Mercurial

Grzegorz Murdzek

Wstęp do kontroli wersji

Tworząc dowolne oprogramowanie czy nawet zwykłe pliki tekstowe z notatkami, które są poddawane wielokrotnym modyfikacjom, prędzej czy później ciężko jest zapanować nad tym, kiedy i jakie zmiany były wprowadzane. Można oczywiście po dodaniu nowej cechy do programu archiwizować jego katalog i co jakiś czas zapisywać w osobnym katalogu, ale wtedy ciężko jest śledzić czy porównywać które zmiany i w jakich plikach projektu były dodawane w wybranym czasie, a sprawa komplikuje się, gdy nad projektem zaczyna pracować więcej niż jedna osoba. Aby ułatwić zarządzanie i wymianę zmian do tworzonego dzieła, łatwiej i wygodniej jest skorzystać z narzędzia, które jest zwane systemem kontroli wersji.

Wśród dostępnych od dawien dawna na Amigę programów tego typu są dwa leciwe: CVS i Subversion. Są to tak zwane scentralizowane systemy kontroli wersji, co oznacza, że każdy z nich potrzebuje postawienia serwera, do którego dopiero będą wysyłane zmiany. To dodatkowy narzut i wymaganie od użytkownika przeznaczenia dodatkowego czasu na konfigurację projektu. Zupełnie inną konwencję reprezentuje Mercurial, który należy do narzędzi umieszczonych w kategorii rozproszonych systemów kontroli wersji (DVCS). Do ciekawszych zalet tego podejścia można zaliczyć nie tylko możliwość lokalnego oznaczania zmian (wersjonowania) w plikach projektu bez konieczności wysyłania ich w inne miejsce, ale też na przykład tworzenie na swoje potrzeby wielu gałęzi zmian w projekcie, co może przydać się, gdy utrzymujemy więcej niż jedną wersję tego samego programu lub gdy jedna z gałęzi zmian dotyczy wydań stabilnych a inne są eksperymentami lub zmianami, które dołączą do głównego programu, gdy będą ukończone. Inaczej mówiąc - Mercurial umożliwi nam tworzenie swego rodzaju dziennika zmian.

Dlaczego akurat Mercurial?

Atuty tego systemu kontroli wersji można streścić w kilku podpunktach: dostępny dla systemów AmigaOS 4 jak i MorphOS, napisany głównie w języku Python, co zapewnia mu dużą przenośność między platformami, narzędzie posiada wiele cech wspólnych z konkurencyjnym Gitem, ale ma też i swoje dodatkowe atuty (niektóre z cech Mercuriala skłoniły Facebooka do przejścia do niego z Gita), choć nie zawiera domyślnie niektórych zalet konkurenta, które przydają się od czasu do czasu, szybkość działania przy każdym rozmiarze projektu, łatwość obsługi, rozszerzalny (rozszerzenia można napisać w Pythonie), dostępny na licencji GNU GPL2.

Warto nadmienić, że na amigowe systemy nie ma póki co GUI do Mercuriala, więc wszystkie polecenia należy wykonywać z poziomu amigowego shella (co wcale nie jest takie złe, jak mogłoby się pozornie wydawać, gdyż mamy w taki sposób lepszy wgląd w komunikaty kontroli wersji).

Jak zacząć?

Wiemy już, w skrócie, czym jest Mercurial, ale w jaki sposób z niego skorzystać? Po pobraniu i instalacji mamy do dyspozycji polecenie o krótkiej i niewiele mówiącej nazwie hg. To jest właśnie główny plik wykonywalny Mercuriala - polecenie którym sterować będziemy całą kontrolą wersji w projekcie. Pierwszą czynnością jaką proponuję zrobić to utworzenie repozytorium, czyli właściwie podkatalogu, do którego Mercurial będzie zapisywał wszelkie wprowadzone w projekcie a zatwierdzone przez nas odpowiednim poleceniem zmiany.

W konsoli przechodzimy do katalogu, w którym jest podkatalog z plikami. Przyjmijmy, że ów katalog ma nazwę "HelloWorld" i znajduje się na dysku DH1: w katalogu "Projekty". Jeśli chcemy jego zawartość objąć wersjonowaniem, wykonujemy polecenie, które utworzy repozytorium, np.:

Mercurial

Grzegorz Murdzek

cd DH1:Projekty/ hg init HelloWorld

Następnie przechodzimy do podkatalogu projektu, w którym dodajemy znajdujące się w nim pliki do repozytorium i zapisujemy je z odpowiednim komentarzem:

cd HelloWorld hg add hg commit -m "Inicjalizacja repozytorium" -u Grzegorz

* parametr -u oznacza nazwę użytkownika, który wprowadza zmianę. Jeśli nie chcesz za każdym razem podawać nazwy, dodaj do pliku .hgrc z podkatalogu .hg w projekcie dodaj dwie linijki:
[ui] username = Imię Nazwisko

* katalog .hg zawiera repozytorium projektu oraz lokalne konfiguracyjne, do których należy właśnie .hgrc.

W ten sposób mamy już w repozytorium pierwszą wersję naszego projektu! Co dalej? Czy system sam będzie wykrywał zmiany i robił cokolwiek automatycznie? Na szczęście nie - o tym decydujemy sami. Załóżmy, że w podkatalogu projektu mamy plik o nazwie "readme.txt" i teraz chcemy zmienić jego zawartość. Przed zmianą zawartość pliku wyglądała tak:

Short: BitTorrent client Type: comm/tcp Version: 0.1

Zmieniamy plik przy okazji uzupełniając o dodatkowe informacje:

Short: BitTorrent client Author: Grzegorz M. Type: comm/tcp Version: 0.2 Architecture: ppc-morphos >= 3.7

Załóżmy, że chcemy zachować w dzienniku zmian tylko tę zmianę w jednym pliku. Sprawdźmy najpierw czy Mercurial zauważy zmianę jakiej przed chwilą dokonaliśmy. W tym celu wykonujemy w katalogu projektu polecenie:

hg status

Wynikiem powinna być jednoelementowa (gdyż zmieniany był tylko jeden plik) lista zmienionych plików przedstawiona w taki sposób:

M readme.txt

"M" to jeden ze statusów, który oznacza, że plik który został wcześniej dodany do repozytorium został zmodyfikowany.

Jeśli lista modyfikacji to dla nas za mało, możemy wyświetlić porównanie zmian stosując polecenie hg diff, które wyświetli bardziej szczegółowe dane o zmianach w stosunku do wersji wcześniej zapisanej w repozytorium:

diff -r f7136a68d033 readme.txt --- a/readme.txt Tue Apr 14 22:40:20 2015 +0200 +++ b/readme.txt Wed Apr 15 00:30:58 2015 +0200 @@ -1,3 +1,5 @@ Short: BitTorrent client +Author: Grzegorz M. Type: comm/tcp -Version: 0.1 +Version: 0.2 +Architecture: ppc-morphos >= 3.7

Na pierwszy rzut oka wyglądać to może nieco dziwnie. Jednak jeśli przyjrzymy się bliżej, widać np. które linie zostały oznaczone jako usunięte (-), a które dodane (+), czyli oznaczenia miejsc, w których nastąpiła zmiana.

Mercurial

Grzegorz Murdzek

Zarówno hg add, hg diff jak i hg status można używać również wskazując konkretny plik - wystarczy po nazwie polecenia, po spacji podać ścieżkę do pliku

Zatwierdzamy modyfikacje w pliku wykonując polecenie:

```
hg commit -m "Dodanie brakujących informacji w pliku readme.txt"
```

Oczywiście nie musimy wykonywać polecenia commit po modyfikacji tylko jednego pliku. Możemy to zrobić w dowolnym momencie i dla dowolnej liczby zmian w plikach, ale dobrą praktyką jest, aby wykonywać commit dla zmian, które już są gotowe lub uznamy, że powinny być zapisane, aby można było ewentualnie cofnąć się do nich w najbliższej lub nieokreślonej przyszłości.

Warto wiedzieć, że każda zmiana jest przyporządkowana w systemie kontroli wersji do konkretnej gałęzi (branch). Na początku nie musimy sobie tym zaprzątać głowy, gdyż domyślnie działamy na gałęzi o nazwie default, a kolejne możemy stworzyć później, jeśli tylko będzie taka potrzeba.

Ocean możliwości

Możliwości, poleceń i rozszerzeń jakich dostarcza Mercurial jest zbyt wiele, aby zmieścić je w jednym artykule, dlatego najlepszą drogą do zgłębienia wiedzy jest chociażby dostępna na oficjalnej stronie www projektu dokumentacja. Niemniej jednak warto poznać już teraz, poza wymienionymi wyżej, jeszcze kilka innych, podstawowych poleceń tego narzędzia. Na przykład:

hg log -l 5 - wyświetlenie listy ostatnich zmian (changesetów) hg pull - pobranie zmian z innego repozytorium (może być zarówno zdalne jak i lokalne - inny katalog na dysku) hg push - podobnie jak w przypadku hg pull, tylko do wysyłania zmian hg update - pobranie plików z lokalnego repozytorium (zapisanie lub nadpisanie plików w katalogu) hg branch - informacja o tym na jakim branchu aktualnie pracujemy hg merge - łączenie zmian między rewizjami (listami zmian) hg remove - usunięcie pliku z repozytorium hg clone - pobranie zdalnego repozytorium hg forget - nie uwzględnia wskazanych plików przy następnym hg commit

Dobre rady i praktyki

Dobrymi praktykami w pracą z systemem kontroli wersji są: stosowanie języka angielskiego do opisywania zmian, unikanie tytułowania zmian jako "update fix" czy "another bugfix", gdyż niewiele one mówią i trudniej jest później odnieść się w historii doszukując się konkretnej zmiany, oddzielanie większych zmiany, które mogą wpłynąć na stabilność i działanie aplikacji do osobnych gałęzi - warto robić, gdy mamy do zrobienia większe poprawki lub refactoring kodu, w przypadku małych poprawek warto zapoznać się z rozszerzeniem hg shelve, które jest odpowiednikiem Gitowego git stash i służy do przechowywania zmian w "schowku" systemu kontroli wersji, w którym przechowujemy inne zmiany, do których wracamy po dokonaniu tych mniejszych, warto wymieniać (wysyłać, pobierać) zmiany ze zdalnym repozytorium (hg push), w sieci są darmowe serwisy takie jak bitbucket.org, który oferują darmowe, prywatne repozytorium do 5 osób w projekcie, a także interfejs webowy, którego możemy użyć również do przeglądania zmian i zarządzania uprawnieniami, nie wszystkie pliki w katalogu należy wersjonować, a na pewno nie warto do repozytorium dodawać plików tymczasowych lub konfiguracji środowiska lokalnego, aby ustrzec się przed przypadkowym dodaniem niechcianych plików można je dodać np. do pliku .hgignore, który tworzymy w katalogu z projektem - można tam wskazywać konkretne pliki lub wzorce (maski), Mercurial nie wersjonuje pustych katalogów - jeśli chcemy by katalog trafił do repozytorium, należy utworzyć w nim choćby jeden pusty plik

Mercurial

Grzegorz Murdzek

Wady amigowych portów

Jedną z nich jest brak GUI. Na innych systemach, takich jak Windows, OSX czy Linux istnieje chociażby TortoiseHg. Nie ma co liczyć, że na amigowe systemy ktoś zrobi cokolwiek podobnego. Z innych wad, jakie zauważyłem, to na przykład: niedziałający wbudowany serwer do przeglądania zmian, który jest dostępny po wykonaniu polecenia hg serve (choć możliwe, że była to wina dystrybucji Pythona jakiej używałem do testów), brak ukrywania hasła zdalnego użytkownika w konsoli, gdy pobieram zdalne repozytorium (jeśli nie zapisałem hasła w konfiguracji).

Dla kogo?

W Mercurialu można przechowywać dowolne pliki, ale tylko pliki reprezentujące tekst będą wersjonowane w taki sposób, aby można było porównywać pomiędzy wybranymi zmianami. Jest to narzędzie ogólnego przeznaczenia, więc nawet jeśli nie jesteś programistą, możesz skorzystać z jego właściwości - nawet jeśli byłaby to choćby forma robienia kopii zapasowej pliku w celu późniejszego, ewentualnego przejrzenia w nim zmian. Na pewno zarządzanie wersjami nie jest domeną ani narzędzi korporacyjnych czy też sensowne jedynie przy dużych projektach, nad którymi pracuje wiele osób. Kontrola wersji, zwłaszcza tak podana jak w Mercurialu, może być wybawieniem dla niektórych problemów w utrzymywaniu żywych projektów poddawanych częstym modyfikacjom. Zachęcam do zapoznania się z tym narzędziem, poznania jego dodatkowych możliwości a także historii powstania i różnic, jakie są na przykład między nim a Gitem. W artykule zaledwie wspomniałem o gałęziach (branches) a warto wiedzieć, że to one są znacznie lepiej przemyślane i obsługiwane niż w systemach niescentralizowanych takich jak SVN czy CVS.

Skąd pobrać?

AmigaOS 4
MorphOS

Sonda

Ponieważ kontrola wersji jest używana zwykle przez programistów postanowiłem sprawdzić czy i w jaki sposób amigowi programiści w Polsce wersjonują kod swoich aplikacji. Wśród ankietowanych niewiele ponad połowa odpowiedziała "tak" na pytanie [i]"Czy używasz systemu kontroli wersji do wersjonowania kodu swoich aplikacji?"[/i] Podobny wynik, choć bardziej rozłożony w szczegółach co do stosowanych narzędzi był przy drugim pytaniu [i]"Jakiego systemu kontroli wersji używasz na systemie AmigaOS, MorphOS lub AROS?"[/i]. Niektórzy ankietowani nie używają systemu kontroli wersji do projektów prywatnych, w tym amigowych - inni, korzystają nawet dla projektów o niewielkim stopniu złożoności, które realizują czy to samodzielnie czy w zespole. Wśród narzędzi używanych na systemach amigowych ankietowani najczęściej wymieniali Subversion (SVN) i CVS a jedna osoba zadeklarowała używanie Fossil (port na własny użytek).

Słowniczek pojęć

changeset - to identyfikator grupy zmian nadawany po każdym dokonanym poleceniu hg commit do repozytorium, który jest reprezentowany przez unikalny 40-znakowy numer zapisany w systemie szesnastkowym, który może się przydać później np. w celu porównania zmian z przeszłości
repozytorium - zbiór wszystkich modyfikacji i historii projektu zawierający strukturę katalogów i plików oraz m.in. gałęzi zmian