PROGRAMIŚCI DLA SZCZECINA

Zachodniopomorska Szkoła Biznesu w Szczecinie

Kontrola wersji i Git

Mariusz Łazor / 09.09.2019

Fundusze Europejskie Program Regionalny

Rzeczpospolita Polska

Pomorze Zachodnie

Unia Europejska Europejski Fundusz Społeczny

Co to jest kontrola wersji?

- System kontroli wersji (VCS – Version Control System) śledzi wszystkie zmiany dokonywane na plikach
- Pliki są umieszczone w zdalnym repozytorium
- Umożliwia przywołanie dowolnej wcześniejszej wersji
- Pozwala odtworzyć stan całego projektu
- Porównać wprowadzone zmiany
- Dowiedzieć się kto dokonał zmian w projekcie
- Jeżeli popełnimy błąd lub utracimy część danych, to możemy przywrócić stan pierwotny
- System kontroli wersji jest jednym z narzędzi wykorzystywanych w procesie Continuous Integration

Git

- Git jest implementacją systemu kontroli wersji
- Utworzony w 2005 roku jako narzędzie wspomagające rozwój jądra Linux
- Jest darmowy
- Wspiera rozgałęziony proces wytwarzania oprogramowania (branches)
- Umożliwia pracę off-line (każdy ma lokalną kopię repozytorium)
- Wspiera różne protokoły sieciowe (np. HTTP, SSH)
- Efektywna praca z dużymi projektami (szybszy niż konkurencyjne rozwiązania, np. SVN)
- Każda rewizja to obraz całego projektu, a nie pojedynczego pliku

Konfiguracja Git

Obsługa konsoli Git Bash

- W konsoli używamy komend Linux CLI
- *ls* – wyświetlenie listy plików w danym katalogu
- *cd $katalog* – przejście do wskazanego katalogu
- *cd ..* – przejście do katalogu niżej
- *cd ../$katalog* - przejście do katalogu niżej i wejście w konkretny katalog
- *strzałka w górę/ w dół* - przywrócenie poprzedniego/ następnego polecenia z historii
- *klawisz Tab* - podpowiadanie komend
- *zaznaczenie tekstu i naciśnięcie klawisza Enter* - skopiowanie tekstu
- *prawy przycisk myszy* - wklejenie tekstu

Git – struktura

- *Katalog roboczy (working dir)* – lista przechowywanych plików
- *Index (Stage)* – tzw. poczekalnia – lista plików dodanych do indeksu, ale jeszcze „niezacommitowane”
- *HEAD* – ostatni utworzony commit

Inicjalizacja repozytorium

1. *git init* - inicjalizacja lokalnego repozytorium
2. dodanie plików i ich zacommitowanie
3. utworzenie nowego repozytorium w github
4. skopiowanie adresu do repozytorium (SSH lub https)
5. *git remote add origin $url* - powiązanie naszego lokalnego repozytorium ze zdalnym

Zatwierdzanie i wysyłanie zmian

- *git status* - wyświetlenie statusu naszych katalogów
- *git add $filename* – dodanie pliku do indeksu
- *git add .* – dodanie wszystkich plików do indeksu
- *git commit –m „Commit message"* – dodanie zmian do HEAD, ale tylko w naszym lokalnym repozytorium
- *git push origin $branch* – wysłanie HEAD do zdalnego repozytorium, jeżeli w repozytorium zdalnym są zmiany, których nie mamy lokalnie (wprowadzone przez innego użytkownika), to musimy je najpierw pobrać i dopiero wypchnąć swoje
- *git push -f* - wypchnięcie zmian z modyfikatorem force, co powoduje ignorowanie zmian w zdalnym repozytorium (czasami jest to jedyne wyjście, należy stosować z rozwagą!)

Tagowanie zmian

- Tagowanie zmian stosuje się w celu oznaczania tzw. kamieni milowych w historii repozytorium, np. nowych wersji aplikacji
- Domyślnie tagi nie są automatycznie wysysyłane przy użyciu komendy **git push**
- *git tag* - wyświetlenie listy tagów
- *git tag $tagName* - dodanie nowego tagu w lokalnym repozytorium
- *git tag -d $tagName* - usunięcie tagu w lokalnym repozytorium
- *git push origin $tagName* - wysłanie tagu do zdalnego repozytorium
- *git push --delete origin $tagName* - usunięcie tagu w zdalnym repozytorium

Ignorowanie plików

- Ignorowanie stosuje się po to, żeby Git nie śledził niektórych plików, których nie chcemy wysyłać do zdalnego repozytorium (np. katalog target dla projektów java)
- Informacje dotyczące ignorowania umieszcza się w pliku .gitignore - sam plik należy zacommitować
- *$fileName.$fileExtension* - ignorowanie konkretnego pliku
- **.$fileExtension* - ignorowanie wszystkich plików z konkretnym rozszerzeniem
- *!$fileName.$fileExtension* - uwzględnienie konkretnego pliku, którego rozszerzenie jest ignorowane (wyłączenie konkretnego pliku z listy ignorowanych rozszerzeń)
- *$directoryName/* - ignorowanie wszystkich plików w danym katalogu

Anulowanie zmian

- *git checkout $filename* - wycofuje zmiany w katalogu roboczym dla danego pliku, zmiany w *stage* i *HEAD* pozostaną nienaruszone
- *git reset --soft* - cofa nasz HEAD do konkretnego miejsca (zmiany z *HEAD* są wyciągane z powrotem do *stage*)
- *git reset --mixed* - usuwa zmiany dodane do indeksu ale zachowuje je w katalogu roboczym
- *git reset --hard* - usuwa wszystkie zmiany włącznie z katalogiem roboczym (jeżeli nie wypchnęliśmy zmian, to tracimy je bezpowrotnie!)
- *git fetch origin* + *git reset --hard origin/$branch* - usuwa wszystkie lokalne zmiany, pobiera historię ze zdalnego repozytorium i ustawia na niej naszą lokalną gałęź

Monitorowanie zmian

- *git log* – przegląd historii zmian w repozytorium
- *git log -i* - przegląd historii repozytorium do wskazanego miejsca (*git log -1* pokaże nam tylko ostatni commit)
- *git log --author=$author* – historia zmian konkretnego użytkownika
- *git log --pretty=oneline* – każda zmiana jest wyświetlana w jednej linii
- *git log --graph --oneline --decorate –all* – prezentacja historii wraz ze strukturą drzewiastą
- *gitk* – graficzna prezentacja zmian w repozytorium

Gałęzie (branching)

- Gałęzie służą do rozwijania projektu w odizolowaniu od głównej gałęzi master
- Zmiany utworzone w innych gałęziach są później scalane do gałęzi głównej
- *git branch* - wyświetlenie listy gałęzi w lokalnym repozytorium
- *git branch -a* - wyświetlenie gałęzie w lokalnym i zdalnym repozytorium
- *git branch $branch* - utworzenie nowej gałęzie bez przełączania się na nią
- *git checkout –b $branch* – utworzenie nowej gałęzi i przełączenie się na nią
- *git checkout $branch* – przełączenie się na inną gałąź
- *git branch –d $branch* – usunięcie gałęzi w lokalnym repozytorium
- *git push --set-upstream origin $branch* – wysłanie gałęzi do zdalnego repozytorium i ustawienie śledzenia
- *git push --delete origin $branch* – usunięcie gałęzi w zdalnym repozytorium

Schowek

- Tzw. schowek (*stash*) stosuje się, kiedy pracujemy nad jedną gałęzią i potrzebujemy przełączyć się na inną, a nie chcemy commitować naszych zmian (ponieważ np. nie są jeszcze skończone)
- Do schowka zapisywane są zmiany w katalogu roboczym i indeksie
- Możliwe jest utrzymywanie kilku schowków na raz
- *git stash* - dodanie zmian do schowka
- *git stash list* - wyświetlenie stanu schowka
- *git stash apply $stashName* - nałożenie zmian z konkretnego schowka, nie podając nazwy Git spróbuje nałożyć zmiany z najnowszego (stash@{0})
- *git stash clear* - wyczyszczenie całego schowka
- *git stash drop $stashName* - usunięcie ze schowka pojedynczego obiektu, użycie samego *git stash drop* spowoduje usunięcie obiektu ostatnio dodanego do schowka

Aktualizacja i scalanie

- *git fetch* – pobranie zmian w zdalnym repozytorium (zmiany nie są umieszczane w HEAD) – rekomendowane jest korzystanie z atrybutu *–p*, który usuwa z listy zdalne branche, które już nie istnieją oraz dodaje nowo powstałe, zwykły fetch wykryje tylko nowo powstałe gałęzie i zmiany w już istniejących
- *git pull origin $branch* – pobranie zmian i umieszczenie ich w naszym *HEAD* – polecenie wykorzystywane w celu aktualizacji naszego lokalnego brancha względem zdalnego (w rzeczywistości jest to *git fetch* + *git merge*)
- *git merge $branch* – scalenie gałęzi poprzez utworzenie tzw. „commita mergującego" – zmiany, które mergujemy do naszej gałęzi są nakładane na *HEAD*
- *git rebase $branch* – zmiana commita bazowego
- *git pull --rebase origin $branch* – pobranie historii ze zdalnej gałęzi i potraktowanie jej jako bazy naszej aktualnej gałęzi, przydatne przy tzw. równaniu brancha z masterem
- *git cherry-pick $rev* – skopiowanie commita do aktualnego brancha
- *git rebase –i* – interaktywny *rebase*, wykorzystywany np. przy scalaniu kilki commitów w jeden (squash)
- *git rebase --abort* - anulowanie polecenia *rebase* (np. kiedy wystąpią nam konflikty i chcemy przerwać proces)
- *git rebase --continue* - kontynuowanie procesu *rebase* (używamy tego polecenia, kiedy wystąpią konflikty, rozwiążemy je i chcemy kontynuować proces)