PODSTAWY INŻYNIERII OPROGRAMOWANIA

LABORATORIUM SYSTEMÓW KONTROLI WERSJI

System GIT - podstawy

wersja 0.3

przygotował:
dr inż. Radosław Adamus

HISTORIA WERSJI

DATA	WERSJA	AUTOR	OPIS
08.11.2013	0.1	Radosław Adamus	Pierwsza robocza wersja dokumentu
17.11.2013	0.2	Radosław Adamus	Uzupełnienie opisu, dodanie zadania “Praca zespołowa”, poprawki edycyjne, uzupełnienie informacji licencyjnych
23.11.2013	0.3	Radosław Adamus	Modyfikacje na podstawie doświadczeń pierwszego laboratorium: częstsze zatwierdzanie zmian, zwrócenie uwagi na kodowanie pliku, poprawki w linkach, poprawki edycyjne, zmiany nazw plików opisujących zmiany w przykładowym projekcie

Cel:

Celem laboratorium jest:
1. Zapoznanie się z funkcjonalnością podstawowych komend rozproszonego systemu kontroli wersji GIT oraz wprowadzenie do wykorzystania platformy GitHub.
2. Zrozumienie znaczenia kontroli wersji i potrzeby wykorzystywania systemu kontroli wersji w codziennej pracy.

Efekty:

Po ukończniu laboratorium student potrafi:
1. Inicjować repozytorium zdalne i lokalne, łączyć je ze soba oraz przesyłać informacje pomiędzy nimi.
2. Dodawać pliki do repozytorium i zatwierdzać w nim zmiany.
3. Tworzyć gałęzie, przełączać się pomiędzy nimi oraz scalać zmiany w nich wprowadzane.
4. Wykorzystać mechanizmy platformy GitHub do tworzenia wydań projektu oraz realizacji podstawowego scenariusza pracy zespołowej.

Wymagania wstępne:

Założone konto w systemie GitHub (https://github.com/).

Narzędzia:

Lokalnie zadania wykonywane są z wykorzystaniem linii poleceń systemu git-scm oraz graficznego narzędzia git-gui (np. TortoiseGit[1]). Zakłada się, że operacje wykonywane są z poziomu linii poleceń, natomiast narzędzie z interfejsem graficzne służy do wizualizacji grafu wersji (ang. revision graph).

Komendy, które na pewno wykorzystasz w trakcie zajęć:

- git add
- git checkout
- git clone
- git commit
- git config
- git branch
- git init
- git merge
- git push
- git rebase
- git remote
- git status

[1] Lub innego (TortoiseGit jest aplikacją systemu Windows), który potrafi wizualizować graf wersji (https://git.wiki.kernel.org/index.php/InterfacesFrontendsAndTools#Graphical_Interfaces).

Reguły wykonywania ćwiczeń laboratoryjnych:

1 Komentowanie zmian w trakcie ich zatwierdzania musi być znaczące(!). Wprowadzona informacja powinna określać czego dotyczy zmiana.
2. Repozytorium musi być tak skonfigurowane, aby zmiany były sygnowane nazwą użytkownika odpowiadającą imieniu i nazwisku studenta oraz jego adresem email.
3. Rezultat musi znaleźć się ostatecznie na koncie studenta w systemie GitHub. W razie braku zrealizowania wszystkich ćwiczeń w ramach dostępnego czasu zajęć należy, przed opuszczeniem laboratorium, zatwierdzić zmiany i przesłać bieżący stan repozytorium lokalnego do repozytorium zdalnego.
4. Historia zmian (wraz z komentarzami) będzie kontrolowana - musi odpowiadać strukturze zadań.
5. OPCJONALNIE: Podczas wykonywania zadań wyświetlaj strukturę grafu wersji (TortoiseGit -> Revision graph) i zapisuj jego stany dla poszczególnych zadań w postaci plików graficznych (File -> Save graph as). Ułatwi to zrozumienie reguł zarządzania wersjami.

Wstęp:

1. Czym jest kontrola wersji

Kontrola wersji (rewizji) to aspekt zarządzania konfiguracją (oprogramowania) związany z zarządzaniem zmianami w obrębie dokumentów elektronicznych.

2. Co to jest system kontroli wersji

System kontroli wersji to narzędzie oprogramowanie wspomagające proces kontroli wersji. Do jego funkcji należy przede wszystkim: zarządzanie składem (repozytorium) przechowującym wersje dokumentów oraz kontrolowanie dostępu do przechowywanych wersji zasobów. W kontekście kontroli wersji dostęp do zasobu może oznaczać pobranie określonej jego wersji, wprowadzenie nowej wersji czy też utworzenie nowej wersji poprzez połączenie zmian występujących w innych wersjach zasobu.

3. Co to jest rozproszony system kontroli wersji

W rozproszonym systemie kontroli wersji może istnieć wiele równoległych repozytoriów przechowujących wersje dokumentów. Oprócz zatwierdzania zmian w repozytorium, systemy takie udostępniają również mechanizmy synchronizacji zawartości repozytoriów.

Opis laboratorium:

0. Uwagi wstępne

Struktura laboratorium wydziela problem kontroli wersji (który jest związany z laboratorium) od projektu podlegającego kontroli wersji (który jest tylko przykładem). Z tego powodu zmiany, które należy wprowadzić w projekcie opisane są w osobnych plikach.

W ramach ćwiczeń każdy student korzysta z repozytorium zdalnego (publicznego), przechowywanego na platformie GitHub, oraz repozytorium lokalnego (prywatnego), znajdującego się na stacji roboczej. Większość działań związana zarządzania wersjami na poziomie lokalnym. Synchronizacja repozytorium lokalnego i zdalnego odbywa się tylko w określonych instrukcją momentach (i/lub na koniec zajęć).

I Część pierwsza - scenariusz pracy indywidualnej

1. Zdalna inicjalizacja

Utwórz repozytorium projektu w serwisie GitHub o nazwie iislabpio_[nr indeksu].

2. Lokalna inicjalizacja repozytorium projektu

Utwórz katalog projektu i dokonaj inicjalizacji repozytorium git w jego wnętrzu (git init).
Skonfiguruj lokalnie dla repozytorium swoją nazwę użytkownika (swoje imię i nazwisko) oraz adres email (git config).

Utwórz plik o nazwie ‘README.md’ i dodaj go do lokalnego repozytorium (git add). Dokonaj edycji pliku wpisując swój numer indeksu oraz grupę laboratoryjną. Zatwierdź (git commit) zmiany, pamiętając o komentarzach.
Połącz lokalne repozytorium (domyślna gałąź master) z repozytorium zdalnym (git remote).
Wyślij (git push) do zdalnego repozytorium lokalną gałąź master.

Struktura projektu i gałęzie robocze

Utwórz lokalnie początkową wersję projektu zgodnie z opisem w pliku *init.txt*. Przetestuj poprawność wprowadzonych zmian. Zatwierdź zmiany (git commit) pamiętając o komentarzach.
Utwórz gałęzie ‘cecha1’ i ‘cecha2’ (git branch).

Przełącz się na gałąź *cecha1* (git checkout) i wprowadź **dwie pierwsze** zmiany zgodnie z opisem w pliku *zmiany_cecha1.txt*. Zmiany wprowadzaj kolejno, zgodnie z punktami opisu, za każdym testując i zatwierdzając je w repozytorium.

Przełącz się na gałąź ‘*cecha2’* i wprowadź **dwie pierwsze** zmiany opisane w pliku *zmiany_cecha2.txt*, testując je kolejno i zatwierdzając (analogicznie jak dla *cechy1*).

Scalanie zmian w gałęzi master

W trakcie testowania zorientowałeś się że podstawowa struktura projektu zawiera błąd. Informacje te powinny znaleźć się w głównej gałęzi projektu, zanim ukończone zostaną prace nad poszczególnymi cechami. W związku z tym należy w pierwszym rzędzie wprowadzić zmiany w gałęzi master. Aby uniknąć rozsynchronizowania gałęzi należy również scalić zmiany

z gałęzi master z zawartością pozostałych gałęzi.

Przełącz się do gałęzi master i wprowadź zmianę zgodnie z opisem w pliku *bugFix_master.txt*. Przetestuj poprawność wprowadzonych zmian. Zatwierdź zmiany.

Przełącz się do gałęzi *cecha1* i scal zmiany z gałęzi master wykorzytując polecenie **git rebase**. Zaobserwuj komunikaty wyświetlane w trakcie wykonywania operacji. Z wykorzystaniem dostępnych narzędzi wyświetl graf wersji (rewizji).
Wykonaj scalanie dla gałęzi *cecha2.* Tym razem wykonaj operację za pomocą polecenia **git merge.** Zaobserwuj komunikaty wyświetlane w trakcie wykonywania operacji. Jakie są różnice w działaniu polecenia *merge* oraz *rebase*? Z wykorzystaniem dostępnych narzędzi wyświetl graf wersji (rewizji).

Finalizowanie prac w gałęziach cecha1 i cecha2

Uzupełnij implementację cechy1 oraz cechy2 w odpowiednich gałęziach zgodnie z pozostałym opisem w plikach *zmiany_cecha1.txt* oraz *zmiany_cecha2.txt*.

Scalanie zmian do gałęzi master

Praca nad sekcjami została zakończona i należy dokonać scalenia głównej gałęzi projektu (master) z zawartościami gałęzi *cecha1* oraz *cecha2*. Wykonaj scalanie (jedną z metod: rebase lub merge).
Przetestuj poprawność scalonej struktury. Z wykorzystaniem dostępnych narzędzi wyświetl graf wersji (rewizji). Czy graf będzie taki sam bez względu na wybraną metodę scalania?

Finalizowanie i upublicznianie zmian

Wprowadź (w gałęzi master) zmiany zgodnie z opisem w pliku *zmiany_master_part1.txt*. Przetestuj poprawność wprowadzonych zmian. Zatwierdź zmiany.
Wyślij rezultat do repozytorium zdalnego.
W ostatnim kroku wprowadź zmianę zgodnie z opisem w pliku *zmiany_master_part1.txt*. Przetestuj poprawność wprowadzonych zmian. Zatwierdź zmiany.
Ponownie wyślij poprawioną wersję do repozytorium zdalnego.
Za pomocą funkcji systemu GitHub utwórz wydanie (release) oznaczając go jako v0.1

II Część druga - scenariusz pracy zespołowej

[*Ćwiczenie wymaga wcześniejszego wykonania zadań części pierwszej laboratorium*]
Utwórz projekt na swoim koncie GitHub na podstawie projektu (komenda systemu GitHub: fork - https://help.github.com/articles/fork-a-repo) kolegi/koleżanki po twojej prawej stronie (jeżeli jesteś ostatnią osobą w rzędzie - osobą siedzącą po twojej prawej stronie jest pierwsza osobą w drugim rzędzie). Jeżeli osoba nie ukończyła jeszcze poprzednich zadań - najpierw jej pomóż.
Następnie dokonaj operacji klonowania tak utworzonego repozytorium zdalnego na komputer lokalny. Wprowadź w projekcie zmiany zgodnie z opisem w pliku *zmiany_fork.txt*. Przetestuj

poprawność wprowadzonych zmian. Zatwierdź zmiany. Wyślij rezultat do repozytorium zdalnego. Wyślij do kolegi/koleżanki informacje o zmianach (pull request).
Na podstawie otrzymanego “pull request” wprowadź zmiany, dokonane przez kolegę/koleżankę, do swojego repozytorium. Utwórz nowe wydanie oznaczając je jako v0.2.

Pytania uzupełniające

1. Jaką komendę systemu git wykorzystuje platforma GitHub do tworzenia wydań (release)?
2. W jaki sposób wycofać zmiany wprowadzone w pliku w kopii roboczej a nie zapisane w repozytorium (czyli przywrócić ostatnią wersję z repozytorium)?
3. W jaki sposób wycofać zmiany zapisane już w repozytorium tak aby najnowsza wersja gałęzi odpowiadała zmianom wprowadzonym N commitów wcześniej? Podaj dwie metody, różnice pomiędzy nimi oraz, wynikający z różnic, zakres stosowalności.
4. W jaki sposób cofnąć się do poprzedniej wersji w repozytorium? W jaki sposób cofnąć się o N wersji?

Interaktywne kursy:

http://try.github.com/
http://pcottle.github.io/learnGitBranching/

Materiały:

http://git-scm.com/book/pl/
http://githubtraining.s3.amazonaws.com/github-git-training-slides.pdf
http://ndpsoftware.com/git-cheatsheet.html
https://commons.lbl.gov/display/pbddocs/Learn+Git