
- Chceme jednoduchú kompiláciu jedným príkazom bez vypisovania všetkých súborov

Riešenie

- zostaviť iba súbory, ktoré boli zmenené
- použiť zostavovací systém

Použitie make

- **`Makefile`** – súbor, v ktorom sú popísané **závislosti** medzi súbormi a **príkazy** na ich *kompiláciu*
- po aktualizácii súborov stačí spustiť príkaz **`make`** a kompilácia bude vykonaná automaticky podľa stavu súborov

Štruktúra Makefile

- **`Makefile`** pozostáva z pravidiel vo formáte:

```
TARGET ... : PREREQUISITES ...
    RECIPE
    ...
    ...
```

- **Target** – **názov súboru** generovaný programom (napr. spustiteľné alebo objektové súbory alebo **názov akcie**)
- **Prerequisite** – súbory alebo akcie, od ktorých pravidlo závisí
- **Recipe** – zoznam príkazov, ktoré **`make`** vykoná

Jednoduchý Makefile

```
todo: date.o file.o list.o main.o todo.o tui.o
    gcc -Wall -Wextra date.o file.o list.o main.o \
        todo.o tui.o -o todo
```

Čo keď aktualizujeme hlavičkový súbor?

Pokročilejší Makefile

```
todo: date.o file.o list.o main.o todo.o tui.o
    gcc -Wall -Wextra date.o file.o list.o main.o \
        todo.o tui.o -o todo
date.o: date.c
file.o: file.c
list.o: list.c todo.h
main.o: main.c todo.h tui.h
todo.o: todo.c date.h file.h list.h todo.h
tui.o: tui.c file.h todo.h
```

Makefile s premennými

```
CC = gcc
CFLAGS = -Wall -Wextra
OBJ = date.o file.o list.o main.o todo.o tui.o

todo: $(OBJ)
    $(CC) $(CFLAGS) $(OBJ) -o todo

date.o: date.c
file.o: file.c
list.o: list.c todo.h
list.o: list.c todo.h
main.o: main.c todo.h tui.h
todo.o: todo.c date.h file.h list.h todo.h
tui.o: tui.c file.h todo.h
```

kompilátor

prepínače pre kompilátor

zoznam objektových súborov

Pravidlo clean

zmaže všetky objektové súbory:

```
.PHONY: clean
clean:
	rm todo $(OBJ)
```

make

- využitie make nie je limitované len na C programy
- využiteľné na automatickú aktualizáciu súborov, ak boli iné súbory zmenené

  make manuál

  iné zostavovacie nástroje

git

časť tretia

správa verzií

Motivácia

- do projektu chcete pridať novú funkciu, čo bude vyžadovať prepísanie veľkého množstva kódu
- po úprave kódu a uložení potrebujete zistiť, ako kód vyzeral pred vašou úpravou
  - *Ako to zistíte?*

Nesprávne riešenie

- vytvoríte zálohu **`todo_old.c`**
- pri ďalšej úprave **`todo_old_2.c`**
  - *neudržuje históriu – ktorý je novší?*
  - *neprehľadný obsah priečinka so zdrojákmi*

Riešenie

- systém na **správu verzií**
- ukladá **históriu** projektu – je možné pozrieť/vrátiť sa do predchádzajúcej verzie
- udržuje viac verzií projektu súčasne
- umožňuje tímovú spoluprácu

Systémy na správu verzií

- Git
- Mercurial
- Subversion

Git

- v súčasnosti najpoužívanejší VCS
- vytvorený z núdze Linusom Torvaldsom v roku 2005 na správu kódu Linuxu
- Využíva ho 87% vývojárov ako hlavný VCS nástroj*, Microsoft v ňom má celú Windows infraštruktúru
- Online repozitáre: GitHub, GitLab, SourceForge

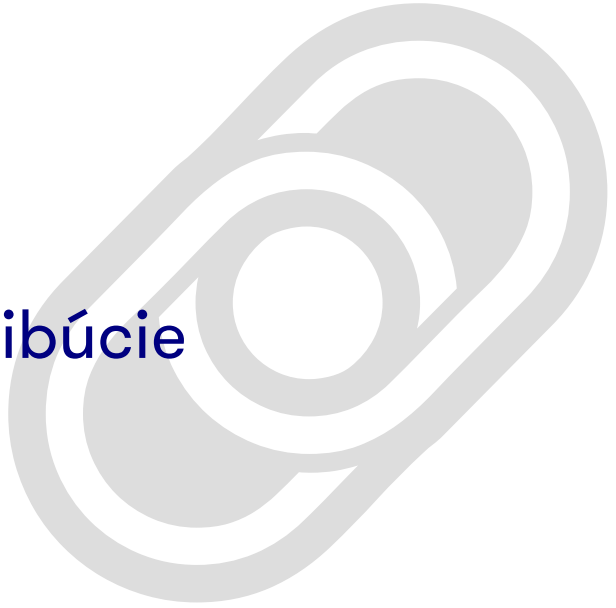
** 2018 StackOverflow developer survey*

Prečo git?

- distribuovaný systém – každý vývojár má vlastnú kópiu repozitára
  - nie je potrebné online pripojenie k hlavnému repozitáru
    ⇒možná práca offline
- Online repozitáre: GitHub, GitLab, SourceForge

Inštalácia gitu

- Windows:
  - Git for Windows
- Linux
  - dostupné v repozitári vašej distribúcie
- Mac
  - Git for Mac



Základná konfigurácia gitu

```
$ git config --global user.name "Adam Novák"
$ git config --global user.email adam.novak@stuba.sk
```

Základná terminológia gitu

- **commit** – *snímok* stavu zdrojových kódov – obsahuje časovú pečiatku, meno autora a krátky popis zmien, ktoré boli v snímku vykonané
- **vetva** – zreťazený zoznam **commitov**, ktorý určuje históriu zmien v projekte

Tri priestory gitu

Working tree

Aktuálny stav priečinka, v ktorom sú zdrojové súbory

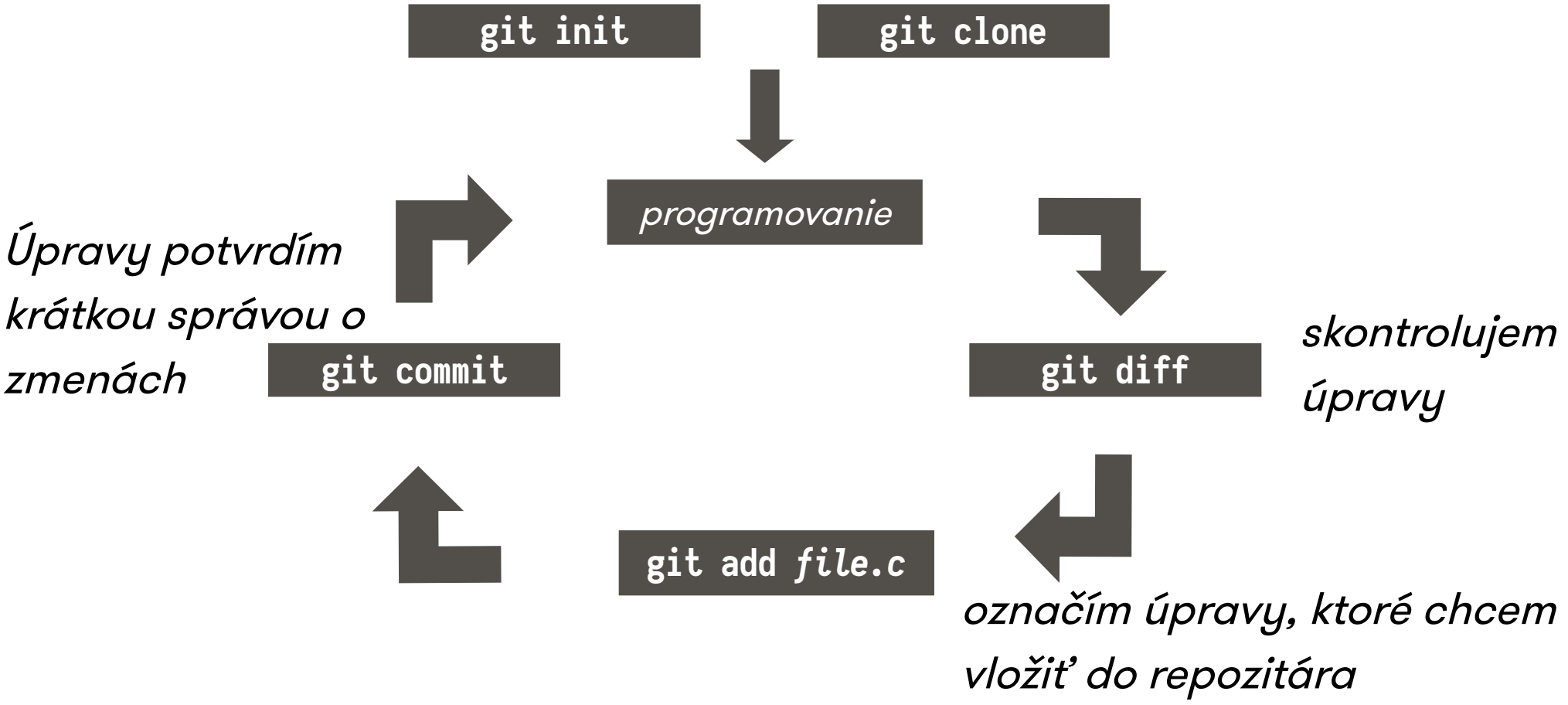
Staging area (Index)

Časti súborov, ktoré sú pripravené na uloženie do histórie

Commit history

Uchováva všetky zmeny v projekte

Základy používania gitu



Základné príkazy gitu

- **`git init`** – vytvorí nový repozitár
- **`git diff`** – zobrazí rozdiely medzi aktuálnym priečinkom a indexom
- **`git add .`** - pridá všetky súbory do **indexu**
- **`git add todo.c todo.h`** – pridá dané súbory do **indexu**
- **`git commit`** – uloží úpravy v indexe do **histórie**

Prezeranie histórie

- **`git log`** – zobrazí históriu (správy ku každému *commitu*)
- **`git log -p`** – zobrazí zmeny súborov v každom *commite*

Vetvenie

- **`git branch`** - zobrazí existujúce vetvy
- **`git branch dev`** – vytvorí novú vetvu **`dev`**
- **`git checkout dev`** – prepne pracovný priečinok na vetvu **`dev`**

Spájanie vetiev

- Úloha: chcem presunúť zmeny z vetvy **`dev`** do vetvy **`master`**
  - **`git checkout master`**
  - **`git merge dev`**
- Úloha: z ľubovoľnej vetvy chcem aplikovať len jeden commit do vetvy **`master`**
  - **`git checkout master`**
  - **`git cherry-pick *commit-hash*`**

Spolupráca

- git je **distribuovaný** systém – každý vývojár má vlastnú kópiu repozitára

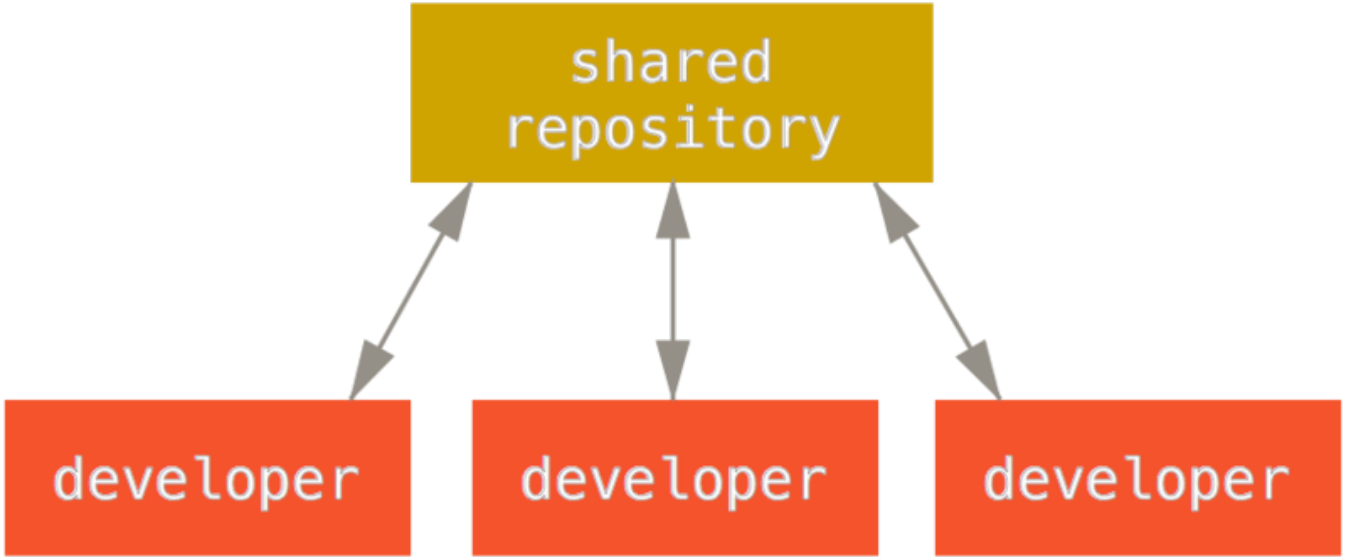


image source

Spolupráca

- Pridanie vzdialenej vetvy
  - **`git remote add origin http://github.com/name/repo`**
- Práca so vzdialenou vetvou
  - **`git push origin`** – vložiť svoju prácu do vzdialenej vetvy
  - **`git pull origin`** – prevziať zmeny zo vzdialenej vetvy

Viac informácií o gite

- inštalácia gitu obsahuje skvelú dokumentáciu
  - Windows: **`Git\mingw64\share\doc\git-doc\index.html`**
  - Linux: **`man git`**
  - Internet: **`https://git.github.io/htmldocs/git.html`**
- Odporúčam sekcie **`gittutorial`** a **`giteveryday`**

časť štvrtá

dokumentácia

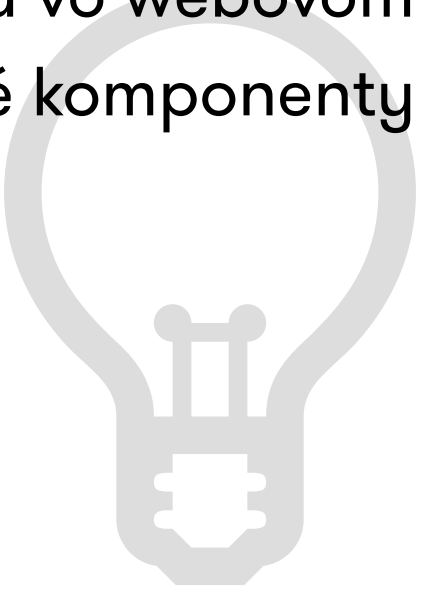
doxygen

Motivácia

- dokumentácia je achillovou pätou každého softvérového projektu
- Programátorom sa nechce písať samostatnú dokumentáciu. Ak existuje a urobia zmenu v kóde, neaktualizujú dokumentáciu.

Riešenie

- písať dokumentáciu priamo do kódu
- automatizovaný nástroj dokumentáciu z kódu extrahuje a vytvorí on-line dokumentáciu prezerateľnú vo webovom prehliadači obsahujúcu všetky previazané komponenty



Doxygen

- štandardný nástroj pre dokumentáciu C++
- podporuje C, Objective-C, C#, PHP, Java, Python a ďalšie jazyky

Ako dokumentovať kód

- Doxygen podporuje viac verzií dokumentačných komentárov:

```
/**
 * ... text ...
 */
```

```
///
/// ... text ...
///
```

```
//!
//!... text ...
//!
```

```
/*!
 * ... text ...
 */
```

```
/*!
 ... text ...
*/
```

```
/*********************************************//**
 *  ... text
 **********************************************/
```

Ako dokumentovať kód

- Pre jazyk C sa dokumentácia obvykle píše do hlavičkových súborov

Dokumentácia funkcie

```
/**
 * @brief Example showing how to document a function with Doxygen.
 *
 * Description of what the function does. This part may refer to the parameters
 * of the function, like @p param1 or @p param2. A word of code can also be
 * inserted like @c this which is equivalent to <tt>this</tt> and can be useful
 * to say that the function returns a @c void or an @c int.

 * @param param1 Description of the first parameter of the function.
 * @param param2 The second one, which follows @p param1.
 * @return Describe what the function returns.
 * @see Box_The_Second_Function
 * @note Something to note.
 * @warning Warning.
 */
BOXEXPORT BoxStruct *
Box_The_Function_Name(BoxParamType1 param1, BoxParamType2 param2 /*, ...*/);
```

source

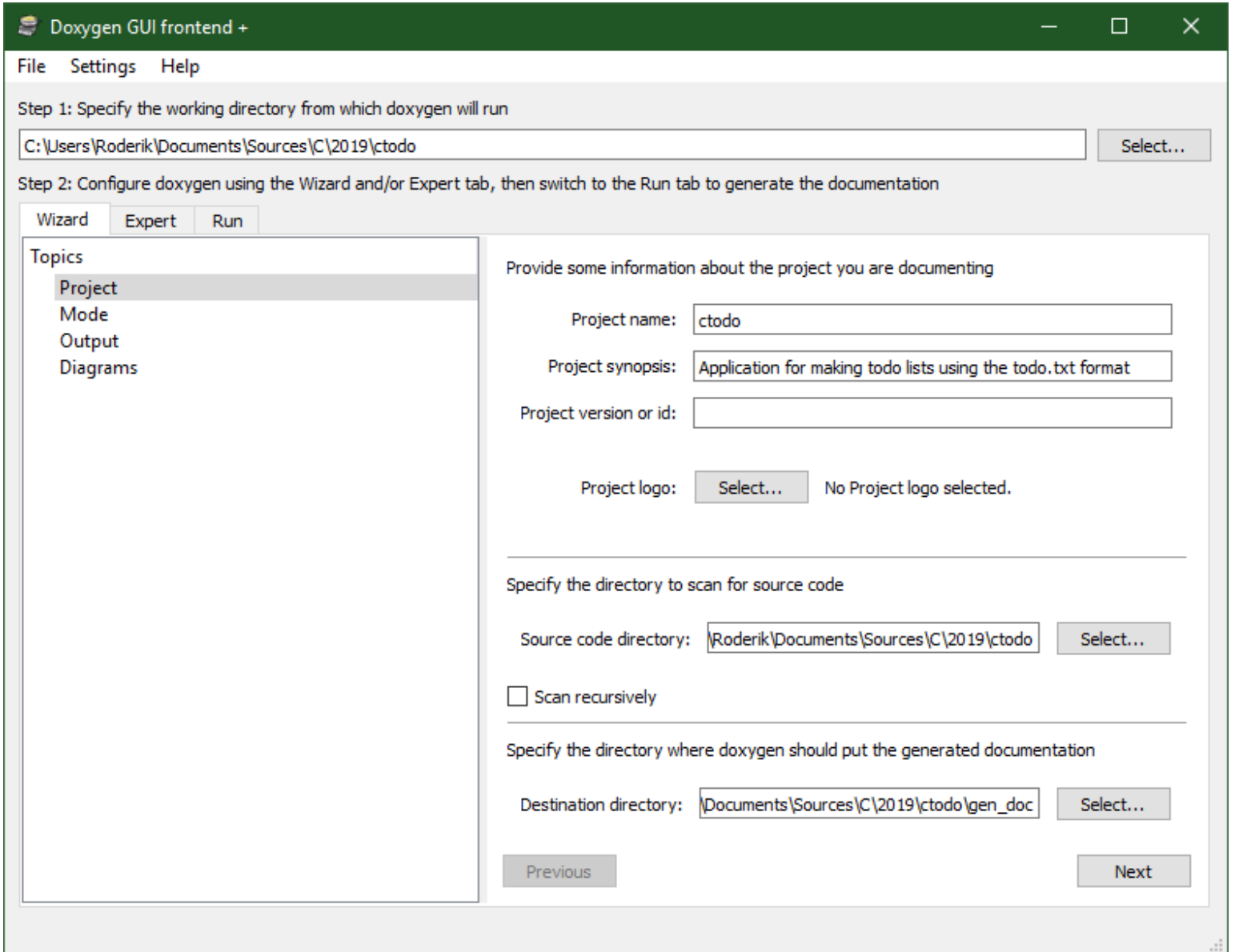
Dokumentácia štruktúry

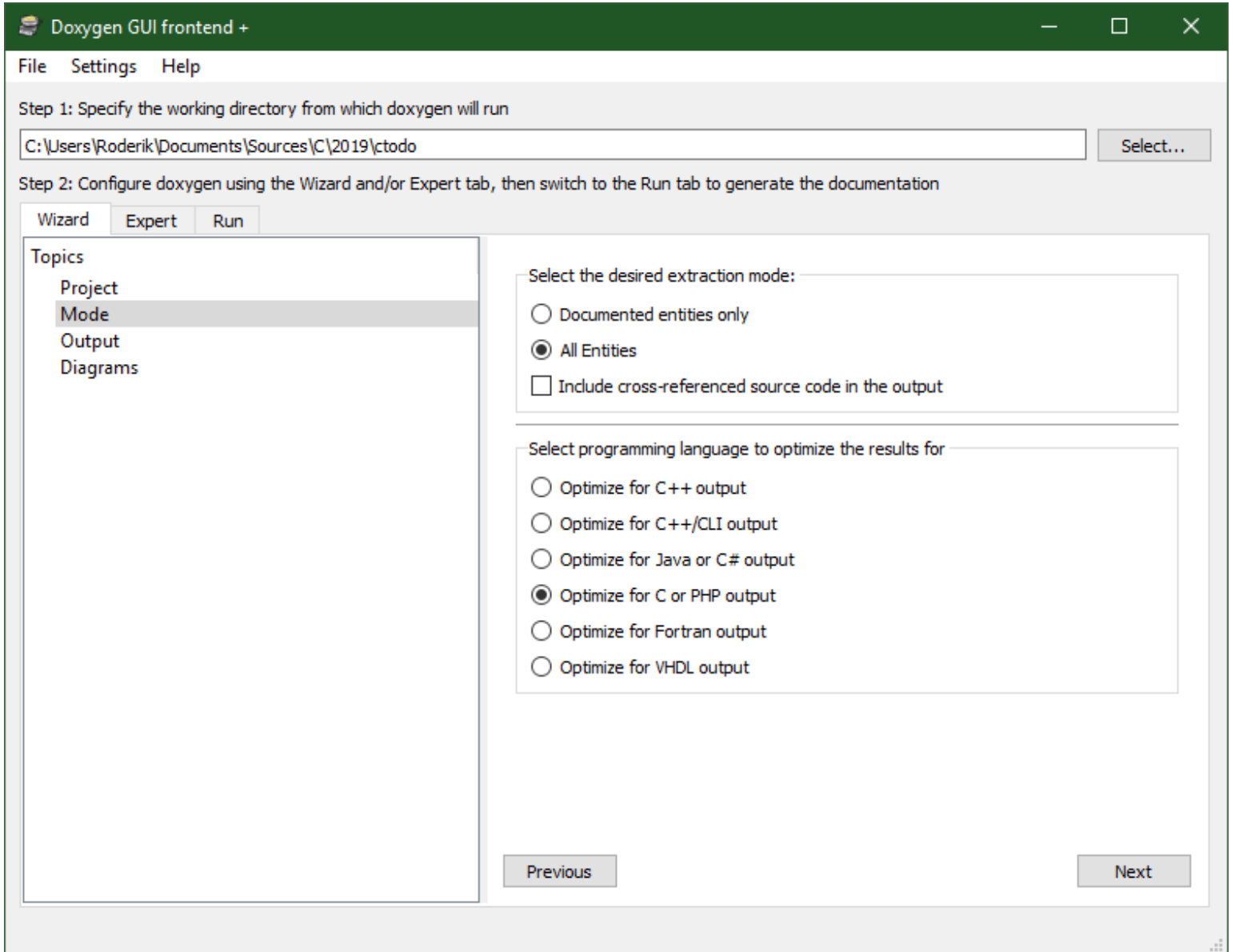
```c
/**
 * @brief Kratky popis struktury.
 *
 * Detailny popis struktury.
 */
typedef struct BoxStruct_struct {
  int a;    /**< Dokumentacia clena BoxStruct#a. */
  int b;    /**< Dokumentacia clena BoxStruct#b. */
  double c; /**< Etc. */
} BoxStruct;
```

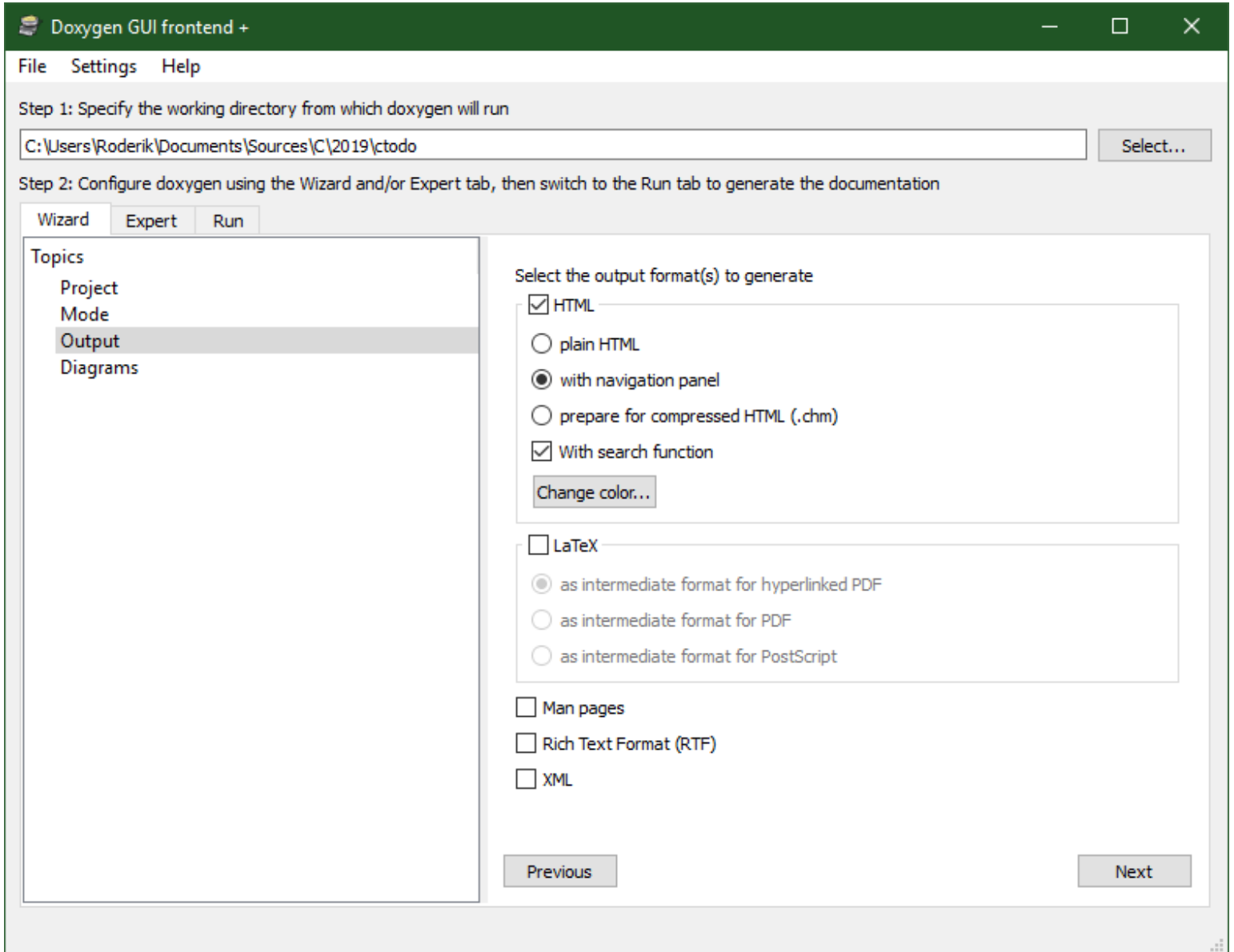
source

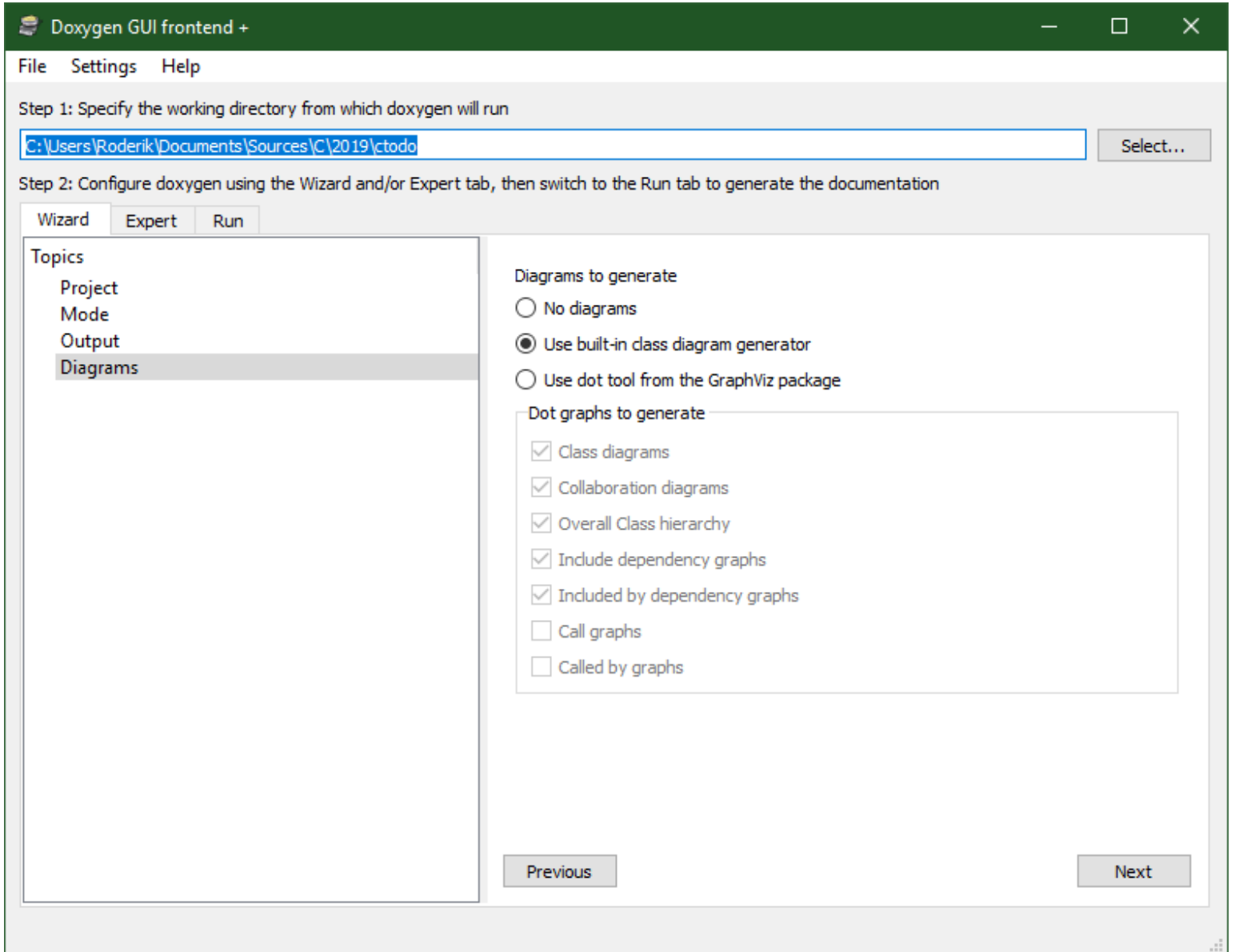
Generovanie dokumentácie

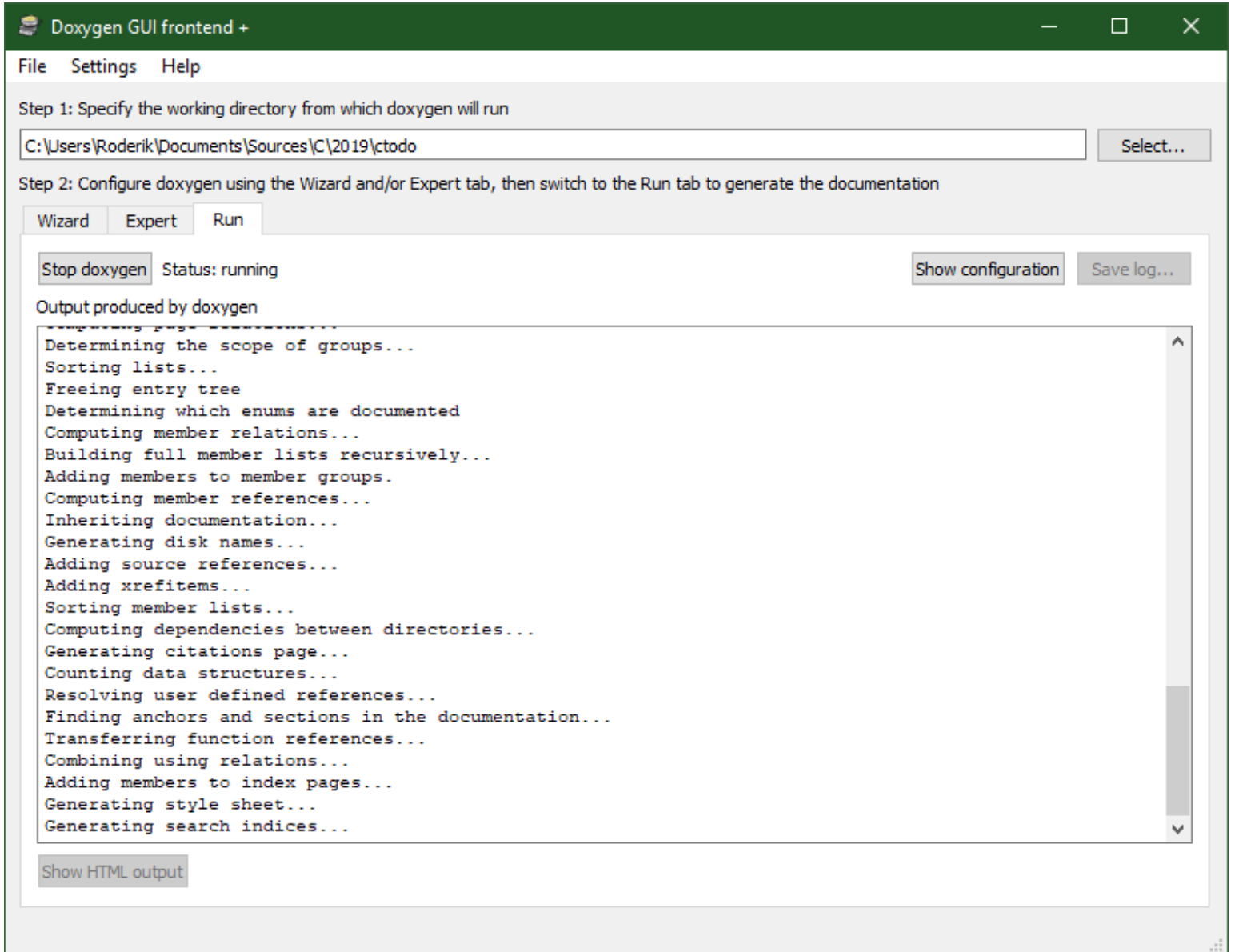
- doxywizard - najpohodlnejšie

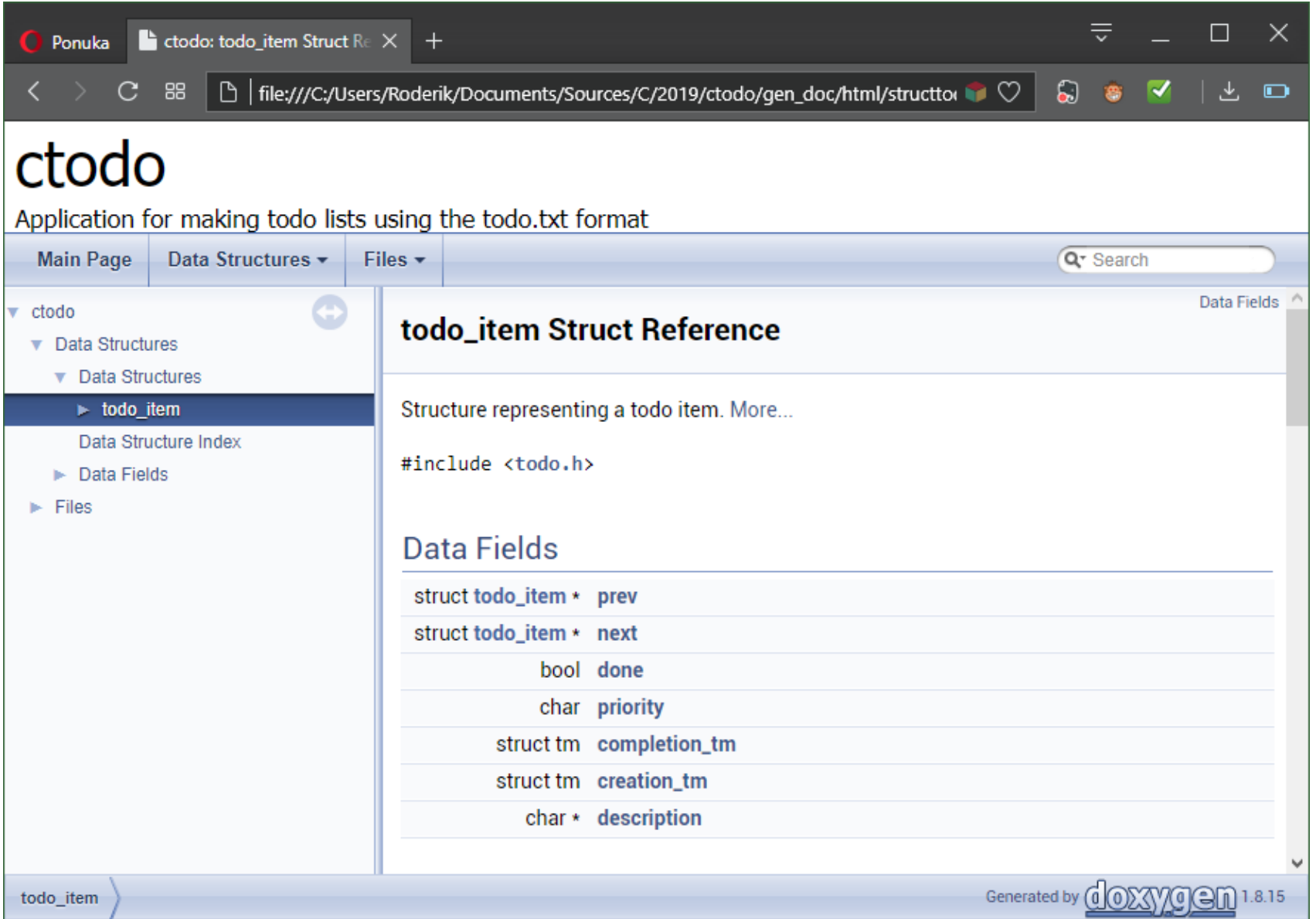

Doxygen GUI frontend +
File
Settings
Help
Step 1: Specify the working directory from which doxygen will run
C:\Users\Roderik\Documents\Sources\C\2019\ctodo
Select...
Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation
Wizard
Expert
Run
Topics
Project
Mode
Output
Diagrams
Provide some information about the project you are documenting
Project name:
ctodo
Project synopsis:
Application for making todo lists using the todo.txt format
Project version or id:
Project logo:
Select...
No Project logo selected.
Specify the directory to scan for source code
Source code directory:
\Roderik\Documents\Sources\C\2019\ctodo
Select...
Scan recursively
Specify the directory where doxygen should put the generated documentation
Destination directory:
\Documents\Sources\C\2019\ctodo\gen_doc
Select...
Previous
Next


Doxygen GUI frontend +
File
Settings
Help
Step 1: Specify the working directory from which doxygen will run
C:\Users\Roderik\Documents\Sources\C\2019\ctodo
Select...
Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation
Wizard
Expert
Run
Topics
Project
Mode
Output
Diagrams
Select the desired extraction mode:
Documented entities only
All Entities
Include cross-referenced source code in the output
Select programming language to optimize the results for
Optimize for C++ output
Optimize for C++/CLI output
Optimize for Java or C# output
Optimize for C or PHP output
Optimize for Fortran output
Optimize for VHDL output
Previous
Next


Doxygen GUI frontend +
File
Settings
Help
Step 1: Specify the working directory from which doxygen will run
C:\Users\Roderik\Documents\Sources\C\2019\ctodo
Select...
Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation
Wizard
Expert
Run
Topics
Project
Mode
Output
Diagrams
Select the output format(s) to generate
HTML
plain HTML
with navigation panel
prepare for compressed HTML (.chm)
With search function
Change color...
LaTeX
as intermediate format for hyperlinked PDF
as intermediate format for PDF
as intermediate format for PostScript
Man pages
Rich Text Format (RTF)
XML
Previous
Next


Doxygen GUI frontend +
File
Settings
Help
Step 1: Specify the working directory from which doxygen will run
C:\Users\Roderik\Documents\Sources\C\2019\ctodo
Select...
Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation
Wizard
Expert
Run
Topics
Project
Mode
Output
Diagrams
Diagrams to generate
No diagrams
Use built-in class diagram generator
Use dot tool from the GraphViz package
Dot graphs to generate
Class diagrams
Collaboration diagrams
Overall Class hierarchy
Include dependency graphs
Included by dependency graphs
Call graphs
Called by graphs
Previous
Next


Doxygen GUI frontend +
File
Settings
Help
Step 1: Specify the working directory from which doxygen will run
C:\Users\Roderik\Documents\Sources\C\2019\ctodo
Select...
Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation
Wizard
Expert
Run
Stop doxygen
Status: running
Show configuration
Save log...
Output produced by doxygen
Determining the scope of groups...
Sorting lists...
Freeing entry tree
Determining which enums are documented
Computing member relations...
Building full member lists recursively...
Adding members to member groups.
Computing member references...
Inheriting documentation...
Generating disk names...
Adding source references...
Adding xrefitems...
Sorting member lists...
Computing dependencies between directories...
Generating citations page...
Counting data structures...
Resolving user defined references...
Finding anchors and sections in the documentation...
Transferring function references...
Combining using relations...
Adding members to index pages...
Generating style sheet...
Generating search indices...
Show HTML output


Ponuka
ctodo: todo_item Struct Re
file:///C:/Users/Roderik/Documents/Sources/C/2019/ctodo/gen_doc/html/structto
ctodo
Application for making todo lists using the todo.txt format
Main Page
Data Structures
Files
Search
ctodo
Data Structures
Data Structures
todo_item
Data Structure Index
Data Fields
Files
Data Fields
todo_item Struct Reference
Structure representing a todo item. More...
#include <todo.h>
Data Fields
struct todo_item * prev
struct todo_item * next
bool done
char priority
struct tm completion_tm
struct tm creation_tm
char * description
todo_item
Generated by doxygen 1.8.15

Iné dokumentačné nástroje

- Doxygen podporuje 12 programovacích jazykov, existujú ale aj iné projekty:
  - C#: XML Documentation Comments
  - Java: Javadoc
  - Python: Sphinx
  - *Porovnanie dokumentačných nástrojov*

časť piata

testovanie

unit

Motivácia

- testovanie aplikácie – skúšanie všetkých možných vstupov a sledovanie, či na ne aplikácia správne reaguje
- malá aplikácia – ručné testovanie
- veľká aplikácia – ručné testovanie by bolo zdĺhavé

Druhy testov

- **Unit test** – testuje najmenšie jednotky kódu (obvykle funkcie)
- **Integration test** – testuje väčšie celky (previazanie komponentov aplikácie)
- Ďalej sa budeme venovať **unit testom**

Testovací framework

- Jazyk C nemá testovací framework, preto je nutné použiť nejaký externý
- Unity
  - jednoduchý unit testing framework (1 `.c` súbor a 3 `.h` súbory)
  - nemýliť si s rovnomenným game enginom

Test v Unity

- testy sú **`.c`** súbory
- štandardne sa píše jeden test pre jeden modul
- test obsahuje:
  - include **`unity.h`** a hlavičkového súboru modulu, ktorý testujeme
  - definície funkcií **`setUp()`** a **`tearDown()`**
  - testovacie funkcie
  - funkciu **`main()`**

Test v Unity - detaily

- funkcia **`setUp()`** bude spustená pred testovaním
- funkcia **`tearDown()`** bude spustená po testovaní
- testovacie funkcie podľa dohody začínajú **`test_`** alebo **`spec_`**
- funkcia **`main()`** obsahuje:
  - volanie makra **`UNITY_BEGIN()`**
  - volanie testov pomocou makra **`RUN_TEST(test_function_A)`**
  - vráti hodnotu, ktorú vráti makro **`UNITY_END()`**

Biolerplate testu v Unity

```
#include "unity.h"
#include "file_to_test.h"

void setUp(void) {
    // set stuff up here
}

void tearDown(void) {
    // clean stuff up here
}
```

```
void test_function_should_doBlahAndBlah(void) {
    //test stuff
}

void test_function_should_doAlsoDoBlah(void) {
    //more test stuff
}

int main(void) {
    UNITY_BEGIN();
    RUN_TEST(test_function_should_doBlahAndBlah);
    RUN_TEST(test_function_should_doAlsoDoBlah);
    return UNITY_END();
}
```

Štruktúra testovacej funkcie

Pri testovaní chceme overiť, či sa výstup funkcie zhoduje s našim predpokladom.

- Využívajú sa na to **`TEST_ASSERT`** makrá

`TEST_ASSERT_TRUE(condition)` – test prejde, ak má **`condition`** nenulovú hodnotu
`TEST_ASSERT_EQUAL(exp, act)` – hodnota **`act`** musí byť rovná hodnote **`exp`**
`TEST_ASSERT_EQUAL_STR(exp, act)` – reťazec **`act`** musí byť zhodný s reťazcom **`exp`**

Ukážka testovacej funkcie

```
void test_date(void) {
    struct tm standard = {.tm_year=2019-1900, .tm_mon=4, .tm_mday=1,};
    struct tm trial = {0};

    char* buffer = "2019-04-01";

    TEST_ASSERT_TRUE(read_date(buffer, &trial));
    TEST_ASSERT_EQUAL_MEMORY(&standard, &trial, sizeof(struct tm));
}
```

Bonus: make pravidlo pre testy

```
TEST_DEP = tests/unity.o tests/unity.h

tests/test_date.o: date.o

test: tests/test_date.o $(TEST_DEP)
    $(CC) $(CFLAGS) tests/test_date.o tests/unity.o date.o -o \
    test_date.exe
```

Výstup

Výstupom Unity sú spustiteľné súbory pre každý test. Po ich spustení sa podľa úspešnosti testu zobrazí výstup:

```
tests/test_date.c:22:test_date:PASS

-----------------------
1 Tests 0 Failures 0 Ignored
OK
```

Úspešný test

```
tests/test_date.c:17:test_date:FAIL: Memory Mismatch. Byte 12 Expected 0x01 Was 0x02

-----------------------
1 Tests 1 Failures 0 Ignored
FAIL
```

Zlyhanie testu

Iné testovacie nástroje

- Veľká časť testovacích frameworkov dodržuje štruktúru xUnit
- C#: MSTest
- Java: JUnit
- Python: unittest (súčasť štandardnej knižnice)
- *Zoznam testovacích frameworkov*

časť šiesta

analýza pamäte

valgrind

Motivácia

- V `C` sa o správu pamäte stará programátor
- `C` nekontroluje prístupy k poliam (môžeme prepísať hodnoty za hranicami poľa
  - Spôsobí to nedefinované správanie – program môže spadnúť alebo pokračovať s porušenou pamäťou
- Ak dynamicky alokujeme pamäť a stratíme na ňu smerník bez zavolania funkcie `free`, už sa k tej pamäti nikdy nedostaneme
  - Ak sa niečo také deje v cykle ⇒ pamäťový únik. Program postupne vyčerpá všetky zdroje systému.
- Ako overiť, či program správne manipuluje s pamäťou?

Riešenie

- Valgrind
  - kontroluje prístupy do pamäte, alokácie, dealokácie pamäte a pamäťové úniky
- Inštalácia
  - v repozitári vašej distribúcie
  - webové stránky projektu

Použitie

`valgrind` *`testovany_program argumenty`* …

Pre hľadanie pamäťových únikov použite prepínač

`--leak-check=yes`

Ukážkový výstup I. *Zápis mimo poľa*

Vo funkcii **`parse_todo_item()`** vyvolajme *zámernú* chybu na riadku 62:

```
description_length = strlen(buffer+position);
todo->description = (char*) malloc((description_length+1)*sizeof(char));
if (todo->description == NULL)
  return -ENOMEM;
memcpy(todo->description, buffer+position, description_length-1);
todo->description[position+description_length-1] = '\0';
```

Ukážkový výstup I. *Zápis mimo poľa*

```
==3418== Memcheck, a memory error detector
==3418== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3418== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==3418== Command: ./todo
==3418==
==3418== Invalid write of size 1
==3418==    at 0x1091B5: parse_todo_item (todo.c:62)
==3418==    by 0x1092EF: load_todo_items (todo.c:107)
==3418==    by 0x108F4C: main (main.c:10)
==3418==  Address 0x522e49b is 2 bytes after a block of size 25 alloc'd
==3418==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3418==    by 0x109146: parse_todo_item (todo.c:58)
==3418==    by 0x1092EF: load_todo_items (todo.c:107)
==3418==    by 0x108F4C: main (main.c:10)
```

Ukážkový výstup II. *Pamäťový únik*

Program s evidentným pamäťovým únikom:

```
#include <stdlib.h>

 int main(void) {
     int i =0;
     char* buffer;

     while (i < 10) {
         buffer = (char*) malloc(16);
         i++;
     }
     return 0;
 }
```

Ukážkový výstup II. *Pamäťový únik*

```
==3455== HEAP SUMMARY:
==3455==     in use at exit: 160 bytes in 10 blocks
==3455==   total heap usage: 10 allocs, 0 frees, 160 bytes allocated
==3455==
==3455== 160 bytes in 10 blocks are definitely lost in loss record 1 of 1
==3455==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3455==    by 0x108664: main (in /home/os2018/Documents/ctodo/todo1)
==3455==
==3455== LEAK SUMMARY:
==3455==    definitely lost: 160 bytes in 10 blocks
==3455==    indirectly lost: 0 bytes in 0 blocks
==3455==      possibly lost: 0 bytes in 0 blocks
==3455==    still reachable: 0 bytes in 0 blocks
==3455==         suppressed: 0 bytes in 0 blocks
==3455==
==3455== For counts of detected and suppressed errors, rerun with: -v
==3455== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

FOOS

časť siedma

open source

Kde nájsť otvorený softvér?

- GitHub
- SourceForge
- OSDN

Prečo sa zapojiť do vývoja open source?

- Budovanie portfólia
- Skúsenosti
- Príležitosť sa naučiť nový jazyk, resp. technológie
- Pomôcť komunite
- Je to zábava!

Ako sa zapojiť do vývoja?

- Nájsť FOSS projekt
- Pozrieť *issues* – tag **`help wanted`** alebo **`good first issue`**
- **`fork`** a **`git clone`**
- Urobiť zmeny v kóde
- Otestovať
- Napísať *pull request*

Do ktorého projektu prispieť?

Zoznam vhodných projektov pre začiatočníkov:

https://github.com/MunGell/awesome-for-beginners

Štruktúra projektov

- doc – dokumentácia
- includes – hlavičkové súbory
- src – zdrojové súbory
- test – zdrojové súbory testov
- CONTRIBUTE – pokyny pre prispievanie kódu do projektu
- README – čítaj ma
- LICENSE – licencia

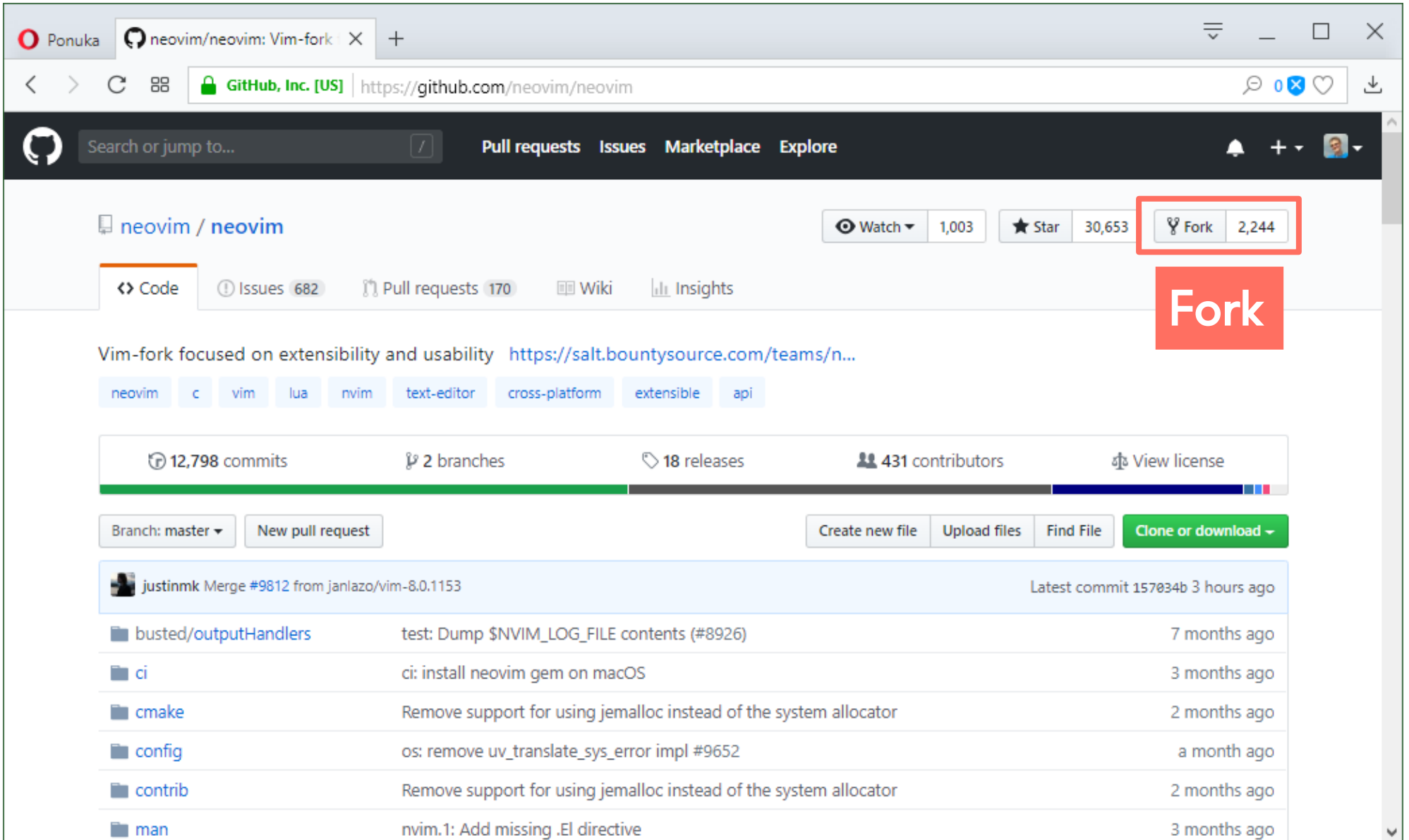
Licencia

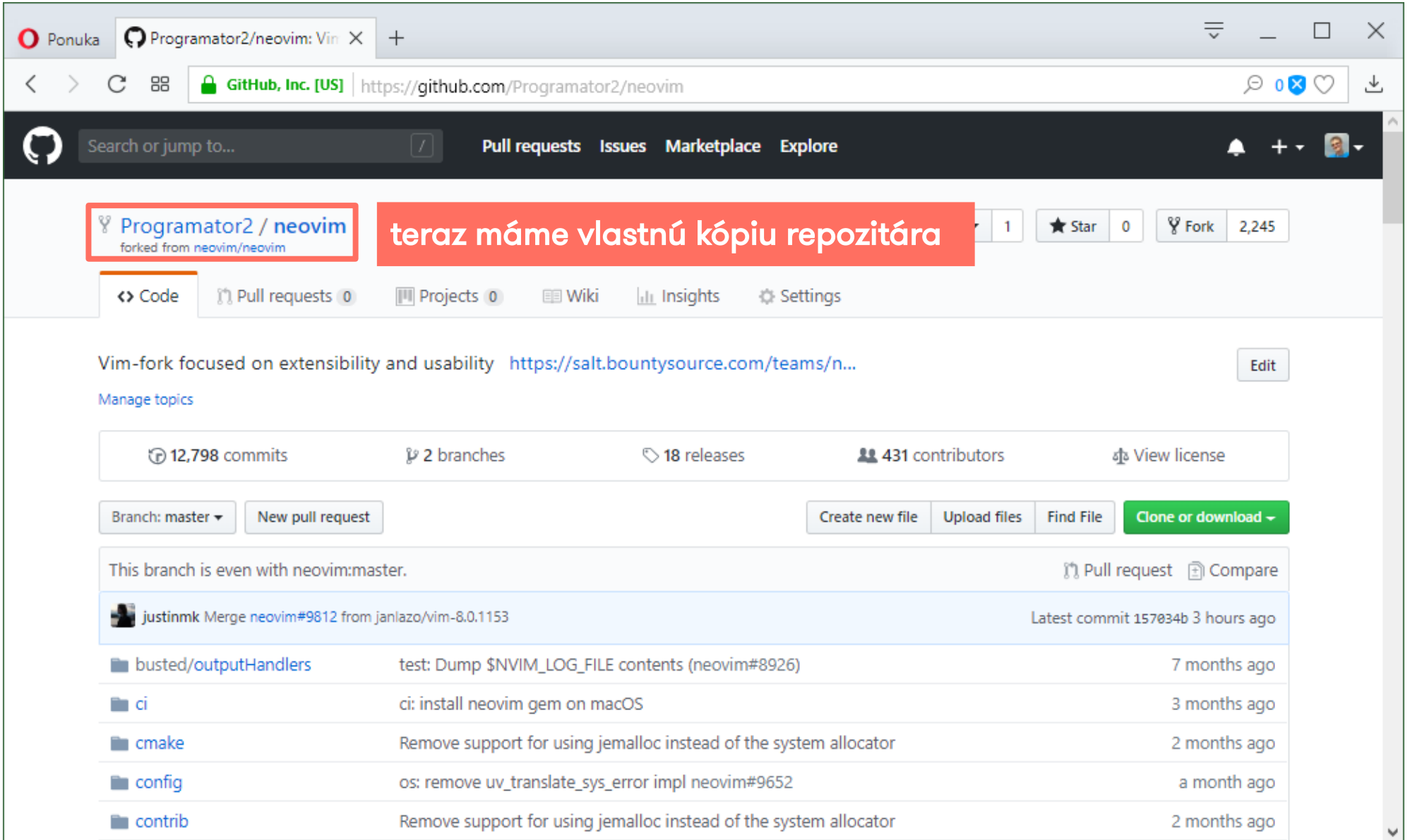
Projekt bez uvedenej licencie je pod plným copyrightom a nikto s ním nemôže nič urobiť!

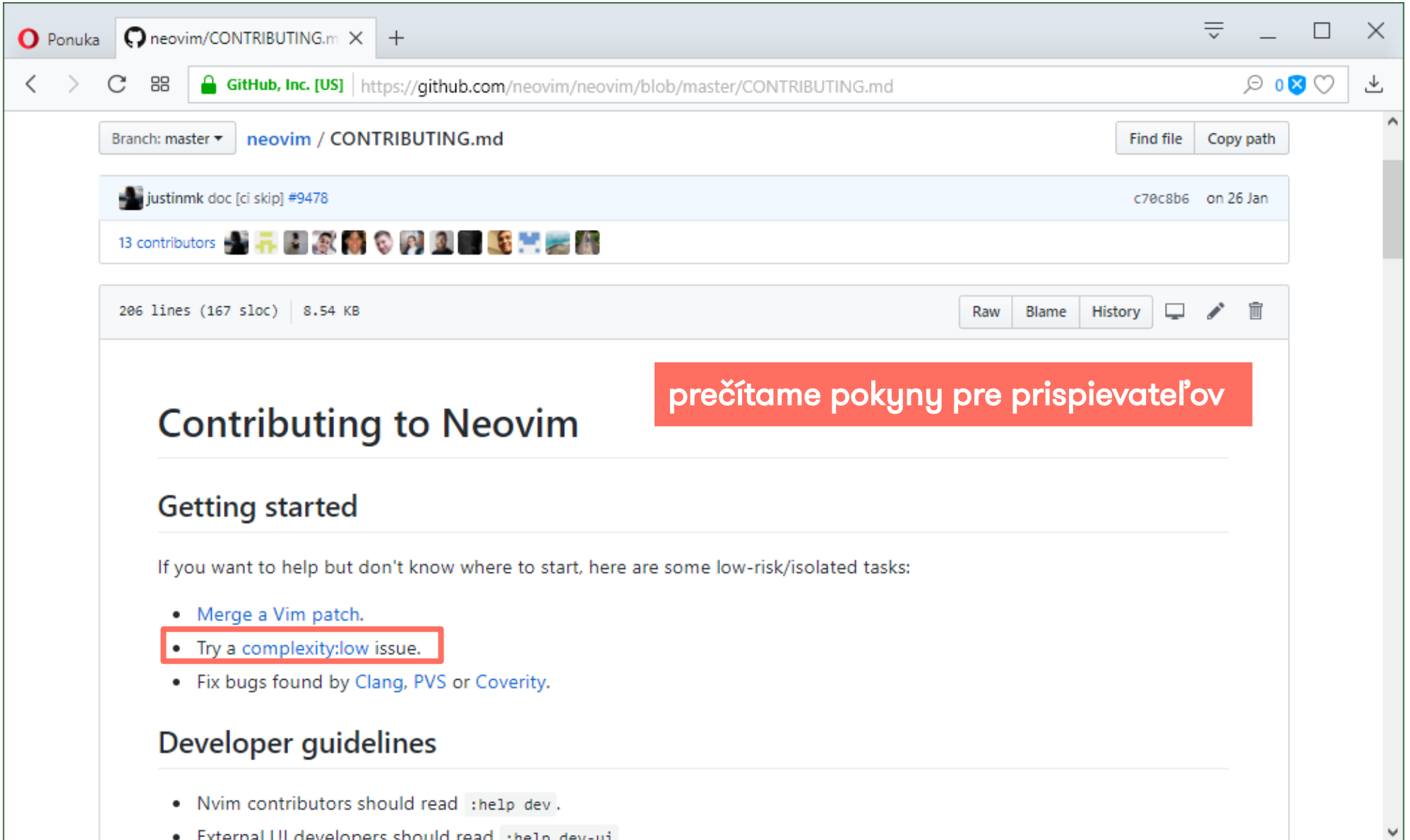
Druhy FOOS licencií

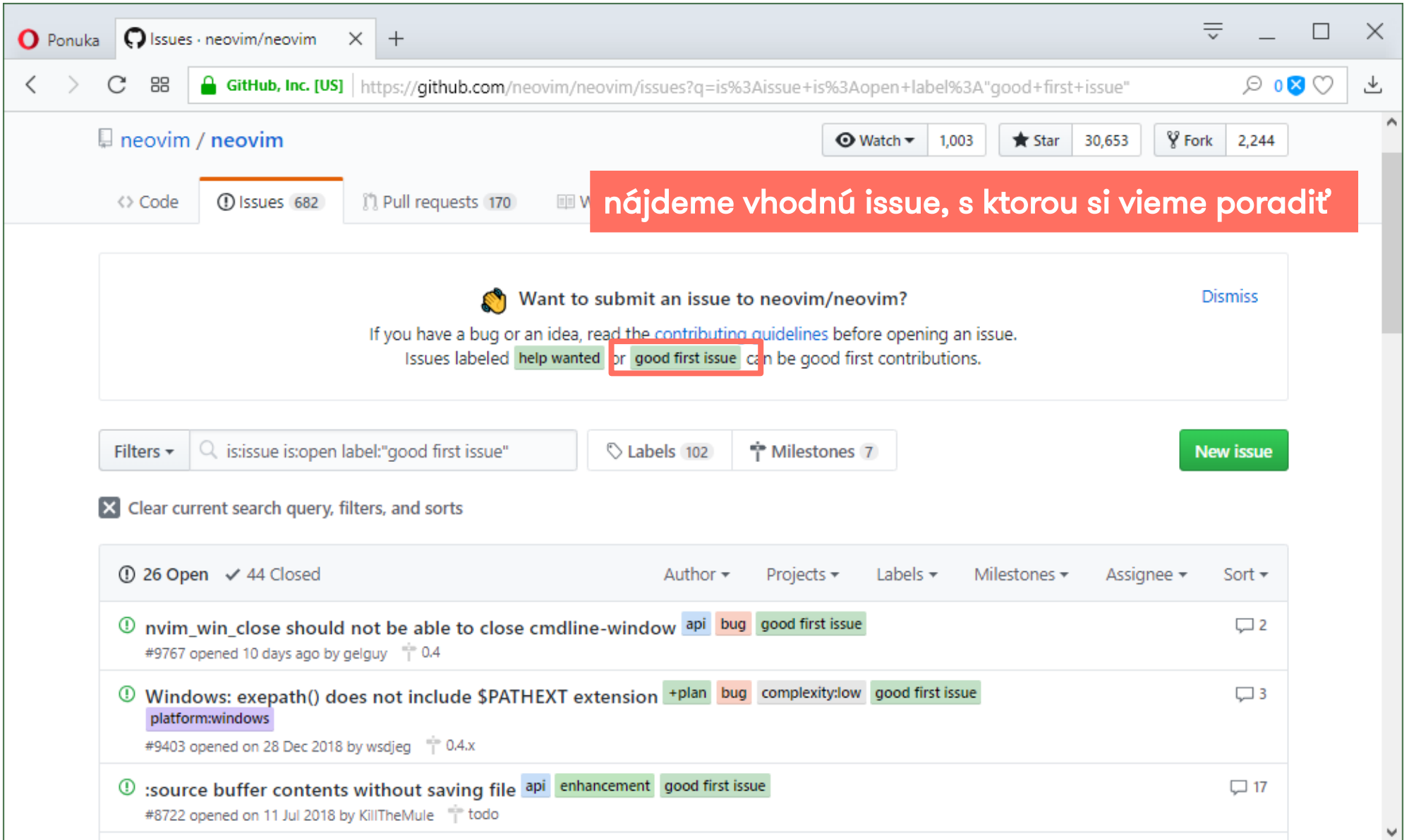
- GPLv3 – každý odvodený **distribuovaný** projekt musí byť licencovaný rovnako (t.j. keď softvér upravím a chcem ho vydať, musím zverejniť zdrojový kód aj s úpravami – ťažšie využitie v komerčnej sfére)
- MIT – umožňuje voľné využitie zdrojového kódu a nevyžaduje jeho zverejnenie v prípade **distribúcie** upraveného softvéru

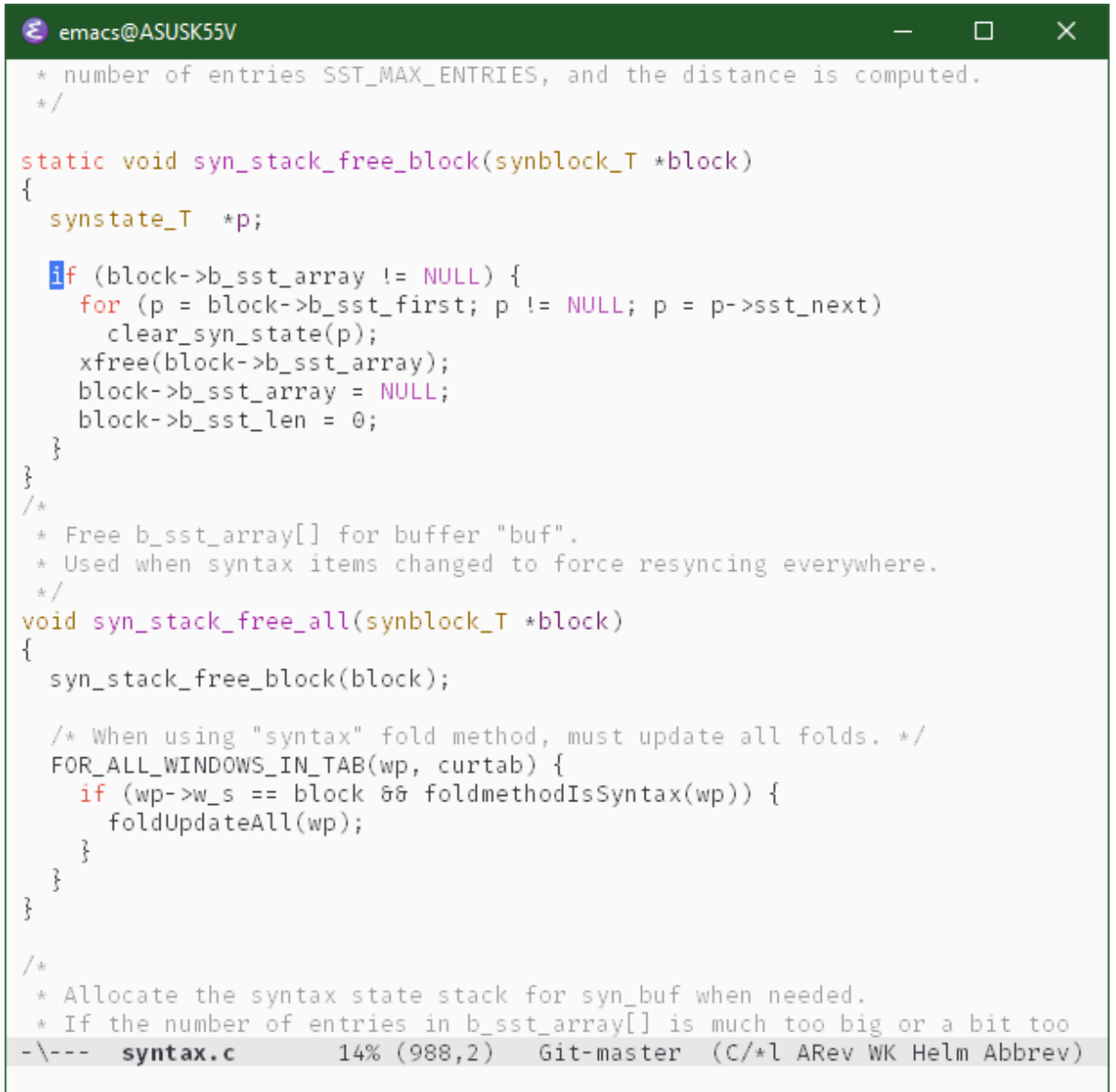
https://choosealicense.com/


Ponuka
neovim/neovim: Vim-fork
GitHub, Inc. [US]
https://github.com/neovim/neovim
Search or jump to...
Pull requests
Issues
Marketplace
Explore
neovim / neovim
Watch 1,003
Star 30,653
Fork 2,244
Code
Issues 682
Pull requests 170
Wiki
Insights
Fork
Vim-fork focused on extensibility and usability https://salt.bountysource.com/teams/n...
neovim
c
vim
lua
nvim
text-editor
cross-platform
extensible
api
12,798 commits
2 branches
18 releases
431 contributors
View license
Branch: master
New pull request
Create new file
Upload files
Find File
Clone or download
justinmk Merge #9812 from janlazo/vim-8.0.1153
Latest commit 157034b 3 hours ago
busted/outputHandlers
test: Dump $NVIM_LOG_FILE contents (#8926)
7 months ago
ci
ci: install neovim gem on macOS
3 months ago
cmake
Remove support for using jemalloc instead of the system allocator
2 months ago
config
os: remove uv_translate_sys_error impl #9652
a month ago
contrib
Remove support for using jemalloc instead of the system allocator
2 months ago
man
nvim.1: Add missing .El directive
3 months ago


Ponuka
Programator2/neovim: Vin
GitHub, Inc. [US]
https://github.com/Programator2/neovim
Search or jump to...
Pull requests
Issues
Marketplace
Explore
Programator2 / neovim
forked from neovim/neovim
teraz máme vlastnú kópiu repozitára
1
Star 0
Fork 2,245
Code
Pull requests 0
Projects 0
Wiki
Insights
Settings
Vim-fork focused on extensibility and usability https://salt.bountysource.com/teams/n...
Edit
Manage topics
12,798 commits
2 branches
18 releases
431 contributors
View license
Branch: master
New pull request
Create new file
Upload files
Find File
Clone or download
This branch is even with neovim:master.
Pull request
Compare
justinmk Merge neovim#9812 from janlazo/vim-8.0.1153
Latest commit 157034b 3 hours ago
busted/outputHandlers
test: Dump $NVIM_LOG_FILE contents (neovim#8926)
7 months ago
ci
ci: install neovim gem on macOS
3 months ago
cmake
Remove support for using jemalloc instead of the system allocator
2 months ago
config
os: remove uv_translate_sys_error impl neovim#9652
a month ago
contrib
Remove support for using jemalloc instead of the system allocator
2 months ago


Ponuka
neovim/CONTRIBUTING.m
GitHub, Inc. [US]
https://github.com/neovim/neovim/blob/master/CONTRIBUTING.md
Branch: master
neovim / CONTRIBUTING.md
Find file
Copy path
justinmk doc [ci skip] #9478
c70c8b6 on 26 Jan
13 contributors
206 lines (167 sloc) 8.54 KB
Raw
Blame
History
prečítame pokyny pre prispievateľov
Contributing to Neovim
Getting started
If you want to help but don't know where to start, here are some low-risk/isolated tasks:
Merge a Vim patch.
Try a complexity:low issue.
Fix bugs found by Clang, PVS or Coverity.
Developer guidelines
Nvim contributors should read :help dev.

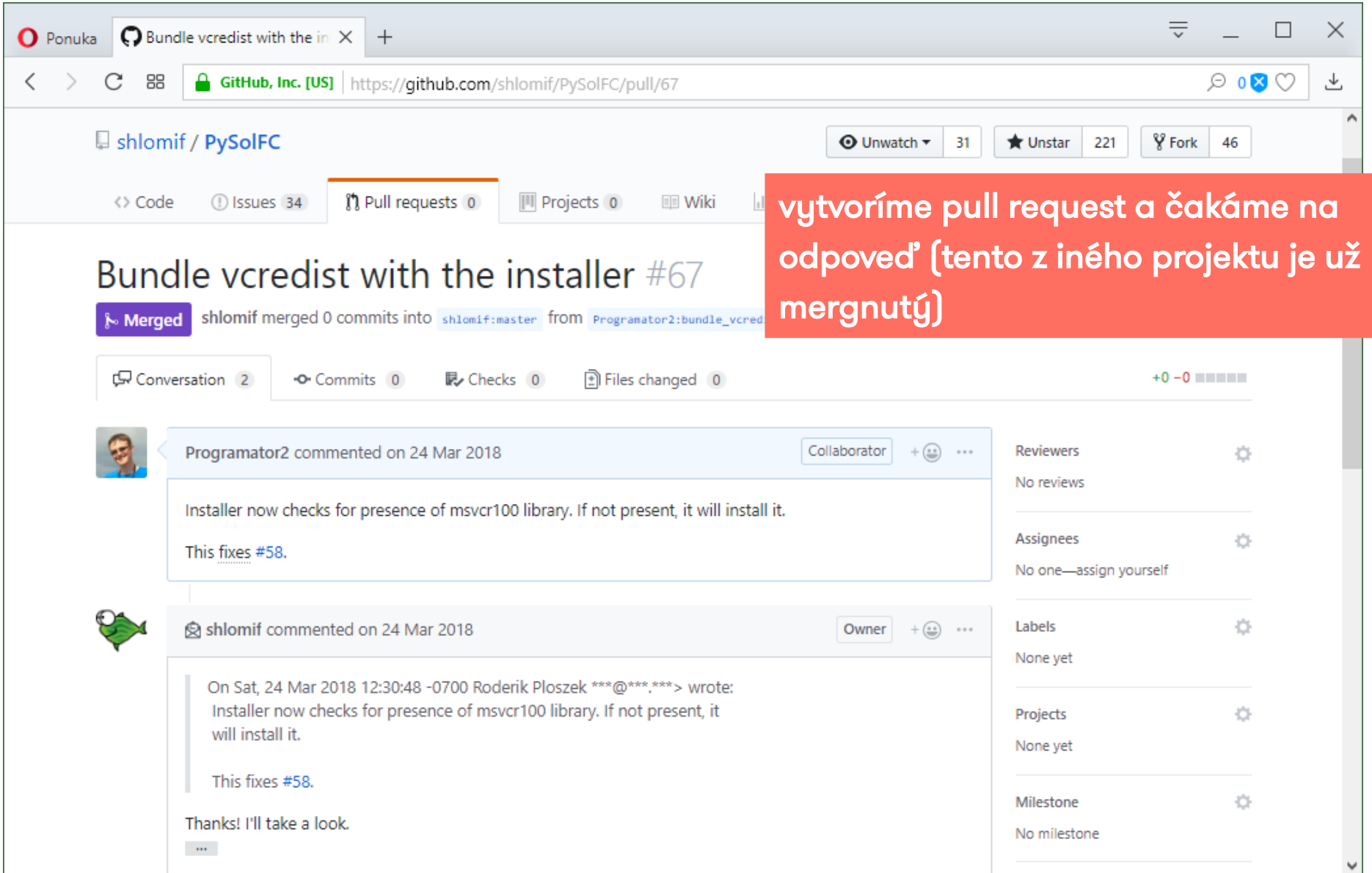

Ponuka
Issues · neovim/neovim
GitHub, Inc. [US]
https://github.com/neovim/neovim/issues?q=is%3Aissue+is%3Aopen+label%3A"good+first+issue"
neovim / neovim
Watch 1,003
Star 30,653
Fork 2,244
Code
Issues 682
Pull requests 170
nájdeme vhodnú issue, s ktorou si vieme poradiť
Want to submit an issue to neovim/neovim?
Dismiss
If you have a bug or an idea, read the contributing guidelines before opening an issue.
Issues labeled help wanted or good first issue can be good first contributions.
Filters
is:issue is:open label:"good first issue"
Labels 102
Milestones 7
New issue
Clear current search query, filters, and sorts
26 Open
44 Closed
Author
Projects
Labels
Milestones
Assignee
Sort
nvim_win_close should not be able to close cmdline-window api bug good first issue
#9767 opened 10 days ago by gelguy 0.4
2
Windows: exepath() does not include $PATHEXT extension +plan bug complexity:low good first issue platform:windows
#9403 opened on 28 Dec 2018 by wsdjeg 0.4.x
3
:source buffer contents without saving file api enhancement good first issue
#8722 opened on 11 Jul 2018 by KillTheMule todo
17



opravíme issue v kóde (dodržujeme predpísaný štýl kódu)

otestujeme, pustíme automatizované testy

vytvoríme commit s jasným popisom úpravy a *pushneme* ho do nášho repozitára


Ponuka
Bundle vcredist with the in
GitHub, Inc. [US]
https://github.com/shlomif/PySolFC/pull/67
shlomif / PySolFC
Unwatch 31
Unstar 221
Fork 46
Code
Issues 34
Pull requests 0
Projects 0
Wiki
vytvoríme pull request a čakáme na odpoveď (tento z iného projektu je už mergnutý)
Bundle vcredist with the installer #67
Merged
shlomif merged 0 commits into shlomif:master from Programator2:bundle_vcred
Conversation 2
Commits 0
Checks 0
Files changed 0
+0 −0
Programator2 commented on 24 Mar 2018
Collaborator
Installer now checks for presence of msvcr100 library. If not present, it will install it.
This fixes #58.
shlomif commented on 24 Mar 2018
Owner
On Sat, 24 Mar 2018 12:30:48 -0700 Roderik Ploszek ***@***.***> wrote:
Installer now checks for presence of msvcr100 library. If not present, it will install it.
This fixes #58.
Thanks! I'll take a look.
...
Reviewers
No reviews
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone

Ďalšie užitočné technológie

- Markdown – markup pre dokumentáciu
- Travis – automatické testovanie a balenie softvéru (continuous integration)
- AppVeyor – *detto* pre Windows

Ďakujem za pozornosť

roderik.ploszek@stuba.sk
C-512