Zmienne

Git

Dziękujemy za nadesłane merge requests do pliku ze złym formatowaniem. Jednak nie zmergujemy ich do brancha master - chcemy, by źle sformatowany kod został "do wglądu". Prosimy zgłaszać merge requests w sytuacjach odnalezienia nieintencjonalnych błędów w kodzie lub treści instrukjci. Błędy znalezione w czasie trwania zajęć prosimy zgłaszać ustnie - poprawimy je na bieżąco.

1. Git - gałęzie

Jednym z najsilniejszych mechanizmów gita jest mechnizm gałęzi. Mechanizm ten umożliwia na tworzenie wersji alternatywnych naszego kodu.

> > Wyobraźmy sobie, że piszemy aplikację na smartfona. Zasadnicze funkcjonalności tej aplikacji zostały zaimplementowane, lecz pojawiła się konieczność dodania funkcjonalności istnienia w aplikacji wielu kont użytkowników wraz z logowaniem (wielodostęp). Jeżeli zaczniemy pracę nad taką funkcjonalnością tak głęboko ingerującą w aplikację - za pewne aktualnie dewelopowana wersja aplikacji przestanie działać - na dłuższy czas. Mechanizm gałęzi rozwiązuje ten problem - możemy jednocześnie rozwijać funkcjonalność logowania a nasi współpracownicy - mogą pracować nad innymi elementami aplikacji, nie zmagając się ze zmianami wprowadzanymi przez nas.

Stworzenie gałęzi powoduje utworzenie alternatywnej wersji

aktualnego repozytorium. Gałąź główna nazywa się najczęściej master. Checkout gałęzi oznacza, że pliki danej gałęzi zastąpią w naszym workspace pliki, które dotychczas się tam znajdowały - oczywiście usunięte pliki nie zostaną utracone.

Polecenie git branch wyświetla dostępne lokalnie gałęzie. Symbolem * oznaczona jest aktualnie checkoutowana gałąź. Polecenie git branch -a wyświetla wszytskie gałęzie - zarówno lokalne jak i dostępne w źródle, z którego klonowaliśmy repozytorium (origin).

W celu utworzenia nowej gałęzi należy wykonać polecenie git branch <nazwa_galezi> lub w celu stworzenia i przełączenia się na nową gałąź git checkout -b <nazwa_galezi>.

> Na potrzeby naszego laboratorium wprowadzamy zasadę, że gałęzie studentów trzymane na gitLabie muszą mieć nazwę wg klucza <Pierwsza_litera_imieniaNazwisko>. (przykład: MGrega). Gałęzie nie spełniające tego wymagania będą regularnie usuwane.

W celu przełączenia się do danej gałęzi należy wykonać polecenie git checkout <nazwa_galezi>.

> Gałęzie są publiczne. Przyjmijmy dobrą zasadę, że nie wysyłamy danych do orgina w cudzej gałęzi.

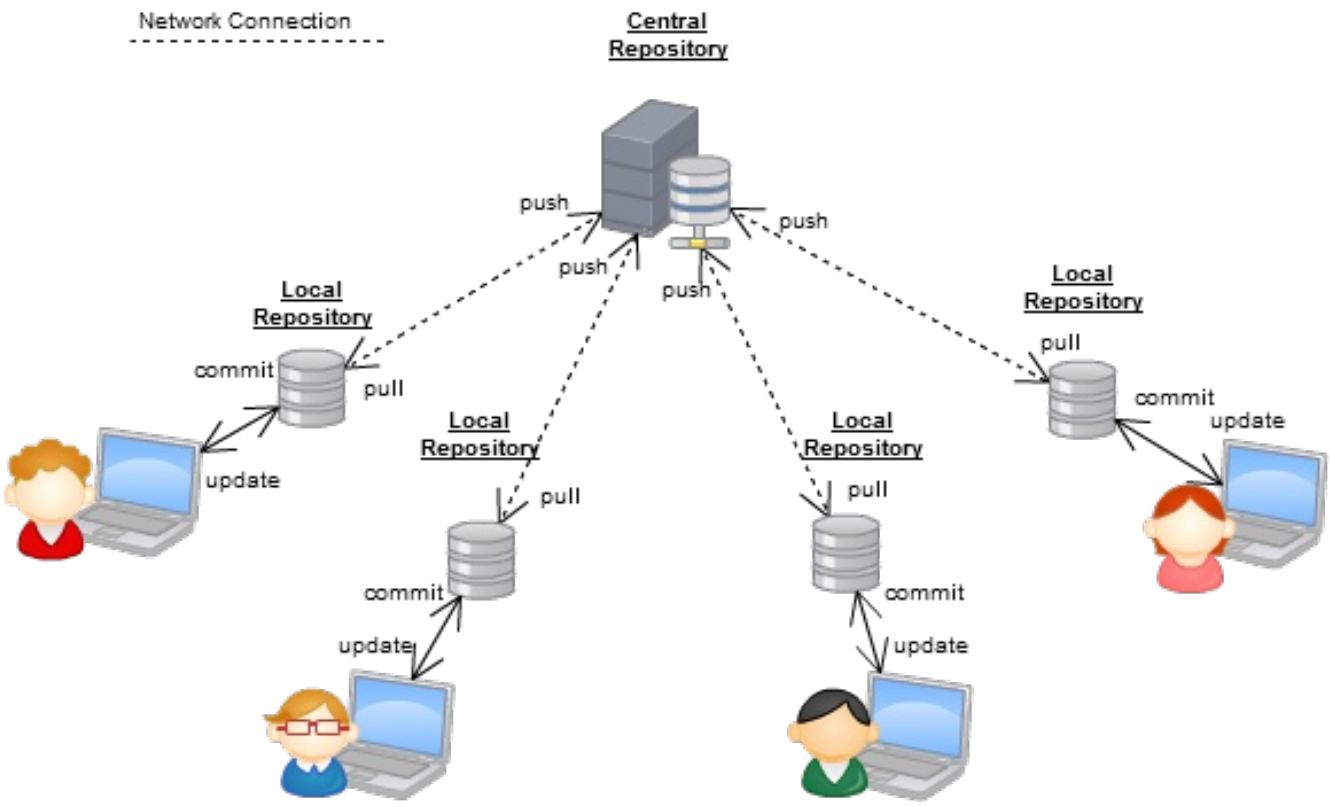
Zadania

1. Wylistuj dostępne w repozytorium zdalnym gałęzie
2. Przełącz się na gałąź dostępną w repozytorium zdalnym
3. Przełącz się na gałąź master (!!)
4. Zrób swój własy branch
5. W swoim branchu zrób nowy plik, commituj go
6. Przełącz sie na branch master - czy jest tam Twój plik?
7. Przełącz sie na swój własny branch - czy jets tam Twój plik?

2. Git - praca z repozytorium zdalnym

Funkcjonalność git jest przydatna w pracy lokalnej - umożliwia przechowywanie wersji plików a także logów zmian. Ale zasadniczą funkcjonalnością git jest praca z repozytorium zdalnym.

Choć, jak napisano w poprzedniej instrukcji git jest repozytorium rozproszonym w którym żadne z repozytoriów nie jest w żaden sposób uprzywilejowane - jednak zazwyczaj używa się go w architekturze gwiazdy, z jednym centralnym repozytorium.



Taka architektura umożliwia wielu osobom współpracę nad jednym kodem.

> Zasadniczo git umozliwia prace nad dowolnymi dokumentami tekstowymi. Nie tylko kodem programów, ale także na przykład dokumentami pisanymi w językach składu teksty (jak na przykład LaTex, w którym pisze się np publikacje naukowe) czy prostszych językach znaczników (jak niniejszy Markdown). Dzięki temu Państwo

> > macie także możliwość poprawienia instrukcji i commitowania zmian (z pewnymi ograniczeniami organizacyjnymi)

Przebieg (wokflow) pracy ze zdelnym repozytorium jest następujący:

1. Pozyskanie repozytorium - git clone <url> jeśli repozytorium jeszcze nie mamy lub git pull jeśli repozytorium mamy - wtedy je aktualizujemy.
2. Wprowadzenie zmian
3. Wykonanie commit -am
4. Wykonanie git push, co spowoduje wysłanie danych z aktualnego brancha do zdalnego repozytorium. Aby wysłać do zdalenego repozytorium nową gałąź należy za pierwszym razem wykonać git push -u origin <nazwa_brancha>

> > Nie macie państwo możliwości robienia push do gałęzi master. Ma to na celu ochronę gałęzi głównej. Natomiast możecie państwo robić push do własnych gałęzi

Zadanie

1. Wyślij swój branch do origin
2. Na zadanie domomwe - podłącz się do gitLaba i ściągnij gałąź master i swoją własną gałąź

> > Od tej pory oczekujemy, że cały napisany w czasie zajęć kod będzie przechowywany w Państwa gałęziach na bieżąco. Oznacza to konieczność wielokrotnego commitowania i pushowania w czasie zajęć. Proszę się do tego przyzwyczaić - to dobry nawyk.

C++

1. Działania arytmetyczne w C++

Proszę napisać program deklarujący i wypisujący zmienne.

- Na zmiennych powinny zostać wykonane różne operacje arytmetyczne takie jak, %, +, *, ++, +=.
- Proszę się nie ograniczyć tylko do tych operacji
- Proszę spróbować podzielić przez zero.

Program powinien pokazać wpływ typu zmiennej oraz kolejności operacji na otrzymaną wartość.

- Dodatkowo proszę dać przykład przepełnienia zmiennej, można do tego użyć wartości INT_MAX z biblioteki climits.

Na początek można posłużyć się kodem:

```
#include <iostream>

int main(){
  //variable initialization
  int x = 1;
  //operation visualization
  std::cout << x << std::endl;
  //example of an operator +
  x = x + 5;
  //operation visualization
  std::cout << x << std::endl;
}
```

Do kompilacji proszę użyć komendy g++ -Wall program.cpp, pozwoli to na otrzymanie informacji o błędach i ostrzeżeniach.

2. Wyrażenia logiczne

Proszę napisać program wykorzystujący operatory logiczne ==, !=, >,

<, >=, <=.

- Proszę połączyć kilka operatorów logicznych z wykorzystaniem &&, || oraz negacji !.
- Proszę sprawdzić jaką wartość liczbową ma prawda i fałsz.
- Proszę wykorzystać słowa specjalne false, true.
- Proszę sprawdzić jak działa kolejność działań i inkrementacja wraz z wyrażeniami logicznymi, czyli wyrażenia typu:

```
int x = 1;
bool b = ++x==2;
bool c = x>0 || ++x>0
std::cout << "x " << x << ", b " << b << ", c " << c
<< std::endl;
```

Mamy nadzieję, że utwierdzi to Państwa w przekonaniu, że nie warto tak pisać kodu.

- Proszę sprawdzić, czy wartość zmiennej x znajduje się w przedziale od 3 do 7. Proszę wykorzystać zaprzeczenia całego wyrażenia logicznego.

3. Operatory bitowe

Proszę napisać program wykorzystujący opertory bitowe |, &, ^, <<, >>, ~.

- Proszę sprawdzić jaka jest relacja przesunięcia bitowego i podnoszenia 2 do potęgi.
- Proszę zmienić wartość pojedynczego bitu.
- Prszę sprawdzić jaki błąd zostanie wygenerowany jeżeli zamiast operatora logicznego && użyjemy bitowego &.

4. Operatory rzutowania

Proszę wykorzystać operatory rzutowania. Proszę sprawdzić działanie takiego kodu:

```
int x = 10;
short a = x;
short b = (short)x;
short c = short(x);
short d = static_cast<short>(x);
```

- Jak zmieni się wynik jeżeli pierwszą linię zastąpimy wpisem int x = 1<<20?
- Jeżeli wyjściowa zmienna x jest typu short to, czy przy rzutowaniu jej na int możemy stracić część informacji?
- Proszę sprawdzić informację otrzymaną z funkcji sizeof().
- Jak działa rzutowanie dla zmiany z float na int?

5. Tekst

Proszę napisać program obrazujący, jak kody ASCII reprezentowane są w C++.

- Proszę dodać liczbę do zmiennej typu char.
- Proszę zainicjować zmienną typu char z wykorzystaniem liczby.
- Proszę sprawdzić działanie std::cout << e << "\n" oraz std::cout << (int)e << "\n" gdzie e zdefiniowane jest jako char e = '2'.

6. Liczby zmiennoprzecinkowe

Proszę sprawdzić, dla jakich liczb a i b otrzymamy błędne wyniki dla kodu:

```
float a = 1;
float b = 10000;
std::cout << (a + b) - b << std::endl;
```

```
std::cout << a + (b - b) << std::endl;
```

Proszę z dużą precyzją wypisać wynik działania operacji 1/5 z wykorzystaniem biblioteki iomanip oraz kodu:

```
std::cout << std::setprecision(10);
std::cout << a << "\n";
```

Czy ten problem wystąpi dla zmiennej typu double?