1. Git - wszystko lokalnie

Git to system kontroli wersji stworzony przez Linusa Torvaldsa - twórcę jądra systemu linux. System kontroli wersji służy do kontroli zmian - najczęściej kodu.

> Proszę sobie wyobrazić następującą sytuację. Alicja tworzy zdanie `Ala ma kta`. W ramach wspólnej pracy jednocześnie Bob i Charlie chcą poprawić to zdanie. Bob poprawia na `Ala ma kota` zaś Charlie na `Ala ma tygrysa szablozębnego`. Pojawia sie problem organizacyjny - które zdanie jest tym poprawnym? Kto wprowadził zmiany, jakie, kiedy i dla czego?

System kontroli wersji pozwala opanować takie sytuacje - a także znacznie bardziej skomplikowane, występujący przy wspólnej pracy nad kodem (lub tekstem, jak w przypadku tych instrukcji).

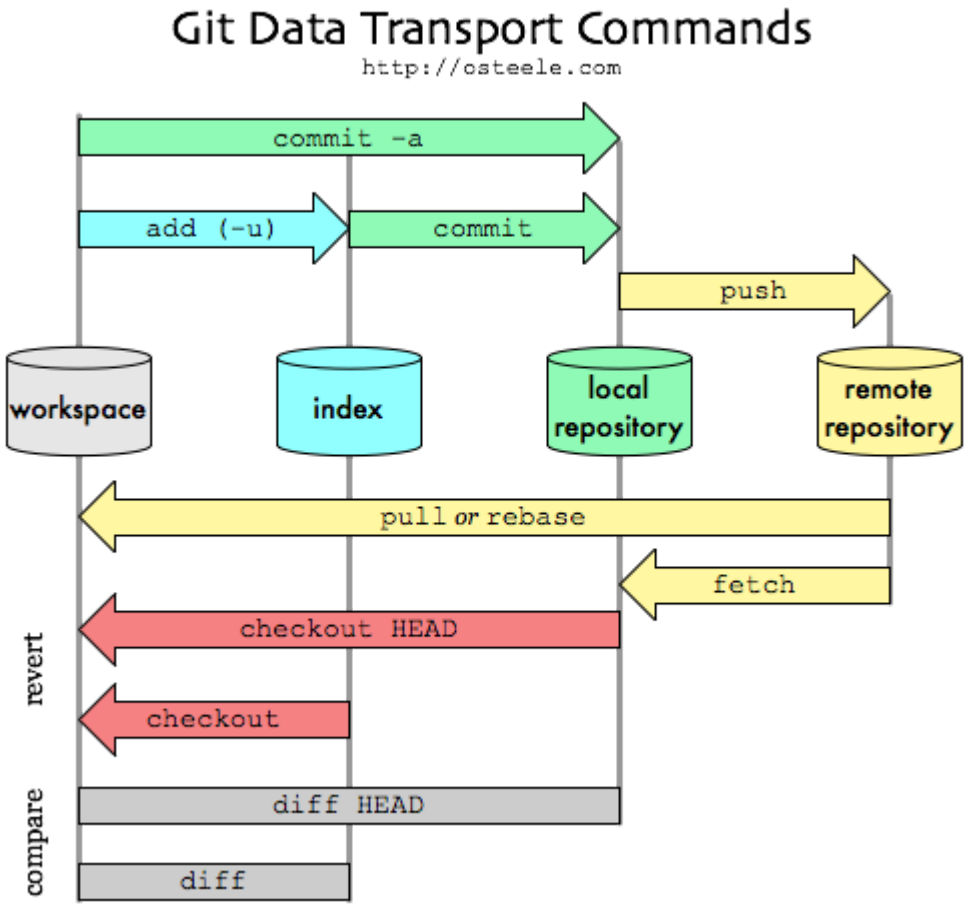
Stosowanie systemów kontroli wersji niesie ze sobą pewne wymagania. Krzywa uczenia (trudność) gita jest dość stroma - pewne aspekty nie są całkowicie intuicyjne, szczególnie jeśli nie zrozumie się istoty repozytorium rozproszonego. Git słabo nadaje się do przechowywania plików binarnych (programów). Ze względu na przechowywanie kompletu historii plików repozytorium git może zajmować bardzo dużo miejsca na dysku. Przykładowo - w chwili pisania tej instrukcji pliki aktualne zajmują 1,7MB, zaś całe repozytorium lokalne - aż 5MB.

Aby zacząć używać gita należy go zainstalować. Jest to aplikacja wieloplatformowa działająca bardzo dobrze za równo pod Windowsem jak i w systemach linuxowych. Także środowiska programistyczne potrafią integrować się z gitem. W trakcie tego przedmiotu polecamy jednak używanie gita z linii poleceń - pozwala to na utrwalenie poleceń i dobrą obserwację działań na repozytorium. Git dla Windows można pobrać tutaj zaś git dla Ubuntu można zainstalować poleceniem `sudo apt-get install git`.

Pierwszym krokiem jest konfiguracja - należy skonfigurować swojego maila i nazwę użytkownika poleceniami `git config --global user.name "Imię Nazwisko"` oraz `https://goo.gl/maps/FBxmwzbLdksgit config --global user.email nazwisko@imie.com`. Pomocne może być też także `git config --global push.default simple` - konfiguruje metodę wysyłania danych do repozytorium zdalnego oraz `git config --global credential.helper "cache --timeout=3600"` które sprawi, że dane logowania będą pamiętane przez 3600 sekund.

> Uwaga - ze względu na szczególną konfigurację maszyny w laboratorium mogą nie pamiętać konfiguracji i wymagać każdorazowego podawania konfiguracji. Ambitni (i leniwi) mogą sobie napisać skrypt w bashu 😉

Git jest rozproszonym systemem wersjonowania. Oznacza to, że każdy (także nasz) węzeł posiada pełną kopię repozytorium - wszystkie dane są przechowywane lokalnie w każdym z węzłów. Uświadomienie sobie tego, że w systemie git nie ma centralnego, uprzywilejowanego serwera zdecydowanie ułatwia zrozumienie istoty tego rozwiązania. Workflow pracy z git przedstawia poniższy rysunek:



`Workspace` to nasza przestrzeń pracy, czyli pliki, z którymi aktualnie pracujemy. `index` to pliki, które chcemy wersjonować. `local repository` to dane, które mamy w swoim lokalnym repozytorium, zaś `remote repository` to zdalne repozytorium, najczęściej używane do wymiany danych z innymi. Strzałki reprezentują polecenia służące do przenoszenia danych między poszczególnymi przestrzeniami.

Poza dostępem przy użyciu klienta git wiele rozwiązań oferuje interfejs WWW pozwalający na przeglądanie repozytorium.

Aby rozpocząć pracę z zainstalowanym gitem należy albo zainicjować lokalne repozytorium albo sklonować już istniejące repozytorium poleceniem `git clone <url>`. URL repozytorium można znaleźć logując się do gita przez przeglądarkę.

Przed rozpoczęciem pracy należy sprawdzić, czy ktoś ze współpracowników nie zmienił czegoś w kodzie (albo czy my nie zaktualizowaliśmy treści instrukcji). Można to zrobić poleceniami `git fetch` (pobrane zostaną dane ze zdalnego repozytorium) a potem `git status` (workspace zostanie porównany z lokalnym, już odświeżonym repozytorium). Jeżeli są różnice - należy zrobić `git pull` by zaktualizować swój workspace.

Każdy nowy plik należy dodać do indeksu (`git add <nazwa pliku>`). Prościej `git add .` doda nowe pliki i usunie usunięte z indeksu. Dodany plik można commitować (`git commit` - uwaga - włączy się edytor vi w którym jesteśmy zobowiązani do podania komentarza do zmiany, commit nastąpi dopiero po wpisaniu komentarza i zamknięciu edytora) a następnie wysłać do zdalnego repozytorium (`git push`). Alternatywą do `git commit` jest `git commit -am <"komentarz">`. Takie polecenie commituje wszystkie pliki i od razu dodaje komentarz do commita. Komentarze nie powinny być dłuższe jak 3-4 słowa, by dobrze formatowały się w historii.

Zadania:

1. Zaloguj się przez przeglądarkę do swojego repozytorium git
2. Skonfiguruj git na swoim komputerze
3. Sklonuj swoje repozytorium lokalnie, w wybranym katalogu (da to od razu dostęp do instrukcji!)
4. Utwórz lokalnie plik zawierający jedną linię `1:Ala ma kota`
5. Sprawdź status swojego repozytorium
6. Dodaj ten plik do lokalnego repozytorium i commituj

2. Zasady formatowania kodu

> Przestrzeganie zasad formatowania kodu zwiększa jego czytelność i jest obligatoryjne w czasie laboratorium. Prowadzący NIE będą pomagać przy rozwiązywaniu problemów w niesformatowanym kodzie. Brak formatowania kodu na kolokwium skutkować będzie obniżeniem oceny. Czuj się ostrzeżony 😃

Istnieje wiele systemów, szkół i podejść do zasad formatowania kodu. Za pewnie Twoj przyszły pracodawca będzie miał własne. Na naszych zajęciach będą obowiązywały te zawarte w dokumencie Google C++ Style Guide. Dokument ten, poza samymi zasadami, zawiera argumentację za i przeciw pozwalającą lepiej zrozumieć poszczególne reguły.

1. Zapoznaj się z zasadami dotyczacymi długości linii kodu.

> Ciekawostka: Od 1928 IBM wprowadził karty perforowane, które kodowały linię długości 80 znaków w urządzeniach teletekstowych. Późniejsze terminale ekranowe także czesto były ograniczone do 80 znaków

2. Zapoznaj się z zasadami używania znaków spoza kodu ASCII. Jest to szczególnie istotne przy pisaniu komentarzy w języku polskim.

3. Zapoznaj się z zasadami tworzenia wcięć. Wcięcia są podstawą czytelności kodu.

> Ucząc się jakiegokolwiek języka programowania unikaj kopiowania kodu metodą ctrl-c ctrl-v. O ile jest ona na pewno szybsza - zdecydowanie spowalnia proces uczenia się. nawet przepisując cudzy kod - uczysz się. Kopiując - nie. Oczywiście zdecydowanie najlepiej jest pisać własny kod. W ramach tego laboratorium często pliki z kodem będą dołączone - przeanalizuj go dokładnie lub wręcz przepisz

Zadanie

1. Popraw załączony w pliku `Lab2.cc` kod zgodnie z powyższymi zasadami. Stanie się zdecydowanie czytelniejszy, choć być może jeszcze nie całkowicie zrozumiały. Sam program jest bardzo prosty - przypisuje wartości dwóm zmiennym i wypisuje na ekran wynik działania.

> Ten przykład WYJĄTKOWO zawiera nazwy zmiennych i komentarze w języku Polskim. Od teraz zmienne i komentarze proszę pisać po angielsku - bo takie będzie oczekiwanie przyszłych pracodawców.

3. Pierwszy program w języku C++

Zadanie:

1. Napisz program w języku C++ wyświetlający na ekranie napis `Hello World!`. Użyj edytora tekstowego vim. Użyj strumieni do wyświetlenia tekstu na ekranie.

> Są (przynajmniej) dwa sposoby na wyświetlanie tekstu na standardowym wyjściu (w naszym przypadku - na ekranie). Drugim, poza strumieniami, jest funkcja `printf`. W dokumencie Googla możesz poczytać, czemu stosowanie strumieni jest preferowane przed funkcją `printf`

2. `iostream` to nazwa pliku. Znajdź ten plik w systemie. Co zawiera?
3. Skompiluj ten program używając konsoli.
4. Uruchom ten program używając konsoli.
5. Zmień treść napisu.
6. Popełnij błąd w progarmie - usuń średnik kończący wiersz. Skompiluj program i zaobserwuj komunikat kompilatora.
7. Popełnij błąd w programie - usuń nawias zamykający blok `main()`. Skompiluj program i zaobserwuj komunikat kompilatora.
8. Skompiluj poprawiony program Lab2.cc (z poprzedniego rozdziału instrukcji) i wykonaj.
9. Spróbuj skompilować kod uyżywając polecenia `gcc` zamiast `g++`. Co się stanie? Jaka jest różnica między tymi poleceniami?

4. Linux na własnym komputerze

Linuxa można zainstalować na dwa sposoby

1. Jako instalację równoległą do istniejącego Windowsa (albo nawet zamiast). Zaletą tego rozwiązania jest szybkość działania - sytsem działa natywnie i bezpośrednio na sprzęcie. Wadą - potencjalne konflikty z itniejącym systemem operacyjnym

2. W wirtualnej maszynie. Zaletą tego rozwiązania, jest to, że można mieć uruchomione jednocześnie dwa systemy operacyjne - hosta - Windows i w nim gościa - Linux. Ta sekcja opisuje właśnie takie rozwiązanie. Zaproponowany zestaw narzedzi - oprogramowanie do wirtualizacji, dystrybucja Linuxa i inne są jedynie propozycją.

3. **Wymagania**. 20GB przestrzeni dyskowej, 4GB RAM, procesor dwurdzeniowy.

4. W biosie komputera włączyć wsparcie wirtualizacji (procesory INTEL)

5. Ściągnąć program VirtualBox i najnowszą dystrybucję Ubuntu 64bit lub 32bit w formacie iso

6. Zainstalować Virtualbox

7. Stworzyć nową wirtualną maszynę typu Ubuntu 64bit (jeśli dostępne są tylko 32 bit to oznacza że nia mamy wsparcia rozszezreń wirtualizacji procesora), przydzielić jej przynajmniej 2GB RAM. Jeśli mamy przyzwoitą kartę graficzną - można przydzielić 128MB pamięci graficznej i włączyć sprzętowe wsparcie grafiki.

8. Stworzyć nowy plik wirtualnego dysku twardego, min 20GB

9. Zamontować plik iso z dystrybucją Ubuntu w wirtualnym napędzie wirtualnej maszyny i ją uruchomić

10. Zainstalować Ubuntu zgodnie z poleceniami na ekranie
11. Po zalogowaniu się do wirtualnej maszyny z menu Virtualboxa wybrać polecenie "Urządzenia->zamontuj obraz płyty z dodatkami gościa".
12. Zainstalować Guest Additions
13. Z menu "Urządzenia" można włączyć wspólny schowek (będzie działało Ctrl-C Ctrl-V między Windowsem a Ubuntu) i "Przeciągnij i upuść".
14. W wirtualnej maszynie wykonać `sudo apt update` - zaktualizuje to listy dostępnych pakietów
15. W wirtualnej maszynie wykonać `sudo apt upgrade` - zaktualizuje to zainstalowane pakiety - tak aktualizuje się dwoma pleceniami całe zainstalowane oprogramowanie
16. W wirtualnej maszynie wykonać `sudo apt-get install build-essential eclipse git` - zainstaluje to kompilator C i C++, Eclipse i git.
17. W Eclipse wejść do menu Help->Intall New Software. Z menu "Work With" wybrać "All Avaliable Sites", w pole wyszukiwarki wpisać `C++` i zainstalować `C\C++ Development Tools` - to doda wsparcie dla C++ do Eclipse.
18. Sprawdzić Eclipse - File->New->C++ Project-> Typ :C++ Hello World Project, nazwa dowolona. Taki projekt powinien się skompilować i wykonać z poziomu Eclipe. Wszystkie pliki są trzymane w katalogu `~/workspace`