Apress

Git

dla profesjonalistów

Wydanie II

Scott Chacon, Ben Straub

Helion

Tytuł oryginału: Pro Git, 2nd Edition

Tłumaczenie: Wojciech Moch

ISBN: 978-83-283-8731-7

First published in English under the title Pro Git by Scott Chacon and Ben Straub, edition: 2

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/gitpr2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

ROZDZIAŁ 1.

Zaczynamy

W tym rozdziale zaczniemy pracować z systemem Git. Na początek omówimy podstawy działania systemów kontroli wersji, następnie przystąpimy do uruchamiania Gita w swoim systemie, a zakończymy odpowiednim skonfigurowaniem go i przygotowaniem do właściwej pracy. Po przeczytaniu tego rozdziału będziesz wiedzieć, dlaczego w ogóle używamy Gita i dlaczego warto z niego korzystać. Będziesz też w stanie przygotować sobie działający system.

Systemy kontroli wersji

Czym właściwie są „systemy kontroli wersji" i dlaczego musimy się nimi zajmować? Każdy z takich systemów zajmuje się zapamiętywaniem zmian wprowadzanych do pliku albo zbioru plików, co pozwala na późniejsze odwoływanie się do poszczególnych wersji. W przykładach prezentowanych w tej książce będziemy korzystać z plików zawierających kod źródłowy programów, choć w rzeczywistości można tu używać niemal dowolnego rodzaju plików, jakie znajdziesz w swoim komputerze.

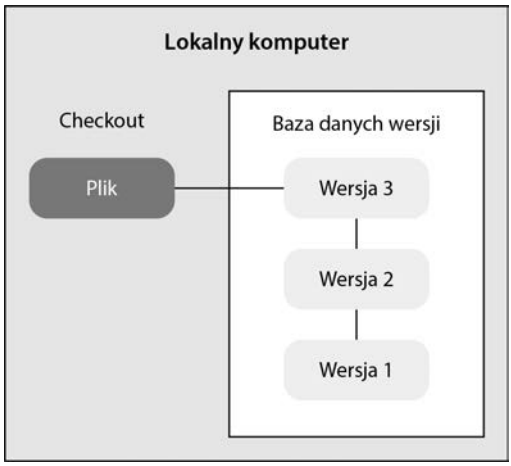
Jeżeli pracujesz jako grafik lub projektant stron WWW i chcesz przechowywać wszystkie wersje tworzonego obrazka lub strony (a na pewno warto to robić), zainteresuj się jednym z systemów kontroli wersji (ang. *Version Control System — VCS*). Każdy z tych systemów pozwala na przywrócenie wcześniejszych wersji pliku albo nawet całego projektu, umożliwia przeglądanie zmian wprowadzanych w kolejnych wersjach, sprawdzanie, kto ostatnio wprowadzał modyfikacje, które mogą powodować różne problemy i wiele innych. Kiedy korzystamy z systemu VCS, to nawet w przypadku utraty plików albo ich zepsucia zawsze możemy w łatwy sposób odzyskać poprawną postać. Co ważne, wszystkie te funkcje wymagają od nas tylko niewielkiego zaangażowania.

Lokalne systemy kontroli wersji

Wiele osób prowadzi własny system kontroli wersji polegający na kopiowaniu plików do innego katalogu (w bardziej zaawansowanych rozwiązaniach takie katalogi oznaczane są

datą i czasem). To popularne rozwiązanie, ponieważ jest bardzo proste, choć jednocześnie jest też niezwykle podatne na różnorakie błędy. Można przecież łatwo pomylić się przy kopiowaniu i przez przypadek zapisać niewłaściwy plik albo nadpisać istniejące już pliki.

Programiści już dawno poradzili sobie z tym problemem, tworząc lokalne systemy kontroli wersji, które budują prostą bazę danych przechowującą wszystkie zmiany wprowadzane do plików objętych kontrolą (rysunek 1.1).



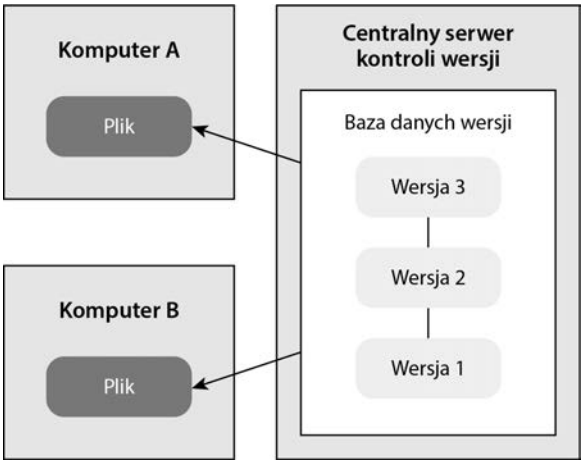
Rysunek 1.1. *Lokalna kontrola wersji*

Jednym z popularniejszych systemów kontroli wersji był system o nazwie RCS, który nadal można znaleźć na wielu komputerach. Nawet w popularnym systemie operacyjnym Mac OS X po zainstalowaniu narzędzi dla programistów dostępne jest polecenie `rcs`. System RCS przechowuje na dysku zestawy poprawek (czyli różnic między plikami) zapisane w specjalnym formacie. Zatem z jego pomocą można odtworzyć zawartość dowolnego pliku w wybranej wersji, po prostu nakładając kolejne poprawki na pierwotną postać pliku.

Scentralizowane systemy kontroli wersji

Kolejnym problemem, z jakim musimy się zmagać, jest konieczność współpracowania z innymi programistami, używającymi innych systemów i komputerów. Z tym problemem poradzono sobie za pomocą scentralizowanych systemów kontroli wersji (ang. *Centralized Version Control Systems — CVCS*). W tych systemach (należą do nich CVS, Subversion albo Perforce) istnieje specjalny serwer przechowujący wszystkie pliki poddawane wersjonowaniu, a dowolna liczba klientów może pobierać te pliki z serwera (rysunek 1.2). Przez wiele lat ten rodzaj systemów był standardem w rozwiązaniach kontroli wersji.

Ten rodzaj systemów ma wiele zalet, szczególnie w porównaniu z systemami lokalnymi. Przykładowo do pewnego stopnia każdy wie, nad czym pracują pozostali członkowie zespołu projektowego. Administratorzy mają możliwość dokładnego określenia uprawnień poszczególnych użytkowników, a samo administrowanie systemem CVCS jest znacznie łatwiejsze niż obsługiwanie lokalnych baz danych znajdujących się na wszystkich komputerach klienckich.



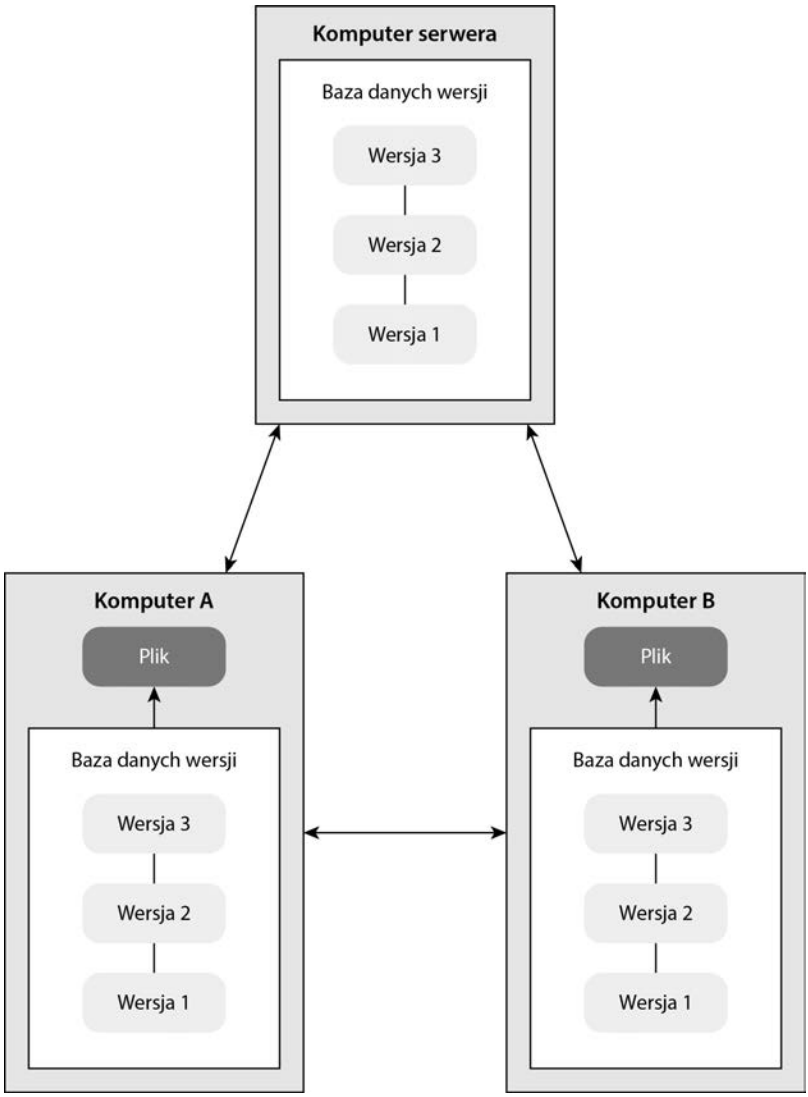
Rysunek 1.2. *Scentralizowany system kontroli wersji*

Niestety takie systemy mają też dość poważne wady. Najbardziej oczywistą jest pojedynczy punkt awarii, którym jest sam centralny serwer. Jeżeli serwer nie będzie działał przez godzinę, przez ten czas nikt nie będzie w stanie zapisać zmienionych przez siebie plików ani pobrać zmian wprowadzonych przez innych. Jeżeli awarii ulegnie dysk twardy centralnego serwera, a nikt nie zadbał o wykonywanie odpowiednich kopii zapasowych, tracimy wszystko — całą historię projektu. Pozostają jedynie pliki, jakie poszczególni użytkownicy mają na swoich lokalnych komputerach. Z podobnym problemem zmagają się też lokalne systemy kontroli wersji. Jeżeli historia projektu jest zapisana w jednym miejscu, istnieje poważne ryzyko utraty wszystkich informacji o tym projekcie.

Rozproszone systemy kontroli wersji

I tu przechodzimy do rozproszonych systemów kontroli wersji (ang. *Distributed Version Control Systems — DVCS*). W tym typie systemów (należą do nich Git, Mercurial, Bazaar albo Darcs) klienty nie pobierają jedynie ostatniej postaci każdego pliku, ale wykonują pełną kopię całego repozytorium. Jeżeli zatem dowolny serwer ulegnie awarii, całe repozytorium może zostać skopiowane z jednego z klientów. W ten sposób stan serwera może zostać łatwo odtworzony. Oznacza to jednak, że przy każdym pobieraniu danych z serwera musimy wykonać pełną kopię zapasową przechowywanych na nim informacji (rysunek 1.3).

Co ważne, każdy z tych systemów świetnie radzi sobie z koniecznością pracy z kilkoma zdalnymi repozytoriami, co oznacza, że możemy w różny sposób współpracować z niezależnymi zespołami w ramach jednego projektu. Ten rodzaj systemów umożliwia stosowanie kilku różnych schematów wykonywania pracy, co nie jest możliwe w scentralizowanych systemach.



Rysunek 1.3. *Rozproszony system kontroli wersji*

Krótka historia Gita

Jak wiele innych wartościowych rzeczy, Git powstał w wyniku kreatywnej destrukcji oraz wielkich kontrowersji.

Jądro systemu Linux jest bardzo rozbudowanym projektem oprogramowania o otwartych źródłach. Przez dużą część czasu istnienia tego projektu (lata 1991 – 2002) zmiany w oprogramowaniu były przesyłane w postaci poprawek oraz plików archiwów. W 2002 roku projekt jądra systemu zaczął korzystać z własnościowego, rozproszonego systemu kontroli wersji o nazwie BitKeeper.

W 2005 roku związki łączące społeczność zajmującą się rozwojem jądra Linuksa oraz firmą rozwijającą system BitKeeper zostały zerwane, a za korzystanie z narzędzia dostępnego dotychczas bez opłat zaczęto żądać zapłaty. W efekcie cała społeczność Linuksa (a szczególnie Linus Torvalds — twórca tego systemu) przystąpiła do tworzenia własnego narzędzia, wykorzystując doświadczenia wyniesione z pracy z systemem BitKeeper. Twórcy nowego systemu chcieli osiągnąć następujące cele:

- szybkość,
- prosty projekt,
- dobrą obsługę nieliniowego rozwoju (tysięcy równoległych rozgałęzień),
- całkowite rozproszenie,
- możliwość skutecznego obsługiwania wielkich projektów, takich jak projekt jądra Linuksa (z naciskiem na szybkość i wielkość danych).

Od powstania w 2005 roku Git cały czas ewoluuje i dojrzewa, starając się zachowywać początkowe założenia i umożliwiać jak najprostszą obsługę. Nadal jest niezwykle szybki, bardzo sprawny w pracy z wielkimi projektami i udostępnia niesamowity system tworzenia odgałęzień pozwalający na nieliniowy rozwój projektu (więcej na ten temat w rozdziale 3.).

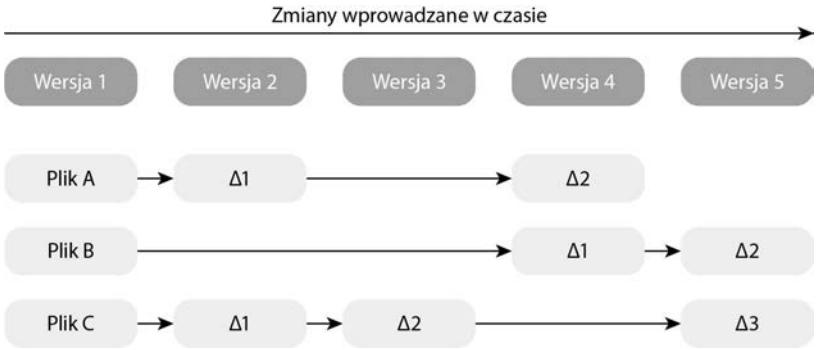
Podstawy Gita

Jak działa Git? To bardzo ważna część rozdziału, ponieważ musisz dowiedzieć się, czym jest Git i jakie mechanizmy działają u jego podstaw, a to bardzo ułatwi Ci skuteczne korzystanie z jego funkcji. Poznając Git, staraj się oczyścić umysł z informacji, jakie masz na temat innych systemów kontroli wersji, takich jak Subversion lub Perforce. Dzięki temu unikniesz małych nieporozumień podczas korzystania z nowego narzędzia. Git przechowuje i obsługuje informacje inaczej niż inne systemy, mimo że nie widać tego w interfejsie użytkownika. Poznanie tych różnic pozwala uniknąć nieporozumień w trakcie korzystania z systemu.

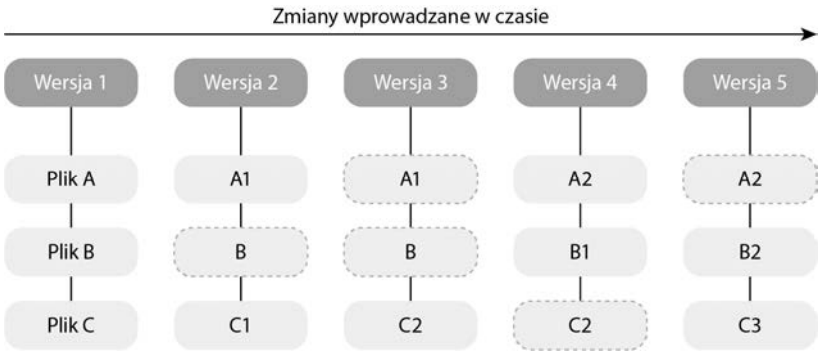
Migawki, a nie różnice

Najważniejszą różnicą między Gitem a pozostałymi systemami kontroli wersji (takimi jak Subversion i podobne) jest sposób, w jaki Git traktuje dane. Inne systemy przechowują informacje w postaci listy zmian wprowadzanych do plików. Systemy, takie jak Subversion, Perforce, Bazaar i tym podobne, wszystkie swoje dane zapisują w postaci pliku bazowego oraz zbioru zmian, jakie z czasem były wprowadzane do tego pliku (rysunek 1.4).

Git nie przechowuje i nie obsługuje danych w ten sam sposób. Traktuje raczej wszystkie dane jak zbiór migawek w miniaturowym systemie plików. Git przy wykonywaniu każdego zatwierdzania zmian albo zapisywania stanu projektu wykonuje migawkę zawartości wszystkich plików projektu i zapisuje sobie referencję do niej. W ramach zwiększania wydajności pracy wszystkie niezmienione pliki nie są ponownie zapisywane w systemie, a w migawce zapisywane jest łącze do zapisanej wcześniej, takiej samej wersji pliku. Jak widać, Git traktuje przechowywane dane jak *strumień migawek* (rysunek 1.5).



Rysunek 1.4. *Przechowywanie danych jako zmian wprowadzanych do bazowej wersji pliku*



Rysunek 1.5. *Przechowywanie danych jako migawek projektu tworzonych w czasie*

To bardzo ważna cecha wyróżniająca Git spośród innych systemów kontroli wersji. Sprawia ona, że Git niemal w każdym aspekcie działa nieco inaczej niż większość systemów kontroli wersji poprzedniej generacji. Powoduje to, że Git jest raczej minisystemem plików obudowanym bardzo zaawansowanymi narzędziami, a nie tylko prostym systemem VCS. W rozdziale 3., podczas omawiania rozgałęzień, przyjrzymy się niektórym zaletom takiego traktowania danych, jakie zastosowano w Gicie.

Niemal każda operacja wykonywana jest lokalnie

Większość operacji w Gicie wymaga dostępu wyłącznie do lokalnych plików i zasobów. Zwykle nie potrzeba żadnych informacji z innych komputerów znajdujących się w sieci. Kto używał wcześniej scentralizowanych systemów kontroli wersji, w których niemal każda operacja obarczona jest opóźnieniami wynikającymi z komunikacji sieciowej, może pomyśleć, że bogowie prędkości obdarowali Git jakimiś nadzwyczajnymi zdolnościami. Ponieważ cała historia projektu znajduje się na Twoim dysku, większość operacji realizowana jest niemal natychmiastowo.

Aby na przykład przejrzeć historię projektu, Git nie musi pobierać jej z centralnego serwera i przygotowywać otrzymanych danych do wyświetlenia na ekranie. Wszystkie potrzebne informacje ma już gotowe w swojej lokalnej bazie danych, a to oznacza, że historię projektu

można zobaczyć właściwie od razu. Jeżeli chcesz zobaczyć zmiany wprowadzone między aktualną wersją pliku a jego wersją sprzed miesiąca, Git może pobrać z bazy wersję pliku sprzed miesiąca i od razu wyznaczyć różnice między tymi plikami. Nie musi prosić zdalnego serwera o wyliczenie tych różnic ani pobierać starej wersji pliku z serwera, aby samodzielnie wykonać porównanie.

Oznacza to również, że w przypadku braku połączenia z siecią albo z firmową siecią VPN nadal mamy możliwość wykonywania niemal wszystkich operacji. Podróżując samolotem albo koleją, nadal możesz pracować nad projektem i zatwierdzać swoje zmiany, aby przesłać je do głównego repozytorium po uzyskaniu połączenia z siecią. Możesz pracować w domu nawet wtedy, kiedy klient VPN nie działa tam poprawnie. W wielu innych systemach kontynuowanie pracy w takich warunkach jest albo niemożliwe, albo bardzo utrudnione. Przykładowo w systemie Perforce bez połączenia z serwerem nie można zrobić prawie nic, natomiast w systemach Subversion i CVS można — co prawda — edytować pliki, ale nie da się już wprowadzić tych zmian do bazy danych (ponieważ ta jest nieosiągalna). Może Ci się to wydawać nieistotnym szczegółem, ale szybko się przekonasz, jaka to wielka różnica w komforcie pracy.

Git sprawdza spójność danych

W Gicie wszystko przed zapisaniem w bazie danych otrzymuje swoją sumę kontrolną, z jej wykorzystaniem można się później odwoływać do poszczególnych obiektów. Oznacza to, że nie da się zmienić zawartości żadnego pliku lub katalogu w taki sposób, żeby Git nie otrzymał o tym informacji. Funkcje te są wbudowane w Git na bardzo niskim poziomie, stając się integralną częścią jego filozofii. Nie da się utracić przesyłanych informacji ani pobrać uszkodzonego pliku, ponieważ Git natychmiast wykrywa takie zdarzenia.

Do wyliczania sum kontrolnych w Gicie używa się algorytmu SHA-1. W efekcie powstaje 40-znakowy ciąg składający się ze znaków szesnastkowych (0 – 9 i a – f), wyliczany na podstawie zawartości pliku lub katalogu. Wyliczona wartość skrótu SHA-1 wygląda podobnie do poniższej:

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

Podczas pracy z Gitem zobaczysz te skróty w wielu miejscach, ponieważ są w Gicie używane często. Co więcej, Git zapisuje dane w swojej bazie, nie przypisując im nazw plików, ale identyfikuje je za pomocą wartości sumy kontrolnej.

Git wyłącznie dopisuje dane

Podczas wykonywania operacji za pomocą Gita zauważysz, że niemal wszystkie dopisują jedynie dane do jego bazy. Bardzo trudno zmusić system do wykonania operacji, której nie da się cofnąć, albo do usunięcia informacji. Jak w każdym innym systemie kontroli wersji, możliwe jest zniszczenie lub utracenie zmian, które nie zostały jeszcze zatwierdzone, ale po przesłaniu migawki do Gita bardzo trudno utracić dane, szczególnie w przypadku, gdy regularnie wypychasz swoje zmiany do innego repozytorium.

To bardzo uprzyjemnia pracę, ponieważ możemy dowolnie eksperymentować bez obawy, że zepsujemy coś w projekcie.

Trzy stany

A teraz skup się. To najważniejsza rzecz, jaką musisz zapamiętać na temat Gita, jeżeli chcesz, żeby późniejsza nauka nie sprawiała Ci trudności. Git definiuje trzy stany, w których mogą znajdować się pliki: zatwierdzony (ang. *comitted*), zmodyfikowany (ang. *modified*) lub śledzony (ang. *staged*). Stan *zatwierdzony* oznacza, że dane są bezpiecznie zapisane w lokalnej bazie danych. Stan *zmodyfikowany* oznacza, że do pliku wprowadzone zostały zmiany, ale nie zostały jeszcze zatwierdzone w bazie. Stan *śledzony* oznacza, że aktualna wersja pliku została oznaczona jako część następnej zatwierdzanej migawki.

Te trzy stany przekładają się na trzy główne sekcje każdego projektu Gita: katalog Gita, katalog roboczy i obszar przechowywania (rysunek 1.6).

Rysunek 1.6. *Katalog roboczy, obszar przechowywania i katalog Gita*

Katalog Gita jest miejscem, w którym system przechowuje swoje metadane oraz bazę danych obiektów Twojego projektu. To najważniejsza część systemu Git, która jest kopiowana podczas klonowania repozytorium z innego komputera.

Katalog roboczy jest kopią jednej z wersji projektu. Pliki z tego katalogu są pobierane ze skompresowanej bazy danych znajdującej się w katalogu Gita i umieszczane we wskazanym miejscu na dysku, gdzie można poddawać je modyfikacjom.

Obszar przechowywania jest plikiem, który najczęściej znajduje się w katalogu Gita. W tym pliku zapisywane są informacje na temat elementów, które mają być częścią następnej operacji zatwierdzania. Czasami stosowany jest też termin „indeks", ale najczęściej mówi się jednak o obszarze przechowywania lub po prostu o przechowalni.

Oto najprostszy sposób pracy z Gitem.

1. Zmieniasz zawartość plików w katalogu roboczym.
2. Oznaczasz pliki jako śledzone, czyli umieszczasz migawki ich zawartości w obszarze przechowywania.
3. Wykonujesz zatwierdzenie, w wyniku czego zawartość plików z obszaru przechowywania zapisywana jest na stałe w katalogu Gita.

Jeżeli jakaś wersja pliku znajduje się w katalogu Gita, to znaczy, że została zatwierdzona. Jeżeli plik został zmodyfikowany, ale też przeniesiony do obszaru przechowywania, to taki plik jest śledzony. Natomiast plik w katalogu roboczym z naniesionymi poprawkami, ale nie przeniesiony do obszaru przechowywania, jest plikiem zmodyfikowanym. W rozdziale 2. dokładniej omówimy te stany, podając metody ich wykorzystywania oraz sposób na pominięcie etapu śledzenia pliku.

Wiersz poleceń

Istnieje wiele różnych metod używania Gita. Do dyspozycji mamy oryginalne narzędzia działające w wierszu poleceń, ale istnieje też wiele rozwiązań z graficznym interfejsem użytkownika. W tej książce będziemy używać Gita w wierszu poleceń. Jedynie w wierszu poleceń można uruchomić wszystkie polecenia Gita, ponieważ większość narzędzi graficznych dla uproszczenia implementuje jedynie pewien podzbiór funkcji Gita. Jeżeli wiesz, jak posługiwać się narzędziem w wierszu poleceń, z pewnością łatwo domyślisz się, jak należy użyć go w programie z interfejsem graficznym. Niestety ta zależność nie działa w drugą stronę. Poza tym wybór klienta graficznego jest kwestią osobistych upodobań, ale każdy użytkownik będzie miał zainstalowane dokładnie te same narzędzia wiersza poleceń.

Oczekujemy zatem, że wiesz, jak należy otworzyć *Terminal* w komputerach Mac albo okno *Wiersza Poleceń* lub *PowerShell* w systemach Windows. Jeżeli nie wiesz, o czym tutaj mówimy, lepiej zatrzymaj się w tym miejscu i szybko poznaj obsługę swojego wiersza poleceń, aby odpowiednio interpretować podawane w tej książce przykłady oraz ich opisy.

Instalowanie Gita

Zanim zaczniesz używać Gita, musisz umieścić odpowiednie oprogramowanie na swoim komputerze. Nawet jeżeli masz zainstalowane narzędzia Gita, dobrym pomysłem jest zaktualizowanie ich do najnowszej wersji. Możesz to zrobić, instalując w systemie pakiet Gita albo stosując specjalny instalator. Można też pobrać kod źródłowy systemu i samodzielnie go skompilować.

■ **Uwaga** Książkę tę pisaliśmy, używając Gita w wersji 2.0.0. Oznacza to, że większość prezentowanych tu narzędzi powinna działać nawet w bardzo starych wersjach Gita. Część z nich może jednak nie pracować zgodnie z opisem. Ponieważ Git świetnie radzi sobie z zachowywaniem zgodności wstecznej, podawane tu instrukcje będą działały również w wersjach nowszych niż 2.0.

Instalowanie w systemie Linux

Jeżeli chcesz zainstalować Git w systemie Linux za pomocą instalatora binarnego, najłatwiej będzie Ci skorzystać z narzędzi do zarządzania pakietami dostępnych w używanej przez Ciebie dystrybucji. Jeśli na przykład pracujesz z systemem Fedora, możesz zastosować narzędzie `yum`:

```
$ yum install git
```

Jeżeli jednak korzystasz z dystrybucji bazującej na Debianie, takiej jak Ubuntu, użyj narzędzia `apt-get`:

```
$ apt-get install git
```

Więcej instrukcji opisujących sposoby instalowania Gita w kilku różnych odmianach systemów uniksowych znajdziesz na stronach Gita, pod adresem *https://git-scm.com/download/linux*.

Instalowanie w systemie Mac

W systemie Mac można zainstalować Git na kilka różnych sposobów. Najprostszym z nich jest chyba zainstalowanie narzędzi wiersza poleceń Xcode. W systemie Maverics (10.9) lub nowszych możesz po prostu spróbować uruchomienia polecenia `git` w oknie Terminal. Jeżeli Git nie jest jeszcze zainstalowany, system sam zapyta, czy chcesz go zainstalować.

Jeżeli chcesz korzystać z aktualniejszej wersji Gita, możesz zainstalować go za pomocą instalatora binarnego (rysunek 1.7). Instalator dla systemu OSX jest dostępny do pobrania na stronach Gita, pod adresem *https://git-scm.com/download/mac*.

Rysunek 1.7. *Instalator Gita dla systemu OS X*

Możesz też zainstalować go jako część instalacji GutHuba dla systemów Mac. W graficznym narzędziu Gita znajduje się też opcja zainstalowania wszystkich narzędzi wiersza poleceń. Odpowiedni instalator możesz pobrać ze stron projektu GitHub for Mac, pod adresem *https://mac.github.com*.

Instalowanie w systemach Windows

Istnieje tylko kilka sposobów instalowania Gita w systemach Windows. Najbardziej oficjalna kompilacja jest zawsze dostępna do pobrania na stronach Gita. Wystarczy wpisać w przeglądarce adres *https://git-scm.com/download/win*, a pobieranie powinno się rozpocząć automatycznie. Pamiętaj, że jest to projekt o nazwie Git for Windows (nazywany też msysGit), który jest niezależny od samego projektu Gita. Więcej informacji na jego temat znajdziesz pod adresem *https://msysgit.github.io/*.

Inną prostą metodą zainstalowania Gita w swoim systemie jest zainstalowanie w nim aplikacji GitHub for Windows. W jej instalatorze znajdują się też narzędzia wiersza poleceń Gita oraz specjalne GUI. Instalator dobrze współpracuje z powłoką PowerShell i właściwie konfiguruje mechanizm buforowania danych uwierzytelniających, jak również poprawnie definiuje znaki końca wiersza. Więcej informacji na ten temat podamy dalej w książce, a na razie wystarczy powiedzieć, że instalator przyjmuje tu właściwe ustawienia dla tych opcji. Instalator GitHub for Windows możesz pobrać z adresu *https://windows.github.com*.

Niektórzy mogą zdecydować się na zainstalowanie Gita, kompilując jego kod źródłowy, ponieważ w ten sposób uzyskują zawsze najnowszą wersję. Instalatory binarne zwykle powstają z małym opóźnieniem, choć nie ma to już większego znaczenia, ponieważ projekt Gita zdążył dojrzeć przez te wszystkie lata.

Jeżeli chcesz zainstalować Git z jego kodów źródłowych, musisz najpierw pobrać biblioteki, z których korzysta Git, czyli *curl*, *zlib*, *openssl*, *expat* i *libiconv*. Przykładowo w systemach używających programów `yum` (takich jak Fedora) lub `apt-get` (takich jak systemy bazujące na Debianie) możesz skorzystać z poniższych poleceń, aby zainstalować wszystkie niezbędne zależności:

```
$ yum install curl-devel expat-devel gettext-devel \
   openssl-devel zlib-devel

$ apt-get install libcurl4-gnutls-dev libexpat1-dev gettext \
   libz-dev libssl-dev
```

Po zainstalowaniu wszystkich zależności możesz przystąpić do pobierania paczki z kodem źródłowym oficjalnego wydania, która jest dostępna w kilku różnych miejscach. Możesz na przykład pobrać ją z witryny *kernel.org*, spod adresu *https://mirrors.edge.kernel.org/pub/software/scm/git/*, ale możesz też skorzystać z serwera lustrzanego GitHuba, pod adresem *https://github.com/git/git/releases*. Zwykle to na stronach GitHuba można znacznie łatwiej zorientować się, który pakiet jest najnowszy, ale na stronach *kernel.org* dostępne są sygnatury pakietów, co pozwala na skontrolowanie poprawności pobranych danych.

Sama kompilacja i instalacja wygląda tak:

```
$ tar -zxf git-1.9.1.tar.gz
$ cd git-1.9.1
$ make configure
$ ./configure --prefix=/usr
$ make all doc info
$ sudo make install install-doc install-html install-info
```

Po wykonaniu takiej instalacji możesz też użyć Gita, aby z jego pomocą pobierać aktualizacje:

```
$ git clone git://git.kernel.org/pub/scm/git/git.git
```

Pierwsze konfigurowanie Gita

Po zainstalowaniu Gita w swoim systemie warto wykonać kilka operacji, aby skonfigurować swoje środowisko. Na każdym komputerze trzeba te operacje wykonać tylko raz, ponieważ ustalona w ten sposób konfiguracja nie zmienia się po zainstalowaniu aktualizacji. Oczywiście opcje te można zmienić w dowolnym momencie, ponownie uruchamiając te same polecenia.

Git udostępnia specjalne narzędzie o nazwie `git config`, które umożliwia odczytywanie i ustalanie zmiennych konfiguracyjnych wpływających na działanie i wygląd Gita. Zmienne konfiguracyjne mogą być zapisywane w trzech różnych miejscach.

- Plik */etc/gitconfig*. Przechowuje wartości konfiguracji obowiązujące wszystkich użytkowników systemu oraz wszystkie repozytoria. Jeżeli do polecenia `git config` dodasz opcję `--system`, konfiguracja będzie odczytywana i zapisywana do tego pliku.
- Plik *~/.gitconfig* lub *~/.config/git/config*. Przechowuje konfigurację właściwą dla wybranego użytkownika. Podając Gitowi opcję `--global`, nakazujemy mu korzystać z tego właśnie pliku.
- Plik *config* znajdujący się w katalogu Gita (czyli *.git/config*) związanym z dowolnym używanym właśnie repozytorium. Przechowuje konfigurację właściwą dla tego repozytorium.

Każdy następny poziom nadpisuje wartości z poprzednich poziomów, a zatem zapisy z pliku *.git/config* są ważniejsze niż te z pliku */etc/config*.

W systemach Windows Git poszukuje pliku *.gitconfig* w katalogu głównym użytkownika (dla większości użytkowników będzie to katalog *C:\Users\NAZWA_UŻYTKOWNIKA*). Poszukuje też pliku */etc/gitconfig*, choć ta ścieżka wyznaczana jest względem katalogu, w którym zainstalowano Git w systemie Windows.

Tożsamość

Pierwszą rzeczą, jaką trzeba zrobić po zainstalowaniu Gita, jest podanie mu swojej nazwy użytkownika oraz adresu e-mail. To bardzo ważne, ponieważ Git używa tych informacji w każdej operacji zatwierdzania, a zatem każda zmiana umieszczona w repozytorium będzie miała je dołączone w sposób uniemożliwiający późniejszą modyfikację.

```
$ git config --global user.name "Adam Nowak"
$ git config --global user.email adamnowak@przyklad.pl
```

W tym przypadku również możesz zdefiniować te ustawienia tylko raz, dodając do tych instrukcji opcję `--global`, ponieważ w ten sposób Git będzie zawsze korzystał z tych informacji przy wykonywaniu jakichkolwiek operacji w systemie. Jeżeli chcesz zmienić te wartości

i w wybranym projekcie używać innej nazwy użytkownika lub adresu e-mail, to w tym projekcie możesz uruchomić te same polecenia bez opcji `--global`.

Większość narzędzi z interfejsem graficznym prosi o podanie tych informacji zaraz po pierwszym uruchomieniu.

Edytor

Po zdefiniowaniu swojej tożsamości możesz przystąpić do konfigurowania domyślnego edytora, który będzie używany za każdym razem, gdy Git będzie prosił Cię o wpisanie wiadomości. Jeżeli ta opcja nie zostanie skonfigurowana, Git będzie korzystał z domyślnego edytora systemowego, którym najczęściej jest Vim. Jeżeli chcesz używać innego edytora tekstu, takiego jak Emacs, skorzystaj z poniższego polecenia:

```
$ git config --global core.editor emacs
```

■ **Uwaga** Vim i Emacs są popularnymi edytorami tekstu często używanymi przez programistów w systemach uniksowych, takich jak Linux lub Mac. Jeżeli nie używasz tych edytorów albo pracujesz w systemach Windows, musisz poszukać specjalizowanych instrukcji konfigurowania edytora dla Gita.

Sprawdzanie ustawień

Jeżeli chcesz skontrolować swoje ustawienia, możesz użyć polecenia `git config --list`, aby wypisać wszystkie zapisy konfiguracyjne, które są w tym momencie dostępne dla Gita:

```
$ git config --list
user.name=Adam Nowak
user.email=adamnowak@przyklad.pl
color.status=auto
color.branch=auto
color.interactive=auto
color.diff=auto
...
```

Niektóre klucze mogą pojawiać się wielokrotnie, ponieważ Git odczytuje wartości tych kluczy z różnych plików (na przykład z plików (*/etc/gitconfig* i *~/.gitconfig*). W takich przypadkach Git będzie używał wartości z ostatniego odczytanego pliku.

Możesz też sprawdzić, jakiej wartości wybranego klucza używa aktualnie, wprowadzając polecenie `git config <klucz>`:

```
$ git config user.name
Adam Nowak
```

Szukanie pomocy

Jeżeli podczas używania Gita będzie Ci potrzebna pomoc, możesz na trzy różne sposoby wywołać stronę podręcznika man opisującą wybrane polecenie Gita:

```
$ git help <polecenie>
$ git <polecenie> --help
$ man git-<polecenie>
```

Aby na przykład wyświetlić stronę pomocy dla polecenia `config`, możesz wpisać:

```
$ git help config
```

To bardzo przydatne polecenia, ponieważ możesz z nich korzystać w dowolnym momencie, nawet bez połączenia z internetem. Jeżeli strony podręcznika man i ta książka okażą się niewystarczające i będziesz potrzebować rozmowy z ekspertem lub innym użytkownikiem, możesz skorzystać z kanałów IRC *#git* lub *#github* dostępnych na serwerze Freenode (*irc.freenode.net*). Na tych kanałach zwykle dyskutują setki osób, które zapewne doskonale wiedzą, jak używać Gita i chętnie pomogą Ci rozwiązać Twój problem.

Podsumowanie

Teraz znasz już podstawowe zasady działania Gita i wiesz, czym różni się od scentralizowanych systemów kontroli wersji, z których korzystałeś wcześniej. W Twoim systemie powinna być już zainstalowana działająca wersja Gita, w której zdefiniowałeś Twoją tożsamość. Nadszedł czas, żeby nauczyć się wykonywania podstawowych operacji w Gicie.

GitHub:
odkryj świat rozproszonej kontroli wersji!

W wypadku pracy zespołowej efektywny system kontroli wersji jest warunkiem sukcesu. Powszechnie stosowanym w tym zakresie rozwiązaniem jest Git, który umożliwia zarządzanie wersjami projektu, przeglądanie wszystkich zmian, a także przywracanie pliku do dowolnej wcześniejszej wersji. Git jest dynamicznie rozwijanym oprogramowaniem. W ostatnich latach wzbogacił się o wiele przydatnych narzędzi, wyjątkowe stały się też możliwości jego konfiguracji i dostosowania. Wciąż jednak pełne wykorzystanie potencjału Gita wymaga wprawy i dobrej znajomości specyfiki pracy tego narzędzia.

To drugie, zaktualizowane i uzupełnione wydanie popularnego podręcznika opracowanego przez specjalistów dla profesjonalistów. Dowiesz się z niego, czym jest rozproszona kontrola wersji i jakie daje korzyści. Poznasz poszczególne funkcjonalności Gita i nauczysz się je rozbudowywać tak, aby zaspokajały nawet najbardziej wyrafinowane potrzeby. W książce znalazły się również wyczerpujące opisy funkcji dostępnych w Git 2.0, a także omówienie systemu usług GitHub i jego zalet. Istotną zmianą w nowym wydaniu jest uwzględnienie w znaczącej większości prezentowanych przykładów protokołu HTTP w miejsce SSH — co stanowi odpowiedź na rosnącą popularność HTTP.

W książce między innymi:

- rodzaje systemów kontroli wersji i podstawy pracy z Gitem
- konfiguracja Gita i rozproszone metody pracy
- praca w zespole z systemem GitHub
- konflikty scalania i ich rozstrzyganie
- dostosowywanie Gita do szczególnych potrzeb
- Git i inne systemy, w tym Subversion, Mercurial i Perforce

Scott Chacon jest programistą, współzałożycielem serwisu GitHub i jednym z prowadzących stronę główną Gita (git-scm.com). Występował na wielu prestiżowych konferencjach na całym świecie i przeprowadził setki korporacyjnych szkoleń dla użytkowników Gita.

Ben Straub jest programistą, współtwórcą biblioteki libgit2 i międzynarodowym popularyzatorem Gita, a także nienasyconym czytelnikiem, odkrywcą i pasjonatem sztuki tworzenia doskonałego oprogramowania.

Helion

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-283-8731-7

Cena: 109,00 zł

Apress