

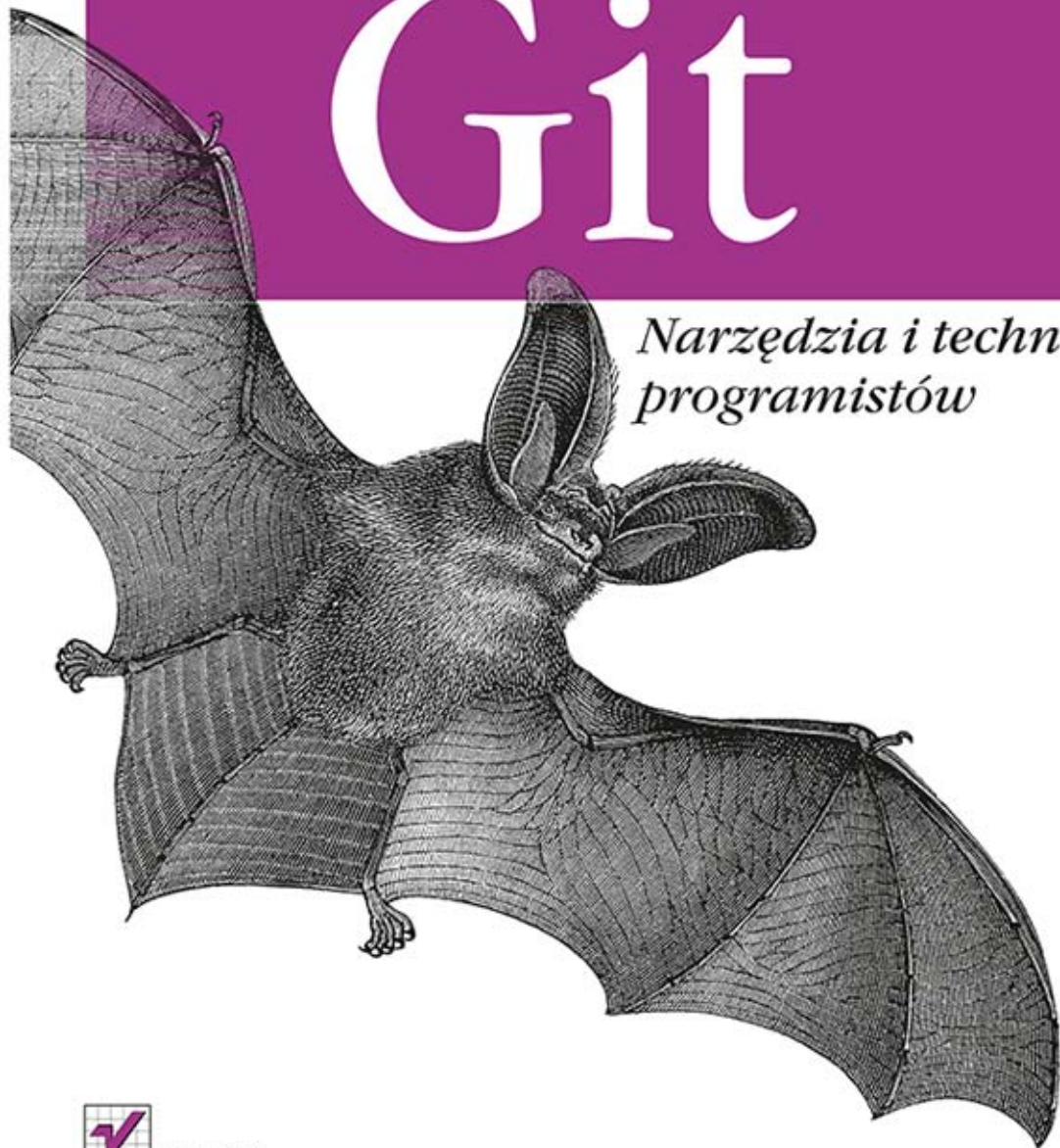
*Poznaj nowoczesny  
system kontroli wersji!*

**Wydanie II**

*Kontrola wersji z systemem*

# Git

*Narzędzia i techniki  
programistów*



**O'REILLY®**

*Jon Loeliger, Matthew McCullough*

Tytuł oryginału: Version Control with Git, Second Edition

Tłumaczenie: Zdzisław Płoski

ISBN: 978-83-246-8176-1

© 2014 Helion S.A.

Authorized Polish translation of the English edition of Version Control with Git, 2nd Edition, ISBN 9781449316389 © 2012 Joe Loeliger.

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION  
ul. Kościuszki 1c, 44-100 GLIWICE  
tel. 32 231 22 19, 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/koweg2>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to!» Nasza społeczność](#)

---

# Spis treści

<b>Przedmowa .....</b>	<b>11</b>
<b>1. Wprowadzenie .....</b>	<b>17</b>
Podstawy	17
Narodziny Gita	18
Poprzednicy	20
Na osi czasu	21
Cóż to za nazwa?	22
<b>2. Instalowanie Gita .....</b>	<b>23</b>
Zastosowanie binarnych dystrybucji Linuksa	23
Dystrybucje Debian lub Ubuntu	23
Inne dystrybucje binarne	24
Uzyskanie wydania źródłowego	25
Budowanie i instalowanie	26
Instalowanie Gita w systemie Windows	27
Instalowanie pakietu Git w systemie Cygwin	28
Instalowanie wolno stojącej wersji Gita (msysGit)	29
<b>3. Na dobry początek .....</b>	<b>31</b>
Polecenie git	31
Szybkie wprowadzenie do użytkowania Gita	33
Tworzenie archiwum początkowego	33
Dodawanie pliku do Twojego archiwum	34
Konfigurowanie autora zatwierdzenia	35
Wykonanie kolejnego zatwierdzenia	36
Przeglądanie Twoich zatwierdzeń	36
Przeglądanie różnic w zatwierdzeniach	37
Usuwanie i przemianowywanie plików w Twoim archiwum	38
Tworzenie kopii Twojego archiwum	39

Pliki konfiguracyjne	39
Konfigurowanie synonimu	41
Zasięganie języka	42
<b>4. Podstawowe koncepcje Gita .....</b>	<b>43</b>
Pojęcia podstawowe	43
Archiwa	43
Typy obiektów Gita	44
Indeks	45
Nazwy adresowane treścią	46
Git nadzoruje treść	46
Nazwy ścieżek a treść	47
Pliki pakowane	48
Obrazy magazynu obiektów	49
Koncepcje Gita uwidocznione w działaniu	51
Zawartość katalogu .git	51
Obiekty, haszowania i bloby	52
Pliki i drzewa	53
Uwaga o zastosowaniu w Gicie algorytmu SHA1	54
Hierarchie drzewiaste	55
Zatwierdzenia	56
Metki	57
<b>5. Zarządzanie plikami a indeks .....</b>	<b>59</b>
Wszystko kręci się wokół indeksu	59
Klasyfikacje plików w Gicie	60
Użycie polecenia git add	62
Kilka uwag o stosowaniu polecenia git commit	64
Użycie polecenia git commit --all	64
Zapisywanie komunikatów dziennika zatwierdzeń	65
Użycie polecenia git rm	65
Użycie polecenia git mv	67
Uwaga o śledzeniu przemianowań	68
Plik .gitignore	69
Szczegółowy przegląd modelu obiektowego i plików Gita	71
<b>6. Zatwierdzenia .....</b>	<b>77</b>
Niepodzielne zbiory zmian	78
Identyfikowanie zatwierdzeń	79
Bezwzględne nazwy zatwierdzeń	79
Refy i symrefy	80
Względne nazwy zatwierdzeń	81

Historia zatwierdzeń	83
Przeglądanie starych zatwierdzeń	83
Grafy zatwierdzeń	85
Przedziały zatwierdzeń	89
Znajdowanie zatwierdzeń	93
Użycie polecenia git bisect	93
Użycie polecenia git blame	97
Użycie kilofa	98
<b>7. Odgałęzienia .....</b>	<b>99</b>
Powody stosowania odgałęzień	99
Nazwy odgałęzień	100
Co używać, a czego nie używać w nazwach odgałęzień	101
Zastosowanie odgałęzień	101
Tworzenie odgałęzień	103
Sporządzanie wykazów nazw odgałęzień	104
Przeglądanie odgałęzień	104
Wyciąganie odgałęzień	106
Elementarny przykład wyciągania odgałęzienia	107
Wyciąganie w wypadku niezatwierdzonych zmian	107
Łączenie zmian w nowe odgałęzienie	109
Tworzenie i wyciąganie nowego odgałęzienia	111
Wysobnione odgałęzienia HEAD	111
Usuwanie odgałęzień	112
<b>8. Różnice .....</b>	<b>115</b>
Postaci polecenia git diff	116
Prosty przykład polecenia git diff	120
Polecenie git diff i przedziały zatwierdzeń	122
Polecenie git diff z ograniczeniem ścieżki	125
Porównanie wyprowadzania różnic w systemach Subversion i Git	126
<b>9. Łączenia .....</b>	<b>129</b>
Przykłady łączeń	129
Przygotowanie do łączenia	130
Łączenie dwóch odgałęzień	130
Konflikt w trakcie łączenia	132
Postępowanie z konfliktami łączenia	135
Lokalizowanie konfliktowych plików	136
Badanie konfliktów	136
W jaki sposób Git śledzi konflikty	140

Zakończenie rozwiązywania konfliktu	142
Zaniechanie lub wznowienie łączenia	143
Strategie łączenia	144
Łączenia zdegenerowane	146
Łączenia zwykłe	147
Łączenia specjalne	149
Stosowanie strategii łączenia	149
Sterowniki łączenia	151
Jak Git rozpatruje łączenia	151
Łączenia i model obiektowy Gita	151
Łączenia zgniatane	152
Czemu nie łączyć po prostu każdej zmiany po kolei?	153
<b>10. Zmianianie zatwierdzeń .....</b>	<b>155</b>
Uwaga dotycząca zmieniania historii	157
Użycie polecenia git reset	158
Użycie polecenia git cherry-pick	164
Użycie polecenia git revert	166
Polecenia reset, revert i checkout	167
Zmiana zatwierdzenia szczytowego	168
Przebazowanie zatwierdzeń	170
Użycie polecenia git rebase -i	172
Operacja rebase a łączenie	176
<b>11. Skrytka stash i rejestr odniesień reflog .....</b>	<b>181</b>
Skrytka	181
Rejestr odniesień	189
<b>12. Archiwa zdalne .....</b>	<b>193</b>
Koncepcje archiwum	194
Archiwa czyste i rozwojowe	194
Klony archiwów	195
Piloty	196
Odgałęzienia nadzorujące	197
Odwoływanie się do innych archiwów	198
Odwołania do archiwów zdalnych	198
Refspec — specyfikator odniesienia	200
Przykład użycia zdalnych archiwów	202
Tworzenie archiwum wzorcowego	203
Uczyń swój własny początek zdalnym	204
Prowadzenie prac we własnym archiwum	206
Wypychanie zmian	206

Dodawanie nowego wykonawcy	207
Pobieranie uaktualnień archiwum	209
Cykl rozwoju zdalnego archiwum w ujęciu rysunkowym	214
Klonowanie archiwum	214
Historie alternatywne	215
Niespieszne wypychanie	216
Pobieranie alternatywnej historii	217
Łączenie historii	218
Konflikty łączenia	218
Wypychanie połączonej historii	219
Konfigurowanie zdalne	219
Użycie polecenia git remote	220
Użycie polecenia git config	221
Obróbka ręczna	222
Działanie na odgałęzieniach nadzorowania	222
Tworzenie gałęzi nadzorowania	222
Przed i za	225
Dodawanie i usuwanie odgałęzień zdalnych	226
Archiwa czyste i polecenie git push	227
<b>13. Zarządzanie archiwum .....</b>	<b>229</b>
Słowo o serwerach	229
Publikowanie archiwów	230
Archiwa z kontrolowanym dostępem	230
Archiwa z anonimowym dostępem do czytania	231
Archiwa z anonimowym dostępem do pisania	235
Publikowanie archiwum na koncie GitHub	235
Wskazówka dotycząca publikowania archiwum	236
Struktura archiwum	237
Struktura archiwum dzielonego	237
Struktura archiwum rozproszonego	238
Przykłady struktur archiwów	239
Jak żyć w warunkach rozproszenia	241
Zmienianie historii upublicznionej	241
Rozdzielność kroków zatwierdzeń i publikowania	242
Ani jednej historii prawdziwej	242
Znaj swoje miejsce	243
Przepływy w górę i w dół	244
Role pielęgnatora i budowniczego	244
Współpraca między pielęgnatorem a budowniczym	245
Dualność ról	246

Praca z wieloma archiwami	247
Twoja własna przestrzeń robocza	247
Gdzie rozpocząć swoje archiwum	248
Przekształcenie w inne archiwum w górze	249
Używanie wielu górnych archiwów	250
Rozwidlanie projektów	252
<b>14. Łaty .....</b>	<b>255</b>
Dlaczego używamy łat?	256
Generowanie łat	257
Łaty i sortowanie topologiczne	264
Pocztowe ekspediowanie łat	264
Stosowanie łat	267
Złe łaty	273
Łatanie a łączenie	273
<b>15. Doczepki .....</b>	<b>275</b>
Instalowanie doczepek	277
Doczepki przykładowe	277
Utworzenie pierwszej doczepki	278
Dostępne doczepki	280
Doczepki powiązane z zatwierdzeniami	280
Doczepki powiązane z łatami	281
Doczepki powiązane z wypychaniem	282
Inne doczepki do lokalnego archiwum	283
<b>16. Zestawianie projektów .....</b>	<b>285</b>
Stare rozwiązanie: wyciągi częściowe	286
Rozwiązanie oczywiste: zaimportuj kod do swojego projektu	287
Importowanie podprojektów przez kopiowanie	289
Importowanie podprojektów poleceniem git pull –s subtree	289
Kierowanie swoich zmian w górę	293
Rozwiązanie zautomatyzowane: wyciąganie podprojektów z użyciem odpowiednich skryptów	293
Rozwiązanie rodzime: gitlinki i git submodule	295
Odsyłacze gitlinks	295
Polecenie git submodule	297
<b>17. Najlepsze praktyki dotyczące podmodułów .....</b>	<b>301</b>
Polecenia podmodułów	301
Dlaczego podmoduły?	302
Przygotowywanie podmodułów	303
Dlaczego tylko do czytania?	304



Dlaczego nie tylko do czytania?	304
Sprawdzanie haszowań zatwierdzeń podmodułów	305
Ponowne wykorzystanie pełnomocnictw	305
Przypadki użycia	306
Wielopoziomowe zagnieżdżanie archiwów	307
Podmoduły na horyzoncie	307
<b>18. Zastosowanie Gita do archiwów systemu Subversion .....</b>	<b>309</b>
Przykład: płytki klon jednego odgałęzienia	309
Pora na wykonywanie zmian w Gicie	312
Pobranie przed zatwierdzeniem	313
Zatwierdzanie za pomocą git svn rebase	314
Wypychanie, ciągnięcie, rozgałęzianie i łączenie za pomocą git svn	315
Utrzymywanie prostoty identyfikatorów zatwierdzeń	316
Klonowanie wszystkich gałęzi	317
Dzielenie Twojego archiwum	319
Ponowne włączanie do Subversion	320
Inne uwagi o pracy z systemem Subversion	321
Cecha svn:ignore a plik .gitignore	321
Rekonstruowanie pamięci podręcznej git-svn	322
<b>19. Działania zaawansowane .....</b>	<b>323</b>
Użycie polecenia git filter-branch	323
Przykłady użycia polecenia git filter-branch	325
Pułapki filter-branch	330
Jak pokochałem polecenie git rev-list	330
Wyciąganie według daty	331
Odzyskiwanie starej wersji pliku	333
Interaktywne wystawianie kawałków	335
Rekonstruowanie utraconego zatwierdzenia	345
Polecenie git fsck	345
Ponowne przyłączenie utraconego zatwierdzenia	349
<b>20. Rady, chwyt i sposoby .....</b>	<b>351</b>
Interaktywne przebazowanie z zabrudzonym katalogiem roboczym	351
Usuwanie zbędnych plików edytora	352
Łączenie nieużytków	352
Podział archiwum	354
Sposoby rekonstruowania zatwierdzeń	355
Rady dotyczące konwersji Subversion	356
Ogólne zalecenia	356
Usuwanie trzonu po zaimportowaniu SVN	356
Usuwanie identyfikatorów zatwierdzeń SVN	357

Manipulowanie odgałęzzeniami pochodzącymi z dwu archiwów	357
Odzyskiwanie z przebazowania w górze	358
Tworzenie własnych poleceń w Gicie	359
Szybki przegląd zmian	360
Czyszczenie	361
Użycie polecenia git-grep do przeszukiwania archiwum	361
Aktualizowanie i usuwanie refów	363
Postępowanie za przemieszczonymi plikami	364
Zachowaj ten plik, lecz go nie nadzoruj	365
Byłeś tu już wcześniej?	366
<b>21. Git i GitHub .....</b>	<b>367</b>
Archiwum kodu powszechnie dostępnego	367
Tworzenie archiwum w GitHubie	369
Kodowanie społeczne na otwartych źródłach	372
Obserwatorzy	373
Kanał informacyjny	373
Rozwidlenia	374
Przygotowywanie zamówień ciągnięcia	376
Obsługiwanie zamówień ciągnięcia	377
Powiadomienia	379
Odnajdywanie użytkowników, projektów i kodu	382
Wikisy	383
Strony GitHuba (Git do witryn)	384
Edytor kodu wprost ze strony	386
Most do systemu Subversion	388
Metki automatycznie zamieniane na pliki archiwalne	389
Organizacje	390
Interfejs REST API	390
Kodowanie społeczne oparte na źródłach zamkniętych	391
Docelowe czerpanie z otwartych źródeł	392
Modele kodowania	393
GitHub jako inicjatywa gospodarcza	395
GitHub — podsumowanie	396
<b>Skorowidz .....</b>	<b>397</b>

---

# Podstawowe koncepcje Gita

## Pojęcia podstawowe

W poprzednich rozdziałach przedstawiliśmy typowe zastosowanie Gita, co prawdopodobnie rozwiązało tylko wórek z pytaniami. Czy Git przechowuje cały plik przy każdym zatwierdzeniu? Do czego służy katalog `.git`? Dlaczego identyfikator zatwierdzenia przypomina jakiś bełkot? Czy mam go sobie zapisywać?

Jeśli używałeś jakiegoś innego systemu kontroli wersji, jak SVN lub CVS, to być może polecenia w poprzednim rozdziale wyglądały znajomo. Rzeczywiście, Git służy do tego samego i zawiera wszystkie operacje, których oczekujesz od nowoczesnego systemu kontroli wersji. Git jednak różni się od innych tego typu systemów w pewien zasadniczy i zaskakujący sposób.

W tym rozdziale zastanowimy się, dlaczego i w jaki sposób Git się różni, analizując kluczowe komponenty jego architektury i pewne ważne koncepcje. Skupiamy się tutaj na sprawach podstawowych i pokazujemy, jak współpracować z jednym archiwum. W rozdziale 12 wyjaśniamy, jak pracować z wieloma połączonymi archiwami. Nadzorowanie wielu archiwów może wydawać się zniechęcającą perspektywą, lecz kwestie elementarne, które poznasz w tym rozdziale, przenoszą się na tamte obszary bez zmian.

## Archiwa

*Archiwum* Gita (*skarbiec*, ang. *repository*<sup>1</sup>) jest po prostu bazą danych, zawierającą wszystkie informacje potrzebne do przechowywania wersji projektu i jego historii oraz zarządzania nimi. W Gicie, jak w większości systemów kontroli wersji, archiwum przechowuje kompletną kopię całego przedsięwzięcia w trakcie jego istnienia. Jednakże — w odróżnieniu od innych systemów VCS — archiwum Gita dostarcza nie tylko kompletnej, działającej kopii wszystkich przechowywanych w nim plików, lecz także służy kopią samego siebie, na której można działać.

---

<sup>1</sup> Nie chcemy używać określenia *repozytorium* z dwóch powodów: (1) kojarzy się z zetłalymi aktami i (2) jest długie — *przyp. tłum.*

Git utrzymuje w każdym archiwum zbiór wartości konfiguracyjnych. W poprzednim rozdziale widziałeś niektóre z nich, takie jak nazwisko (nazwa) użytkownika archiwum i adres jego poczty elektronicznej. W odróżnieniu od danych plikowych i innych metadanych, ustawienia konfiguracji nie są przenoszone między archiwami podczas operacji *klonowania*, czyli powielania. Zamiast tego Git zarządza konfiguracją oraz ustawialnymi informacjami i nadzoruje je na szczeblu stanowiska (maszyny), użytkownika i archiwum.

Wewnątrz archiwum Git utrzymuje dwie podstawowe struktury danych: *magazyn obiektów* (ang. *object store*) i *indeks*. Wszystkie te dane z archiwum są przechowywane u korzenia Twojego katalogu roboczego<sup>2</sup>, w ukrytym podkatalogu o nazwie *.git*.

Magazyn obiektów jest zaprojektowany tak, aby można go było sprawnie kopiować w czasie wykonywania operacji klonowania jako część mechanizmu wspomagającego w pełni rozproszony system VCS. Indeks stanowi informację przejściową, dotyczy indywidualnego archiwum i może być tworzony lub modyfikowany na życzenie — stosownie do potrzeb.

W następnych dwóch podrozdziałach opisano magazyn obiektów i indeks bardziej szczegółowo.

## Typy obiektów Gita

Magazyn obiektów stanowi centralny element implementacji archiwum Gita. Zawiera Twoje oryginalne pliki danych i wszystkie komunikaty dziennika, informacje autorskie, daty i inne informacje niezbędne do odbudowania dowolnej wersji lub odgałęzienia projektu.

Git lokuje w magazynie obiektów tylko cztery ich typy: *bloby* (ang. *blobs*), *drzewa* (ang. *trees*), *zatwierdzenia* (ang. *commits*) i *metki* (ang. *tags*). Te cztery elementarne (niepodzielne) obiekty tworzą podstawę gitowych struktur danych wyższego poziomu.

### *Bloby*

Każda wersja pliku jest reprezentowana w postaci blobu. Blob<sup>3</sup>, jako skrót od *binary large object* (z ang. *duży obiekt binarny*), jest terminem powszechnie używanym w technice obliczeniowej na określenie pewnej zmiennej lub pliku, które mogą zawierać dowolne dane i których wewnętrzna struktura jest ignorowana przez program. Blob jest traktowany jako coś nieprzeniknionego. Blob przechowuje dane pliku, lecz nie zawiera żadnych metadanych dotyczących pliku — nawet jego nazwy.

### *Drzewa*

Obiekt drzewa reprezentuje jeden poziom informacji katalogowej. Zapisane są w nim identyfikatory blobów, nazwy ścieżek i trochę metadanych dotyczących wszystkich plików w jednym katalogu. Może on również zawierać rekurencyjne odniesienia do innych obiektów (pod)drzew, wskutek czego tworzy pełną hierarchię plików i podkatalogów.

---

<sup>2</sup> W oryginale: *at the root of your working directory*, określenie jest trochę niejasne w swej skrótowości; chodzi o korzeń poddrzewa katalogów, w którym użytkownik założył archiwum — *przyj. tłum.*

<sup>3</sup> Nazwa jest o tyle zabawna, że po angielsku *blob* oznacza m.in. *kleks* lub *plamę* — *przyj. tłum.*

## Zatwierdzenia

Obiekt zatwierdzenia zawiera metadane dotyczące każdej zmiany wprowadzonej w archiwum, w tym autora, zatwierdzającego (ang. *committer*), datę i komunikat dziennika. Każde zatwierdzenie wskazuje na obiekt drzewa, który w jednym, migawkowym i całościowym ujęciu obejmuje stan archiwum z chwili wykonywania zatwierdzenia. Początkowe zatwierdzenie, czyli zatwierdzenie korzeniowe, nie ma przodka. Większość zatwierdzonych zapisów ma jednego przodka zatwierdzenia, przy czym w dalszej części książki (w rozdziale 9) wyjaśniamy, w jaki sposób zatwierdzenie może mieć odniesienia do więcej niż jednego przodka.

## Metki

Obiekt metki przypisuje danemu obiektowi (zwykle obiektowi zatwierdzenia) dowolną nazwę, taką jednak, która przypuszczalnie jest czytelna dla człowieka. Choć `9da581d910c9c4ac93557ca4859e767f5caf5169` odnosi się do dokładnie i dobrze określonego zatwierdzenia, sensowniejszą dla człowieka i więcej mówiącą będzie metka w stylu `Ver-1.0-Alpha`.

Z biegiem czasu w magazynie obiektów przybywa coraz to nowych informacji odzwierciedlających i odwzorowujących redakcje, uzupełnienia i usunięcia dokonywane w Twoim zadaniu. Aby oszczędzać przestrzeń dyskową i przepływność sieci, Git kompresuje i przechowuje obiekty w *plikach pakowanych* (ang. *pack files*<sup>4</sup>), które również są umieszczane w magazynie obiektów.

## Indeks

Indeks jest tymczasowym i dynamicznym plikiem binarnym, opisującym strukturę całego archiwum. Dokładniej mówiąc, indeks wyraża wersję ogólnej struktury projektu w danej chwili. Stan projektu można reprezentować za pomocą zatwierdzenia i drzewa, wychodząc od dowolnego punktu w historii projektu; może to być również stan przyszły, ku któremu czynnie podążasz.

Jedną z wyróżniających cech Gita jest możliwość zmieniania zawartości indeksu w metodycznych, dobrze określonych krokach. Indeks pozwala na rozdział między przyrostowymi krokami rozwojowymi a zatwierdzaniem tych zmian.

Działa to następująco. Jako budowniczy wykonujesz polecenia Gita mające na celu *wybranie* (*wystawienie*, ang. *stage*) zmian do wykonania, co jest odnotowywane w indeksie. Zmiany polegają zazwyczaj na dodaniu, usunięciu lub przeredagowaniu jakiegoś pliku lub zbioru plików. Indeks zapisuje i zachowuje te zmiany, utrzymując je bezpiecznie do czasu, aż będziesz gotowy do ich *zatwierdzenia*. Zmiany możesz również z indeksu usunąć lub zastąpić innymi. Indeks umożliwia Ci zatem stopniowe przejście, zazwyczaj zależne od Ciebie, od jednego kompleksowego stanu archiwum do drugiego, z założenia — lepszego.

Jak zobaczysz w rozdziale 9, indeks odgrywa ważną rolę w operacjach *łączenia*, umożliwiając jednocześnie zarządzanie, nadzorowanie i manipulowanie wieloma wersjami tego samego pliku.

---

<sup>4</sup> To samo pojęcie jest również określane w oryginale nazwą *packed file*; pozostajemy przy jednej — *przyj. tłum.*

## Identyfikatory globalnie jednoznaczne

Ważną cechą obliczenia haszowania SHA1 jest to, że zawsze powoduje ono wytworzenie tego samego identyfikatora dla danej treści, niezależnie od tego *gdzie* ta treść się znajduje. Innymi słowy, ta sama treść pliku w różnych katalogach, a nawet na różnych maszynach, prowadzi do powstania dokładnie tego samego identyfikatora haszowego SHA1. Tak więc haszowanie SHA1 pliku wytwarza identyfikator efektywny i globalnie jednoznaczny (unikatowy).

Istotną konsekwencją tego faktu jest możliwość sprawdzania w całym Internecie równości plików lub blobów dowolnego rozmiaru po prostu za pomocą porównania ich identyfikatorów SHA1.

## Nazwy adresowane treścią

Magazyn obiektów Gita jest zorganizowany i zrealizowany w postaci systemu pamięci adresowanej treścią. W szczególności każdy obiekt w magazynie obiektów ma niepowtarzalną nazwę wytworzoną przez potraktowanie treści obiektu algorytmem SHA1; w rezultacie powstaje wartość haszowania SHA1. Ponieważ do wytworzenia wartości haszowania jest używana cała treść obiektu i można z dużym prawdopodobieństwem przyjąć, że jest ona unikatowa dla tej konkretnej treści — wartość haszowania SHA1 starczy za indeks, tj. nazwę danego obiektu w bazie danych obiektów. Najmniejsza zmiana pliku powoduje zmianę wartości haszowania SHA1, powodując osobne zaindeksowanie nowej wersji pliku.

Wartości SHA1 są 160-bitowe i zwykle są reprezentowane w postaci 40-cyfrowej liczby szesnastkowej, jak np. 9da581d910c9c4ac93557ca4859e767f5caf5169. Niekiedy, podczas wyświetlania, wartości SHA1 są skracane do mniejszego, jednoznacznego przedrostka. Użytkownicy Gita zwykli zamiennie używać określeń *SHA1*, *kod haszowania* (ang. *hash code*), a czasami *ID obiektu*<sup>5</sup>.

## Git nadzoruje treść

Warto dojrzeć w Gicie coś więcej niż tylko system kontroli wersji. Git jest *systemem nadzorowania treści* (ang. *content tracking system*). Rozróżnienie to, aczkolwiek subtelne, wywarło duży wpływ na projekt Gita i niewykluczone, że w głównej mierze przyczyniło się do tego, że może on stosunkowo łatwo dokonywać manipulacji na wewnętrznych danych. To również jest być może jedną z najtrudniejszych koncepcji do opanowania przez nowych użytkowników Gita, warto więc poświęcić jej trochę czasu.

Gitowe śledzenie treści przejawia się na dwa zasadnicze sposoby, co stanowi podstawową różnicę w stosunku do prawie wszystkich innych<sup>6</sup> systemów kontroli uaktualnień.

Po pierwsze, magazyn obiektów Gita jest oparty na haszowaniu *treści* jego obiektów, a nie na nazwach plików lub katalogów, takich, jakimi je ogląda użytkownik. Kiedy więc Git umieszcza plik w magazynie obiektów, robi to, wykorzystując haszowanie danych, a nie nazwę pliku.

<sup>5</sup> W innych tekstach produkt podobnych algorytmów haszujących zwykł być nazywany *skrótem* lub *streszczeniem komunikatu* (ang. *message digest*); przewaga liczebna nazw nad bytami jest w informatyce powszechna — *przypp. tłum.*

<sup>6</sup> Chwalebnyymi wyjątkami są tu systemy Monotone, Mercurial, OpenCMS i Venti.

Git w rzeczywistości nie śledzi nazw plików lub katalogów, są kojarzone z plikami drugorzędnie. Powtórzmy: Git nadzoruje treść, a nie pliki.

Jeśli dwa pliki mają tę samą treść — wszystko jedno, czy występują w tym samym, czy w różnych katalogach — to Git przechowuje w magazynie obiektów jedną kopię tej treści w postaci blobu. Git oblicza kod haszowy każdego pliku, polegając wyłącznie na jego treści, ustala, że pliki mają tę samą wartość SHA1, a więc i tę samą zawartość, i umieszcza obiekt blobu w magazynie obiektów zaindeksowany tą wartością SHA1. Oba plikom w projekcie, niezależnie od ich usytuowania w strukturze katalogowej użytkownika, odpowiada ten sam obiekt ich treści.

Jeśli jeden z tych plików zostanie zmieniony, Git oblicza dla niego nową wartość SHA1, ustala, że jest on teraz innym obiektem blobu, i dodaje nowy blob do magazynu obiektów. Pierwotny blob pozostaje w magazynie obiektów na wypadek użycia niezmienionego pliku.

Po drugie, wewnętrzna baza danych Gita efektywnie przechowuje każdą wersję każdego pliku — nie ich różnice<sup>7</sup> — w miarę powstawania kolejnych wydań pliku. Ponieważ Git używa haszowania całej treści jako nazwy danego pliku, musi zawsze działać na kompletnej kopii pliku. Nie może opierać swej pracy ani wpisów w magazynie obiektów wyłącznie na fragmencie zawartości pliku ani na różnicach między dwoma jego wersjami.

Z punktu widzenia użytkownika pliku rzeczy zazwyczaj mają się tak, że plik ma kolejne uaktualnienia, pozostając jednym przedmiotem. Git oblicza historię tych uaktualnień jako zbiór zmian między poszczególnymi blobami o różnych haszowaniach, nie zapamiętuje natomiast nazwy pliku i zbioru zmian w sposób bezpośredni. Może to się wydawać osobliwe, lecz ta właściwość umożliwia Gitowi łatwe wykonywanie pewnych zadań.

## Nazwy ścieżek a treść

Podobnie jak wiele innych VCS-ów, Git musi utrzymywać jawną listę plików zawartych w archiwum. Konieczność ta nie wymaga jednak, aby manifest<sup>8</sup> Gita opierał się na nazwach plików. Git traktuje nazwę pliku jako część danych odrębną od treści pliku. W ten sposób oddziela indeks od danych w tradycyjnym rozumieniu bazy danych. Rzut oka na tabelę 4.1, w której z grubsza porównano Gita z innymi znanymi systemami, może się tu okazać pomocny.

Tabela 4.1. Porównanie baz danych

System	Mechanizm indeksowania	Pamięć danych
Tradycyjna baza danych	Metoda indeksowanego dostępu sekwencyjnego (ISAM)	Rekordy danych
Uniksowy system plików	Katalogi ( <i>/ścieżka/do/pliku</i> )	Bloki danych
Git	<i>.git/objects/hash</i> , treści obiektów drzew	Obiekty blobów, obiekty drzew

Nazwy plików i katalogów pochodzą z bieżącego systemu plików, lecz dla Gita nie mają one w istocie znaczenia. Git zapisuje po prostu każdą nazwę ścieżki i zapewnia możliwość dokładnego odtworzenia plików i katalogów na podstawie jej treści, indeksowanej wartością haszowania.

<sup>7</sup> W oryginale: *not their differences*, porównaj jednak uwagę o deltach w punkcie „Pliki pakowane” w tym podrzdziale — *przyp. tłum.*

<sup>8</sup> Tu: wykaz plików w bazie — *przyp. tłum.*

Fizyczny wygląd danych Gita nie odzwierciedla struktury katalogowej plików użytkownika. Posługuje się on natomiast zupełnie inną strukturą, na podstawie której można jednak odtworzyć oryginalną postać, widzianą przez użytkownika. Wewnętrzna struktura danych Gita lepiej nadaje się do jego wewnętrznych operacji i rozwiązań pamięciowych.

Kiedy Git chce utworzyć katalog roboczy, mówi systemowi plików: „Hej, mam tu do umieszczenia dużego bloba, do którego prowadzi ścieżka *droga/do/katalogu/plik*. Pojmujesz?”. System plików odpowiada: „O tak, rozpoznaję ten napis jako zestaw nazw podkatalogów i wiem, gdzie umieścić twój blob danych. Dziękuję!”.

## Pliki pakowane

Bystry czytelnik mógłby sformułować narzucające się pytania o model danych Gita i sposób przechowywania poszczególnych plików: „Czy to nie jest skrajnie nieefektywne, aby każdą wersję każdego pliku zapamiętywać bezpośrednio? Nawet jeśli się go skompresuje, czyż nie jest brakiem oszczędności utrzymywanie pełnej treści różnych wersji tego samego pliku? A jeśli, dajmy na to, dodasz do pliku tylko jeden wiersz, to Git będzie przechowywał treść obu wersji w całości?”.

Na szczęście odpowiedź brzmi: „Nie, naprawdę nie!”.

Git stosuje efektywniejszy mechanizm zapamiętywania, zwany *plikiem pakowanym*. Aby utworzyć plik pakowany, Git wynajduje pliki o bardzo zbliżonej treści i zapamiętuje całą treść jednego z nich. Następnie oblicza różnice, czyli delty, między podobnymi plikami i zapamiętuje tylko te różnice. Na przykład, gdybyś zmienił tylko jeden wiersz pliku, to Git mógłby zapamiętać w całości nowszą wersję, odnotować tę jednowierszową zmianę jako deltę i również zapamiętać ją w paczce.

Przechowywanie kompletnej wersji pliku oraz delt potrzebnych do skonstruowania innych wersji podobnych plików nie jest niczym nowym. Ten sam mechanizm jest od dziesięcioleci używany w innych VCS-ach, na przykład w systemie RCS<sup>9</sup>.

Trzeba przyznać, że Git wykonuje to pakowanie bardzo sprytnie. Ponieważ Git jest sterowany *treścią*, zupełnie nie dba o to, czy delty, które oblicza jako różnice między dwoma plikami, faktycznie odnoszą się do dwóch wersji tego samego pliku, czy nie. To znaczy Git może wziąć dowolne dwa pliki z dowolnego miejsca archiwum i obliczyć delty między nimi, jeśli uważa, że są dostatecznie podobne, aby zapewnić dobrą kompresję. Git operuje zatem dobrze dopracowanym algorytmem lokalizacji i dopasowywania potencjalnych kandydatów na delty globalnie, wskroś całego archiwum. Co więcej, Git potrafi konstruować ciągi delt z jednej wersji pliku do drugiej, trzeciej itd.

Dla każdego kompletnego pliku w pakowanej reprezentacji Git utrzymuje również wiedzę o pierwotnym kodzie SHA1 blobu (stosuje się albo pełną treść, albo rekonstrukcję według delt). Stanowi to podstawowy mechanizm indeksu, umożliwiający odnajdywanie obiektów w paczce.

Pliki pakowane są przechowywane w magazynie obiektów wraz z innymi obiektami. Używa się ich również do sprawnego przesyłania archiwów przez sieć.

---

<sup>9</sup> A jego początków można szukać jeszcze dawniej, w początku lat 70. XX wieku, kiedy to np. w systemie operacyjnym GEORGE 3 (ICL) istniał programowalny edytor tekstu: wszystkie zmiany w plikach były wyrażane w postaci ciągu jego instrukcji; przechowując plik początkowy i (małe) pliki z instrukcjami dla edytora, można było, w drodze kolejnych przetworzeń edytorem, uzyskać każdą wersję pliku — *przyp. tłum.*

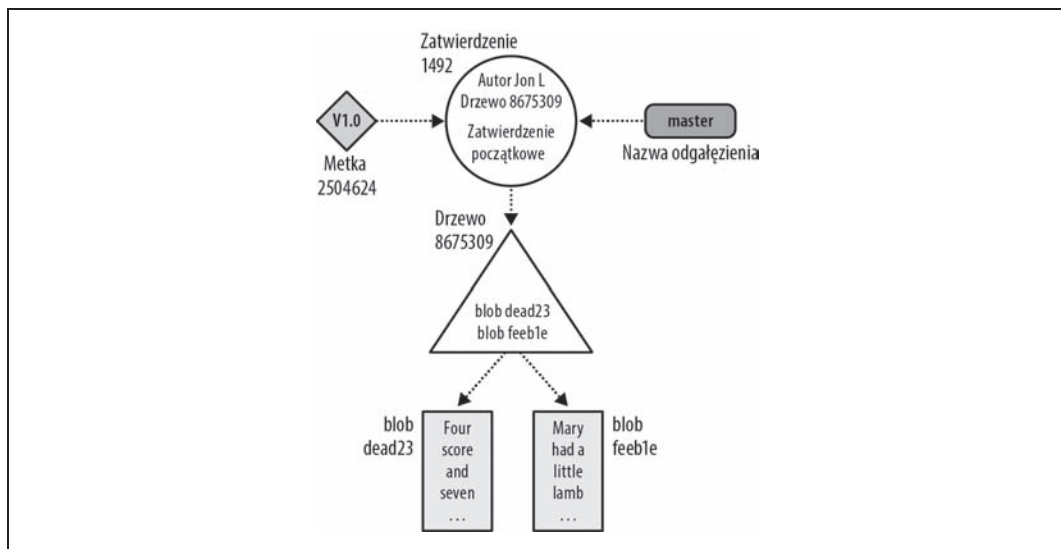


# Obrazy magazynu obiektów

Spójrzmy, jak obiekty Gita przystają do siebie i współpracują, tworząc kompletny system.

Obiekt blobu znajduje się „na dnie” struktury danych; nie wskazuje niczego, a odniesienia do niego pochodzą wyłącznie z obiektów drzewa. Na poniższych rysunkach każdy blob jest przedstawiony w postaci prostokąta.

Obiekty drzew wskazują na bloby i być może na inne drzewa. Dowolny obiekt drzewa może być wskazywany przez wiele różnych obiektów zatwierdzeń. Każde drzewo jest reprezentowane przez trójkąt.



Rysunek 4.1. Obiekty Gita

Kółko reprezentuje zatwierdzenie. Zatwierdzenie wskazuje na konkretne drzewo, wprowadzone do archiwum wskutek zatwierdzenia.

Każda *metka* (ang. *tag*) jest reprezentowana za pomocą równoległoboku. Metka może wskazywać najwyżej jedno zatwierdzenie.

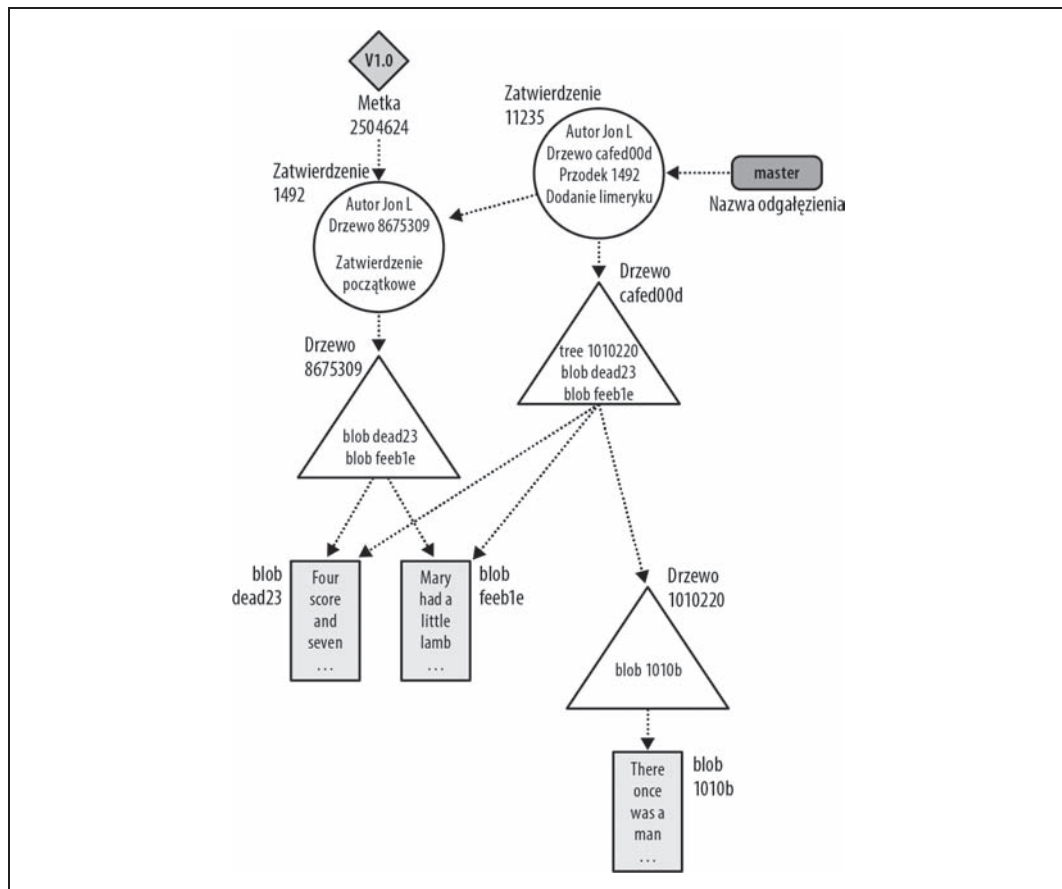
Odgałęzienie (gałąź) nie jest podstawowym obiektem Gita, lecz odgrywa zasadniczą rolę w nazywaniu zatwierdzeń. Każde odgałęzienie jest rysowane w postaci prostokąta z zaokrąglonymi wierzchołkami.

Na rysunku 4.1 pokazano, jak te wszystkie elementy mają się do siebie. Rysunek przedstawia stan archiwum po jednym, początkowym zatwierdzeniu, w którym dodano dwa pliki. Oba pliki znajdują się w szczytowym katalogu. Zarówno odgałęzienie *master*, jak i metka *V1.0* wskazują na zatwierdzenie z identyfikatorem 1492.

Skomplikujmy więc nieco sprawę. Pozostawmy dwa oryginalne pliki bez zmian i dodajmy nowy podkatalog z jednym plikiem. Wynikowy magazyn obiektów będzie wyglądał jak na rysunku 4.2.

Jak na poprzednim rysunku, nowe zatwierdzenie powoduje dodanie jednego, skojarzonego z nim obiektu drzewa do reprezentowania ogólnego stanu struktury katalogowej i plików. W tym przypadku jest to obiekt drzewa z identyfikatorem `cafed00d`.

Ponieważ szczytowy katalog uległ zmianie przez dodanie nowego podkatalogu, zmieniła się również *treść* szczytowego obiektu drzewa, toteż Git wprowadził nowe drzewo: `cafed00d`.



Rysunek 4.2. Obiekty Gita po drugim zatwierdzeniu

Blobs `dead23` i `feeb1e` nie zmieniły się jednak między pierwszym a drugim zatwierdzeniem. Git jest świadom, że ich identyfikatory nie uległy zmianie, można się więc do nich odwoływać bezpośrednio i dzielić je w nowym drzewie `cafed00d`.

Zwróć uwagę na kierunek strzałek między zatwierdzeniami. Zatwierdzenie rodzicielskie (lub więcej takich zatwierdzeń) wystąpiło wcześniej. Dlatego w implementacji Gita każde zatwierdzenie wskazuje na swojego przodka lub przodków. Wielu osobom to się myli, ponieważ stan archiwum jest na zasadzie umowy przedstawiany w odwrotnym kierunku: jako przepływ danych *od* zatwierdzenia rodzicielskiego do zatwierdzeń potomnych.

W rozdziale 6 poszerzamy te obrazy, aby pokazać, na czym polega budowanie historii archiwum i manipulowanie nią za pomocą różnych poleceń.

# Koncepcje Gita uwidocznione w działaniu

Mając za sobą omówienie ogólnych pomysłów, zobaczymy, jakie znajdują one (i odpowiednie komponenty) odzwierciedlenie w samym archiwum. Utwórzmy nowe archiwum i przyjrzymy się nieco dokładniej jego wewnętrznym plikom i magazynowi obiektów.

## Zawartość katalogu .git

Zacznijmy od zainicjowania pustego archiwum za pomocą polecenia `git init`, po czym wykonajmy polecenie `find`, aby zobaczyć, co powstało.

```
$ mkdir /tmp/hello
$ cd /tmp/hello
$ git init
Initialized empty Git repository in /tmp/hello/.git/
# Sporządź wykaz wszystkich plików bieżącego katalogu
$ find .
.
./.git
./.git/hooks
./.git/hooks/commit-msg.sample
./.git/hooks/applypatch-msg.sample
./.git/hooks/pre-applypatch.sample
./.git/hooks/post-commit.sample
./.git/hooks/pre-rebase.sample
./.git/hooks/post-receive.sample
./.git/hooks/prepare-commit-msg.sample
./.git/hooks/post-update.sample
./.git/hooks/pre-commit.sample
./.git/hooks/update.sample
./.git/refs
./.git/refs/heads
./.git/refs/tags
./.git/config
./.git/objects
./.git/objects/pack
./.git/objects/info
./.git/description
./.git/HEAD
./.git/branches
./.git/info
./.git/info/exclude
```

Jak łatwo zauważyć, katalog `.git` zawiera niemało. Pliki są wyświetlane według wzorcowego katalogu w układzie, który możesz w razie potrzeby zmienić. Zależnie od używanej przez Ciebie wersji Gita, to, co naprawdę ujrzysz, może prezentować się nieco inaczej. Na przykład w starszych wersjach Gita nie jest stosowany przyrostek `.sample` w plikach katalogu `.git/hooks`.

Ogólnie biorąc, nie musisz zaglądać do plików w `.git` ani nimi manipulować. Te „ukryte” pliki są uważane za część *instalacji* Gita, czyli jego konfiguracji. Git ma niewielki zbiór poleceń instalatorskich (ang. *plumbing*) do działania na tych ukrytych plikach, lecz korzystać z nich będziesz rzadko.

Na początku katalog `.gitobjects` (katalog przeznaczony na wszystkie obiekty Gita) jest pusty, wyjąwszy kilka wypełniaczy.

```
$ find .git/object
```

```
.git/objects  
.git/objects/pack  
.git/objects/info
```

Utwórzmy teraz starannie prosty obiekt:

```
$ echo "hello world" > hello.txt  
$ git add hello.txt
```

Jeśli wpisałeś "hello world" dokładnie tak, jak tu podano (nie zmieniając spacji ani wielkości liter), to Twój katalog obiektów powinien obecnie wyglądać tak:

```
$ find .git/objects  
.git/objects  
.git/objects/pack  
.git/objects/3b  
.git/objects/3b/18e512dba79e4c8300dd08aeb37f8e728b8dad  
.git/objects/info
```

Wszystko to wygląda dość tajemniczo, lecz tylko na pozór, co wyjaśniamy w następnych podrozdziałach.

## Obiekty, haszowania i bloby

Tworząc obiekt dla *hello.txt*, Git nie dba o to, że nazwą pliku jest *hello.txt*. Interesuje go tylko zawartość tego pliku: ciąg 12 bajtów stanowiących napis „hello world” wraz z kończącym go znakiem zmiany wiersza (sam blob, utworzony wcześniej). Git wykonuje na tym blobie kilka operacji, oblicza jego haszowanie SHA1 i wprowadza go do magazynu obiektów jako plik nazwany według szesnastkowej reprezentacji haszowania.

### Skąd wiemy, że haszowanie SHA1 jest jednoznaczne?

Istnieje nader mała szansa, że dwa różne bloby wytworzą to samo haszowanie SHA1. Jeśli tak się zdarzy, nazywamy to *kolizją*. Kolizja SHA1 jest jednak tak mało prawdopodobna, że możesz bezpiecznie przyjmować, że nigdy nie zakłóci ona naszego użytkownika Gita.

SHA1 jest „haszowaniem kryptograficznie bezpiecznym”. Do niedawna nie znano żadnego sposobu (lepszego niż zdanie się na ślepy los) spowodowania przez użytkownika kolizji na życzenie. Czy jednak kolizja mogłaby się zdarzyć losowo? Zobaczmy!

Mając 160 bitów, dysponujesz  $2^{160}$ , czyli około  $10^{48}$  (jedynka z 48 zerami po niej) możliwymi haszowaniami SHA1. Jest to liczba niewyobrażalnie wielka. Nawet gdybyś wynajął bilion ludzi do produkowania biliona nowych blobów na sekundę przez bilion lat, otrzymalibyś dopiero około  $10^{43}$  blobów.

Gdybyś pohaszował  $2^{80}$  losowych blobów, mógłbyś znaleźć kolizję.

Nie wierz nam. Poczytaj Bruce’a Schneiera<sup>10</sup>.

<sup>10</sup> Zobacz: Bruce Schneier, *Kryptografia dla praktyków. Protokoły, algorytmy i programy źródłowe w języku C*. Wydawnictwa Naukowo-Techniczne, Warszawa 2002 — przyp. tłum.

Haszowaniem w tym wypadku jest `3b18e512dba79e4c8300dd08aeb37f8e728b8dad`. 160 bitów haszowania SHA1 odpowiada 20 bajtom, do wyświetlenia których w kodzie szesnastkowym trzeba 40 bajtów, treść ta jest zatem zapamiętana jako `.git/objects/3b/18e512dba79e4c8300dd08aeb37f8e728b8dad`. Git wstawia znak `/` po pierwszych dwóch cyfrach, aby polepszyć efektywność systemu plików. (Niektóre systemy plików zwalniają, jeśli umieścisz zbyt wiele plików w tym samym katalogu; zrobienie z pierwszego bajta SHA1 nazwy katalogu jest prostym sposobem utworzenia stałego, 256-drożnego podziału przestrzeni nazw wszystkich możliwych obiektów z równomiernym rozkładem).

Aby pokazać, że Git naprawdę niewiele zrobił z treścią pliku (jest nią wciąż podnoszące na duchu „hello world”), możesz posłużyć się tym haszowaniem do wydobycia jej z magazynu obiektów, kiedy tylko zechcesz:

```
$ git cat-file -p 3b18e512dba79e4c8300dd08aeb37f8e728b8dad
hello world
```



Git wie również, że ręczne wpisywanie 40 znaków jest nieco ryzykowne, udostępnia więc polecenie do poszukiwania obiektów za pomocą jednoznacznego przedrostka haszowania obiektu:

```
$ git rev-parse 3b18e512d
3b18e512dba79e4c8300dd08aeb37f8e728b8dad
```

## Pliki i drzewa

Kiedy już blob „hello world” został bezpiecznie rozlokowany w magazynie obiektów, co stało się z jego nazwą pliku? Niewiele byłoby pożytku z Gita, gdyby nie potrafił znajdować plików na podstawie ich nazw.

Wspomnieliśmy, że Git śledzi nazwy ścieżek plików za pomocą obiektu innego rodzaju, nazywanego *drzewem*. Gdy używasz `git add`, Git tworzy obiekt treści każdego dodawanego przez Ciebie pliku, lecz nie od razu tworzy obiekt Twojego drzewa. Zamiast tego uaktualnia *indeks*. Indeks znajduje się w `.git/index` i rejestruje nazwy ścieżek plików i odpowiednich blobów. Ilekroć wydajesz polecenie w rodzaju `git add`, `git rm`, `git mv`, Git aktualizuje indeks, używając nowych ścieżek i informacji blobowej.

Kiedy tylko zechcesz, możesz utworzyć obiekt drzewa na podstawie bieżącego indeksu, tworząc migawkowe ujęcie jego bieżącej informacji za pomocą niskopoziomowego polecenia `git write-tree`.

W danej sytuacji indeks zawiera tylko jeden plik: *hello.txt*.

```
$ git ls-files -s
100644 3b18e512dba79e4c8300dd08aeb37f8e728b8dad 0      hello.txt
```

Możesz tu zaobserwować powiązanie pliku *hello.txt* z blobem `3b18e5...`

Uchwyćmy teraz stan indeksu i przechowajmy go w obiekcie drzewa:

```
$ git write-tree
68aba62e560c0ebc3396e8ae9335232cd93a3f60

$ find .git/objects
.git/objects
.git/objects/68
.git/objects/68/aba62e560c0ebc3396e8ae9335232cd93a3f60
```

```
.git/objects/pack
.git/objects/3b
.git/objects/3b/18e512dba79e4c8300dd08aeb37f8e728b8dad
.git/objects/info
```

Istnieją obecnie dwa obiekty: „hello world” w 3b18e5 i nowy — obiekt drzewa w 68aba6. Jak widać, nazwa SHA1 obiektu odpowiada dokładnie podkatalogowi i plikowi w `.git/objects`.

Ale jak wygląda drzewo? Ponieważ jest obiektem, podobnie jak blob, możesz do jego obejrzenia użyć tego samego niskopoziomowego polecenia.

```
$ git cat-file -p 68aba6
100644 blob 3b18e512dba79e4c8300dd08aeb37f8e728b8dad    hello.txt
```

Zawartość tego obiektu powinna być łatwa do zinterpretowania. Pierwsza liczba, 100644, reprezentuje ósemkowo atrybuty pliku, co powinno być znane każdemu, kto używał uniksowego polecenia `chmod`. Z kolei 3b18e5 jest nazwą obiektu blobu *hello world*, a *hello.txt* jest nazwą stowarzyszoną z tym blobem.

Łatwo teraz zauważyć, że obiekt drzewa ujmuje informacje, które były w indeksie, gdy wykonywałeś polecenie `git ls-files -s`.

## Uwaga o zastosowaniu w Gicie algorytmu SHA1

Nim przyjrzymy się dokładniej obiektowi drzewa, zwróćmy uwagę na ważną cechę haszowań SHA1:

```
$ git write-tree
68aba62e560c0ebc3396e8ae9335232cd93a3f60
```

```
$ git write-tree
68aba62e560c0ebc3396e8ae9335232cd93a3f60
```

```
$ git write-tree
68aba62e560c0ebc3396e8ae9335232cd93a3f60
```

Każdorazowo, gdy obliczasz kolejny obiekt drzewa według tego samego indeksu, haszowanie SHA1 pozostaje bez zmian. Git nie musi odtwarzać nowego obiektu drzewa. Jeśli wykonasz powyższe kroki na komputerze, powinieneś zobaczyć *dokładnie te same haszowania SHA1* co wydrukowane w książce.

W tym sensie haszowanie jest prawdziwą funkcją w rozumieniu matematycznym: dla danej wartości wejściowej zawsze wytwarza ten sam wynik. Funkcja taka bywa czasami nazywana *streszczeniem* (ang. *digest*), aby podkreślić, że służy jako pewien sposób uzwięźlenia haszowanego obiektu. Oczywiście każda funkcja haszująca, nawet skromniutki bit parzystości, ma tę właściwość.

Jest to niesłychanie ważne. Jeśli na przykład utworzysz taką samą treść jak inny budowniczy, to niezależnie od tego kiedy lub jak obaj (lub oboje) pracowaliście, jednakowe haszowanie dowodzi w stopniu wystarczającym, że również obie (i całe) treści są identyczne. I rzeczywiście, Git będzie je uważał za jednakowe.

Przejdźmy jednak do drugiego — przecież haszowania SHA1 miały być unikatowe! Jak to jest z tym bilionem ludzi i ich bilionem blobów na sekundę, którzy nigdy nie wyprodukują ani jednej kolizji? Jest to typowe źródło nieporozumień wśród nowych użytkowników Gita. Czytaj więc dalej uważnie, bo jeśli zrozumiesz tę różnicę, to cała reszta w tym rozdziale będzie łatwa.

Identyczne w danym wypadku haszowanie SHA1 *nie stanowią kolizji*. Do kolizji doszłoby tylko wówczas, gdyby dwa *różne* obiekty dały w wyniku to samo haszowanie. Tutaj utworzyłeś dwa egzemplarze tej samej treści, a ta sama treść ma zawsze to samo haszowanie.

Git opiera się na jeszcze innej konsekwencji funkcji haszowania SHA1: nieważne, *jak* otrzymałeś drzewo o nazwie `68aba62e560c0ebc3396e8ae9335232cd93a3f60`. Jeśli je masz, możesz być ponad wszelką wątpliwość pewny, że jest to ten sam obiekt drzewa co ten, który ma — dajmy na to — inny czytelnik tej książki. Benek mógł utworzyć drzewo z kombinacji zatwierdzeń A i B od Janki i zatwierdzenia C od Sergiusza, a Ty na podstawie zatwierdzenia A od Zuzi i uaktualnienia od Lakshmi, która połączyła zatwierdzenia B i C. Wyniki są takie same, a to umożliwia prowadzenie prac w rozproszeniu.

Jeśli zostaniesz poproszony o wgląd w obiekt `68aba62e560c0ebc3396e8ae9335232cd93a3f60` i zdołasz taki obiekt znaleźć, to na mocy tego, że SHA1 jest haszowaniem kryptograficznym, możesz być pewny, że patrzysz na te same dane, z których utworzono to haszowanie.

Odwrotność jest również prawdziwa: jeśli nie znajdziesz w swoim magazynie obiektu o danym haszowaniu, to możesz mieć pewność, że nie dysponujesz kopią tego właśnie obiektu. Podsumujmy: możesz określić, czy magazyn obiektów zawiera konkretny obiekt, czy nie, nawet wówczas, gdy nie wiesz nic o jego (być może bardzo obszernej) treści. Haszowanie służy więc jako niezawodna etykieta lub nazwa obiektu.

Git jednak polega jeszcze na czymś mocniejszym niż powyższa konkluzja. Rozważmy najnowsze zatwierdzenie (lub skojarzony z nim obiekt drzewa). Ponieważ jako część treści zawiera ono haszowanie zatwierdzeń rodzicielskich i jego drzewa, a *to* z kolei zawiera haszowanie wszystkich jego poddrzew i blobów — rekurencyjnie wskroś całej struktury danych, przez indukcję wynika z tego, że haszowanie oryginalnego zatwierdzenia jednoznacznie identyfikuje stan całej struktury danych ukorzenionej w danym zatwierdzeniu.

Implikacje naszego stwierdzenia z poprzedniego akapitu prowadzą na koniec do mocnego zastosowania funkcji haszowania: stanowi ona efektywny środek porównywania dwóch obiektów, w tym nawet bardzo dużych i złożonych struktur danych<sup>11</sup>, bez przekazywania żadnej z nich w całości.

## Hierarchie drzewiaste

Dobrze jest mieć informacje dotyczące pojedynczego pliku, jak pokazaliśmy w poprzednim podrozdziale, lecz przedsięwzięcia są złożone z głęboko zagnieżdżonych katalogów, refaktoryzowanych i przemieszczanych z biegiem czasu. Zobaczmy, jak Git radzi sobie z tym, tworząc nowy podkatalog, który będzie zawierał identyczną kopię pliku `hello.txt`.

```
$ pwd
/tmp/hello
$ mkdir subdir
$ cp hello.txt subdir/
$ git add subdir/hello.txt
$ git write-tree
492413269336d21fac079d4a4672e55d5d2147ac

$ git cat-file -p 4924132693
100644 blob 3b18e512dba79e4c8300dd08aeb37f8e728b8dad hello.txt
040000 tree 68aba62e560c0ebc3396e8ae9335232cd93a3f60 subdir
```

<sup>11</sup> O tej strukturze danych mówimy więcej w podrozdziale „Grafy zatwierdzeń” na stronie 74 rozdziału 6.

Nowe drzewo szczytowego poziomu zawiera dwa elementy: oryginalny plik *hello.txt* i nowy katalog *subdir*, który jest typu *drzewo*, a nie *blob*.

Czy zauważamy coś niezwykłego? Przypatrzmy się uważniej nazwie obiektu *subdir*. To Twoja stara znajoma, 68aba62e560c0ebc3396e8ae9335232cd93a3f60!

Co się stało? Nowe drzewo dla *subdir* zawiera tylko jeden plik, *hello.txt*, a ten plik zawiera tę samą co poprzednio treść „hello world”. Wobec tego drzewo *subdir* jest identyczne z poprzednim drzewem szczytowego poziomu! I oczywiście ma tę samą nazwę SHA1 obiektu co przedtem.

Spójrzmy na katalog *.git/objects* i zobaczmy, co spowodowała ostatnia zmiana:

```
$ find .git/objects
.git/objects
.git/objects/49
.git/objects/49/2413269336d21fac079d4a4672e55d5d2147ac
.git/objects/68
.git/objects/68/aba62e560c0ebc3396e8ae9335232cd93a3f60
.git/objects/pack
.git/objects/3b
.git/objects/3b/18e512dba79e4c8300dd08aeb37f8e728b8dad
.git/objects/info
```

Nadal są w nim tylko trzy *unikatowe* obiekty: *blob* zawierający „hello world”, drzewo zawierające plik *hello.txt*, który zawiera tekst „hello world” i zmianę wiersza, oraz drugie drzewo, zawierające *drugie* odniesienie do *hello.txt*, wraz z pierwszym drzewem.

## Zatwierdzenia

Następnym obiektem do omówienia jest *zatwierdzenie* (ang. *commit*). Obecnie, po dodaniu *hello.txt* za pomocą `git add` i wyprodukowaniu za pomocą `git write-tree` obiektu drzewa, możesz utworzyć obiekt zatwierdzenia, korzystając z polecenia niskiego poziomu, na przykład tak:

```
$ echo -n "Zatwierdzenie pliku z powitaniem\n" \
| git commit-tree 492413269336d21fac079d4a4672e55d5d2147ac
3ede4622cc241bcb09683af36360e7413b9ddf6c
```

Wynik przybierze postać mniej więcej taką:

```
$ git cat-file -p 3ede462
tree 492413269336d21fac079d4a4672e55d5d2147ac
author Jon Loeliger <jdl@example.com> 1220233277 -0500

committer Jon Loeliger <jdl@example.com> 1220233277 -0500
```

Zatwierdzenie pliku z powitaniem

Jeśli dorównujesz nam kroku na swoim komputerze, prawdopodobnie zauważysz, że wygenerowany przez Ciebie obiekt zatwierdzenia *nie* ma tej samej nazwy co w książce. Jeżeli rozumiałeś wszystko do tej pory, przyczyna będzie oczywista — to nie jest to samo zatwierdzenie. Twoje zatwierdzenie zawiera Twoje nazwisko i czas, w którym go dokonałeś, różni się zatem, choć jest to różnica niewielka. Z drugiej strony, w Twoim zatwierdzeniu występuje to samo *drzewo*. To właśnie powoduje odrębność obiektów zatwierdzeń od ich obiektów drzew: różne zatwierdzenia często odnoszą się do tego samego drzewa. Kiedy się tak zdarzy, Git wykazuje na tyle inteligencji, aby przekazywać dalej tylko nowy obiekt zatwierdzenia, który jest mały, a nie obiekt drzew i blobów, które są zapewne znacznie większe.



W praktyce możesz (a nawet powinieneś!) omijać niskopoziomowe kroki `git write-tree` i `git commit-tree` i używać po prostu polecenia `git commit`. Nie musisz pamiętać wszystkich tych instalatorskich poleceń, żeby być świetnym użytkownikiem Gita.

Podstawowy obiekt zatwierdzenia jest dość prosty i jest ostatnim składnikiem wymaganym w prawdziwym systemie kontroli uaktualnień. Pokazany właśnie obiekt zatwierdzenia jest najprostszym z możliwych i zawiera:

- nazwę obiektu drzewa, która faktycznie identyfikuje stowarzyszone pliki;
- nazwisko osoby, która utworzyła nową wersję (autora) i czas utworzenia;
- nazwisko osoby, która umieściła nową wersję w archiwum (zatwierdzającego) i czas dokonania zatwierdzenia;
- opis powodów wykonania danego uaktualnienia (komunikat zatwierdzenia).

Domyślnie przyjmuje się, że autor i zatwierdzający to ta sama osoba; w nielicznych sytuacjach mogą to być różne osoby.



Aby zobaczyć dodatkowe szczegóły danego zatwierdzenia, możesz użyć polecenia `git show --pretty=fuller`.

Obiekty zatwierdzeń są również zapamiętywane w strukturze grafu, zupełnie odmiennej od struktury używanej przez obiekty drzew. Gdy dokonujesz nowego zatwierdzenia, możesz dla niego określić jedno lub więcej zatwierdzeń *rodzicielskich*. Cofając się w łańcuchu przodków, możesz odkryć historię swojego przedsięwzięcia. Więcej szczegółów o zatwierdzeniach i grafie zatwierdzeń podano w rozdziale 6.

## Metki

Ostatnim obiektem, którym Git rozporządza, jest *metka* (ang. *tag*). Chociaż Git realizuje tylko jeden rodzaj obiektu metki, istnieją dwa podstawowe typy metek, zazwyczaj nazywane *lekką* (ang. *lightweight*) i *skomentowaną* (ang. *annotated*).

Lekkie metki są po prostu odniesieniami do obiektu zatwierdzenia i zwykle uważa się je za prywatne w archiwum. Metki te nie tworzą stałego obiektu w magazynie obiektów. Metka skomentowana jest ważniejsza i tworzy obiekt. Zawiera sformułowany przez Ciebie komunikat i może być podpisana cyfrowo z użyciem klucza GnuPG, zgodnie z określeniami zawartymi w dokumencie RFC4880.

Na użytek nazywania zatwierdzenia Git traktuje nazwy metek lekkich i skomentowanych równoważnie. Jednak domyślnie wiele poleceń Gita działa tylko na metkach skomentowanych, ponieważ są one uważane za obiekty „trwałe”.

Metkę skomentowaną i niepodpisaną, zawierającą komunikat dotyczący zatwierdzenia, możesz utworzyć poleceniem `git tag`:

```
$ git tag -m "Tag version 1.0" V1.0 3ede462
```

Obiekt metki możesz obejrzyć za pomocą polecenia `git cat-file -p`, jakie jest jednak SHA1 obiektu metki? Aby je znaleźć, skorzystaj ze wskazówki w podrozdziale „Obiekty, haszowania i bloby”:

```
$ git rev-parse V1.0
6b608c1093943939ae78348117dd18b1ba151c6a

$ git cat-file -p 6b608c
object 3ede4622cc241bcb09683af36360e7413b9ddf6c
type commit
tag V1.0
tagger Jon Loeliger <jdl@example.com> Sun Oct 26 17:07:15 2008 -0500

Tag version 1.0
```

Oprócz komunikatu dziennika i informacji o autorze, metka odwołuje się do zatwierdzenia 3ede462. Git na ogół opatruje konkretne zatwierdzenie metką wywodzącą się od nazwy jakiegoś odgałęzienia. Zwróćmy uwagę, że to zachowanie jest istotnie różne niż w innych VCS-ach.

Git metkuje (etykietuje) na ogół obiekt zatwierdzenia, który wskazuje obiekt drzewa obejmującego sumaryczny stan całej hierarchii plików i katalogów w Twoim archiwum.

Przypomnijmy za rysunkiem 4.1, że metka V1.0 wskazuje na zatwierdzenie 1492, które z kolei wskazuje na drzewo (8675309) obejmujące wiele plików. W ten sposób metka odnosi się jednocześnie do wszystkich plików tego drzewa.

Wygląda to inaczej niż na przykład w systemie CVS, w którym metki odnoszą się do poszczególnych plików i na podstawie kolekcji wszystkich tych metkowanych plików rekonstruuje się całą metkowaną wersję. Tak więc, podczas gdy CVS pozwala Ci na przenoszenie metek na indywidualny plik, Git wymaga nowego zatwierdzenia, obejmującego zmianę stanu pliku, które zostanie opatrzone daną metką.

- --all, 330
- \$GIT\_DIR, 80
- .dotest, 268
- .git/config, 39, 196
- .git/FETCH\_HEAD, 81
- .git/hooks, 275
- .git/info/exclude, 71
- .git/logs, 192
- .git/logs/refs/, 192
- .git/logs/refs/heads, 192
- .git/MERGE\_HEAD, 140
- .git/MERGE\_MSG, 140
- .git/rebase-apply, 268
- .git/svn, 322
- .gitignore, 61, 69, 321
  - a klonowanie, 70
  - format, 70
  - GitHub, 370
  - reguły, 70
  - rozbiór złożonej hierarchii, 70
  - wzorce nazw plików, 70
- .gitmodules, 298
- .rr-cache, 366
- .sample, 51, 277
- /tmp/Depot, 203
- [ARGS], 32
- ~/ .gitconfig, 39
- ~/bin/, 26
- ~/lib/, 26
- ~/share/, 26
- 1, 85
- 3, 270
- 3way, 270

## A

- a, 119
- abbrev-commit, 84
- Access Control List, 283
- ACL, 283
- add, 34
  - plik, 34
- adopcje, 307
- adres tylko do czytania, 304
- adres archithuba, 236
- agent przesyłania poczty, 266
- agent użytkownika poczty, 264
- aktualizowanie refów, 363
- algorytm SHA1
  - zastosowanie w Gicie, 54
- alias, 41
- aliteracyjna nazwa, 366
- all, 185, 260, 365
- already up-to-date, 146, 149
- alternatywna ścieżka rozwoju, 253
- alternatywne historie
  - konflikty łączenia, 218
  - łączenie, 218
  - pobieranie, 217
  - wypychanie połączonej historii, 219
- amend, 32, 168
- amendment, 169
- annotated, 57
- anonimowe pobieranie, 24
- API, 307
- API GitHuba, 390
  - wywołanie, 391
- applypatch-msg, 281
- approximate(), 332
- arbiter, 394

- archiwum, 43
  - alternatywne historie, 217
    - konflikty łączenia, 218
    - łączenie, 218
    - pobieranie, 217
  - bezpośrednie odniesienie do innego archiwum, 295
  - bieżące, 196
  - czyste, 227
  - czyszczenie, 361
  - doczepki, 275, 283
  - dodawanie pliku, 34
  - domyślna gałąź, 100
  - dostępne do eksportu, 232
  - gałąź nadzorująca, 100
  - git push, 206
  - historia zdalna, 217
  - historia zmian, 36
    - refów, 163
  - importowanie historii, 291
  - indeks, 44, 45
  - klon, 193
  - klonowanie, 39, 44, 197
  - kodu powszechnie dostępnego, 367
  - kopia, 193
  - lokalne, 196
  - łaty, 256
  - magazyn obiektów, 44
    - obrazy, 49
  - mechanizm indeksu, 48
  - nazwy adresowane treścią, 46
  - nazwy ścieżek a treść, 47
  - nazwy użytkowników, 305
  - odgałężenia tematyczne, 197
  - odzyskiwanie z przebazowania w górze, 358
  - optymalizowanie rozmiaru, 353
  - pliki pakowane, 48
  - podmoduły, 285
  - podział, 354
  - powiązania, 244
  - przekształcenie w archiwum w górze, 249
  - przemianowywanie plików, 38
  - przeszukiwanie, 94, 361
  - publikowanie, 194, 230
  - reflog, 189
  - reorganizacja, 249
  - rozpatrywanie czasu, 191
  - rozpoczynanie, 248
  - rozwidlenia
    - GitHub, 375
    - skrypty klonowania podprojektów, 294
    - skrytka stash, 181
    - specyfikator odniesienia, 198, 200
    - struktura, 237
    - struktury danych, 44
    - szybki przegląd zmian, 360
    - tworzenie, 33
      - w GitHubie, 369
    - tworzenie archiwum zdalnego, 204
    - tworzenie kopii, 39
    - typy obiektów Gita, 44
    - uaktualnienie, 205
    - upublicznienie, 194
    - usuwanie gałęzi zdalnego nadzorowania, 221
    - usuwanie pilota, 221
    - usuwanie plików, 38, 65
    - utrwalanie zmian, 77
    - utrzymywanie, 372
    - w hierarchii katalogów, 33
    - w lokalnym systemie plików, 198
    - w składzie, 202, 209
    - wyciągi częściowe, 286
    - wymiana danych, 256
    - wyosobnione HEAD, 94
    - wypychanie połączonej historii, 219
    - wypychanie zmian, 206, 216
    - z anonimowym dostępem do czytania, 231
    - z anonimowym dostępem do pisania, 235
    - z kodem źródłowym SVN, 312
    - z kontrolowanym dostępem, 230
    - zabrudzony indeks, 130
    - zabrudzony katalog roboczy, 130
    - zagnieżdżone, 301
    - zarządzanie, 229
    - zatwierdzenie pliku, 34
    - zawierające zdalne odniesienie, 205
    - zbiór wartości konfiguracyjnych, 44
    - zmienianie historii, 157
  - archiwum dzielone, 193, 237
  - archiwum portiera, 317
    - rekonstruowanie, 322
    - tworzenie, 318
    - upublicznianie, 319
  - archiwum rozproszone, 238
    - pielegnator, 238
    - pośrednie, 238
  - archiwum scentralizowane, 237
  - archiwum systemu Subversion, 309
    - ciągnięcie, 315
    - działanie, 319

- identyfikatory zatwierdzeń, 316
- klonowanie wszystkich gałęzi, 317
- łączenie, 315
- plytki klon odgałęzienia, 309
- ponowne włączanie do SVN, 320
- rozgałęzianie, 315
- utrzymanie równoległe z archiwum Gita, 356
- wypychanie, 315
- archiwum w dole, 244
- archiwum w górze, 196, 197, 203, 244
- archiwum wzorcowe, 203
  - dodawanie nowego wykonawcy, 207
  - tworzenie, 203
- archiwum zdalne, 193, 196
  - archiwum wzorcowe, 203
  - cykl rozwojowy, 214
  - czyste, 194, 203, 227
    - kopia treści w górze, 195
  - development, 194
  - doczepki, 228, 275
  - gałąź lokalnego nadzorowania, 194, 197, 215, 222
  - gałąź zdalnego nadzorowania, 194, 195, 196, 197, 205, 215, 222
  - git clone, 214
  - git config, 221
  - git fetch, 196
  - git fls-remote, 196
  - git init, 195
  - git ls-remote, 201
  - git merge, 210, 212
  - git pull, 196, 201, 210
  - git push, 196, 201, 202, 206, 226, 227
  - git rebase, 210, 212
  - git remote, 220
  - gołe, 194
  - klonowanie, 214
  - klony, 195, 196
  - koncepcje, 194
  - konfigurowanie zdalne, 219
    - obróbka ręczna, 222
  - krok łączenia, 212
  - krok pobierania, 211
  - krok przebazowania, 212
  - nadzorowanie gałęzi, 211
  - nonbare, 194
  - odgałęzienia bieżące, 194
    - udostępniona kopia, 194
  - odgałęzienia nadzorowania, 197, 222
  - odgałęzienia zdalne
    - dodawanie i usuwanie, 226
    - odwoływanie do innych archiwów, 198
    - piloty, 193, 196, 198
    - pobieranie uaktualnień, 209
    - problem niespiesznego wypychania, 217
    - rdzenny protokół Gita, 199
    - refsPEC, 200
    - rozwojowe, 194, 208, 215, 228
    - specyfikator odniesienia, 200
    - użycie, 202
    - wiele archiwów, 222
    - wypychanie, 227
    - zdalny początek, 202
    - zwykle, 194
  - argument
    - pojedyncza kropka, 34
  - ASCII
    - refy, 260
  - assemblies, 302
  - assume-unchanged, 365
  - author, 170
  - authors-file, 311, 316
  - autobuilder, 281
  - automatyzacja łączenia, 151
  - autor zatwierdzenia, 35

## B

  - b, 111, 224
  - badanie konfliktów, 136
    - znaczniki różnic, 136
  - bare, 194, 203
  - baza danych, 47
  - baza łączenia, 124, 145
  - bezpieczna powłoka, 29
  - bezwzględne nazwy zatwierdzeń, 79
  - biblioteki
    - importowanie a zmiany w plikach, 289
    - importowanie do podkatalogu w projekcie, 287
    - zestawianie projektów, 286
  - binarny sterownik łączenia, 151
  - binary, 151
  - binary large object, 44
  - bit grupowego bezpiecznego umocowania, 237
  - BitKeeper, 18, 21
  - blobs, 44
  - bloby, 44, 47, 49, 52
    - bez odniesień, 114
    - dodawanie pliku do indeksu, 348
  - błędy typograficzne, 168

branch, 99  
branch.autosetupmerge, 213  
branch.autosetuprebase, 213  
--branches, 318  
budowanie Gita, 26  
budowniczy, 244  
build, 305

## C

--cached, 117  
caching a file, 62  
cafed00d, 50  
CAS, 395  
cat, 40  
--cc, 267  
centralne usługi uwierzytelniania, 395  
centralny magazyn, 20  
check-out, 223  
chmod, 233  
ciąg lat, 154  
clean, 35  
clone origin, 249  
--color, 119  
combined diff, 143  
commit, 34, 73  
commit range, 89  
commit-filter, 324  
commit-msg, 276, 281  
commits, 44  
committer, 45  
Concurrent Version System, 21  
conflicted, 136  
content tracking system, 46  
contrib, 278  
core.logAllRefUpdates, 189  
criss-cross merge, 145  
CRLF, 29  
CSV  
    schemat blokowania, 21  
    wyciągi częściowe, 286  
curl-config, 26  
CVS, 18, 21  
    importowanie nowych modułów, 288  
Cygwin, 27  
    instalowanie pakietu Git, 28  
    powłoka systemu, 29  
Cygwin Bash Shell, 28  
czubek, 102  
czyszczenie, 361

## D

-D, 114  
DAG, 86  
dangling, 345  
dashboard, 235  
dcommit, 315  
demon, 24  
demon HTTP, 231  
    upublicznianie archiwów, 233  
depot, 202  
dereference, 297  
detached HEAD, 94  
developer, 244  
development branch, 100, 197  
dif, 115, 127, 184  
    kombinowany, 143  
    sterowniki, 151  
    zatwierdzenia łączenia, 143  
diff, 115, 358  
diff drivers, 151  
diff -r, 39, 116  
diffs, 12  
digest, 54  
directed acyclic graph, 86  
DISPLAY, 96  
DNS, 384  
doczepki, 228, 275  
    .sample, 277  
    applypatch-msg, 281  
    błędny skrypt, 277  
    commit-msg, 276, 281  
    do lokalnego archiwum, 283  
    dostępne, 280  
    git commit --amend, 280  
    git help hooks, 280  
    instalowanie, 277  
    katalog szablonów, 277  
    konsekwencje użycia, 276  
    lokalne, 275  
    obejście, 279  
    odblokowanie, 278  
    poprzedzająca, 275  
    post-applypatch, 281  
    post-checkout, 276, 283  
    post-commit, 281  
    post-merge, 283  
    post-receive, 283  
    post-update, 276, 283  
    powiązane z latami, 281

- powiązane z wypychaniem, 282
- powiązane z zatwierdzeniami, 280
- powielanie, 277
- pre-applypatch, 281
- pre-commit, 276, 278, 280
- prepare-commit-msg, 281
- pre-rebase, 283
- pre-receive, 282
- przednia, 275
- przeгляд, 276
- przykładowe, 277
- skrypty, 277
- szablonowe, 277
- tworzenie, 278
- tylna, 275
- update, 276, 282
- w zdalnym katalogu, 275
- wykonywalne, 277
- występująca po, 275

dopasowywanie archiwum, 249

downstream, 244

downstream consumer, 247

downstream producer/publisher, 247

drobnoziarnista historia realna, 156

--dry-run, 361

drzewa, 44, 49, 53

- obiektów, 116
- odsyłacze do zatwierdzeń, 295
- pełne przekształcenie, 78
- porównywanie w poleceniu git diff, 117

dumb, 199

duże repo, 307

duży obiekt binarny, 44

DVCS, 129

dydaktyczna historia realna, 156

dziennik odniesień, 189

dziennik zatwierdzeń

- zapisywanie komunikatów, 65

## E

- e, 342
- echo -n, 56
- edytor Ace, 386
- edytor kodu w przeglądarce, 386
- emerge, 24
- env-filter, 324
- etc/gitconfig, 40
- Existing Git repo, 236
- expat.h, 26
- export-all, 232

- ext, 295
- externals, 295

## F

- f, 327
- fast-forward, 146, 149
- fetch, 194, 211
- FETCH\_HEAD, 81
- file system chec, 345
- filtr plama-wyczyść, 283
- filtrowanie, 323
- filtrowanie gałęzi, 330
- filtry, 323
  - cat, 324
  - commit-filter, 324
  - env-filter, 324
  - index-filter, 324
  - kolejność wykonywania, 324
  - msg-filter, 324
  - parent-filter, 324
  - subdirectory-filter, 325
  - tag-name-filter, 324
  - tree-filter, 324, 326
- find, 51
- first-parent, 153
- flags, 26
- folder poczty, 265
- follow, 68, 364
- force, 67
- force, 327
- forge, 372
- forking, 252, 374
- format .gitignore, 70
  - nazwa katalogu, 70
  - pojedyncza nazwa pliku, 70
  - puste wiersze, 70
  - wykrzyknik, 70
- format .headers, 264
- forward porting, 171
- framework, 306
- fsck, 345
- funkcja haszująca, 54
- funkcje bezpiecznego haszowania, 19

## G

- gałąź, 49, 99
  - filtry, 323
  - git-svn, 314, 315

- gałąź
  - lokalnego nadzorowania, 194, 197, 215, 222
    - nowe zatwierdzenia, 225
  - przebazowanie, 213
  - master, 100
  - nadzorująca, 100
  - oparta na gałęzi nadzorowania, 222
  - proponowanych uaktualnień, 241
  - przetwarzanie, 323
  - rozwojowa, 197
  - scalania, 100
  - tematyczna, 197
  - zdalna, 197
  - zdalnego nadzorowania, 194, 195, 196, 197, 205, 215, 222
    - nowe zatwierdzenia, 225
    - szybkie przekazanie, 207
    - usuwanie z archiwum, 221
    - wyciągnięcie, 223
- garbage collection, 353
- gc.auto, 354
- gc.autopacklimit, 354
- gc.pruneexpire, 354
- gc.pruneExpire, 114
- gc.reflogexpire, 354
- gc.reflogExpire, 114, 192
- gc.reflogexpireunreachable, 354
- gc.reflogExpireUnreachable, 192
- gecos, 35
- generowanie łań, 257
- gettext, 26
- git, 24, 31
- Git, 17
  - archiwum, 33
  - czasowe następstwo zatwierdzeń, 87
  - do witryn, 384
  - dokumentacja online, 32
  - identyfikowanie użytkownika, 311
  - instalowanie, 23
  - kompletność archiwów, 20
  - kompletność i pewność, 19
  - łączenia, 151
  - łączenie asymetrycznie, 132
  - manifest, 47
  - mechanizm zapamiętywania, 48
  - model archiwum rozproszonego, 238
  - model obiektowy, 151
  - nadzorowanie treści, 46
  - nadzorowanie zmiany nazw plików, 68
  - niepodzielność działań, 78
  - oparty na środowisku Cygwin, 27
  - partnerskie składowanie, 243
  - podporadzenia, 31
  - podstawa struktury danych, 44
  - podstawowe koncepcje, 43
  - poprzednicy, 20
  - porównanie baz danych, 47
  - powstanie, 18
  - protokół przesyłania, 255
  - przejrzysty projekt wewnętrzny, 20
  - stałe obiekty danych, 19
  - stałość, 19
  - system rozgałęziania, 99
  - transakcje niepodzielne, 19
  - tworzenie wykazów różnic, 116
  - wielotorowość prac rozwojowych, 19
  - wprowadzenie do użytkowania, 33
  - wymuszanie odpowiedzialności, 19
  - zastosowanie algorytmu SHA1, 54
  - zastosowanie do archiwów systemu Subversion, 309
  - zbiór poleceń instalatorskich, 51
- git add, 53, 62
  - rozwiązanie konfliktu łączenia, 134
- git add git/, 296
- git add -i, 344
- git add -p, 336
- git add -p main.c, 339
- git am, 267
  - aplikowanie łań, 246
  - wymiana łań, 255
- git am -3, 271, 272
- git am --abort, 270
- git apply, 267
- git archive, 290
- Git Bash, 29
- git bisect, 93
- git bisect replay, 96
- git bisect reset, 97
- git bisect visualize, 96
- git blame, 97
- git branch, 103, 112, 225
- git cat-file, 141
  - p, 53, 58
  - t, 346
- git checkout, 106, 111, 167, 223
  - b, 112, 224, 225
  - m, 143
  - my\_branch, 152
  - ours, 141
  - ref-format, 101



- theirs, 141
- track, 223
- git cherry-pick, 164
- git clean, 361
  - x, 361
- git clone, 39, 194, 195, 214
  - bare, 195
- git commit, 32, 34, 36, 38, 57, 64, 122
  - zapisywanie komunikatów dziennika zatwierdzeń, 65
- git commit
  - a, 64
  - all, 64
  - amend, 168, 169
  - amend --author, 170
  - no-verify, 279
- git config, 35, 40
  - parametry, 354
- git config --global, 40
- git config -l, 40, 222
- git config --unset, 41
- Git Cygwin, 28
- git diff, 37, 60, 116, 117
  - a, 119
  - base, 138
  - cached zatwierdzenie, 117
  - cached, 60, 117, 119
  - color, 119
  - HEAD, 137
  - konflikty w trakcie łączenia, 133
  - M, 119
  - MERGE\_HEAD, 137
  - ours, 138
  - postaci polecenia, 116
  - przedziały zatwierdzeń, 122
  - przykład polecenia, 120
  - S, 126
  - staged, 117
  - stat, 85, 119
  - theirs, 138
  - theirs, 141
  - tree, 361
  - w, 119
  - z konfliktami, 137
  - z ograniczeniem ścieżki, 125
  - zatwierdzenie, 117
  - zatwierdzenie1 zatwierdzenie2, 117
  - źródła obiektów drzew, 116
- git fetch, 81, 196
  - refspec, 201
- git filter-branch, 303
  - podział archiwum, 354
  - ponowne zapisywanie historii, 323
  - pułapki, 330
  - redagowanie komunikatu zatwierdzenia, 328
  - użycie, 323
  - wymazanie pliku, 325
  - zastosowania, 325
- git filter-branch --msg-filter, 329, 357
- git filter-branch --subdirectory-filter, 355
- git filter-branch --tag-name-filter cat, 355
- git format-patch, 255, 257, 267
  - aplikowanie łat, 246
  - folder poczty, 265
  - git diff, 257
  - parametr z jednym zatwierdzeniem, 262
  - z przedziałem zatwierdzeń, 258
- git format-patch -n, 258
- git format-patch -o, 267
- git format-patch --pretty, 266
- git format-patch --root, 262
- git fsck, 114, 345
- git fsck --no-reflog, 347
- git gc, 114, 353
- git gc --auto, 283
- git grep -i, 363
- git grep -l, 363
- git grep -untracked, 363
- git hash-object, 63
- git help --all, 32
- git help hooks, 280
- git help submodule, 295
- git init, 33
- git log, 36, 83, 225, 257
  - porównanie z git diff, 123
  - z konfliktami, 139
  - git log --left-right, 140
  - git log --merge, 140
  - git log -p, 140
- git log dev@, 192
- git log 'dev@', 192
- git log --first-parent, 321
- git log --follow, 68, 364
- git log --follow --name-only, 364
- git log --graph, 131
- git log HEAD, 83
- git log -S, 98
- git log --stat, 85

- git ls-files, 60
  - pliki konfliktowe, 136
  - wpisy w indeksie, 141
  - git ls-files -s, 141
  - git ls-files -u, 141
- git ls-files -s, 54
- git ls-remote, 196, 201
- git merge, 131, 210, 212
- git merge some\_branch, 152
- git merge-base, 89, 102, 145
- git mv, 38, 67
- Git native protocol, 199
- git pull, 150, 196, 201, 210, 246
- git pull --rebase, 185, 186, 359
- git pull -s ours, 290
- git pull -s subtree, 291
- git pull -s subtree, 289
- git push, 196, 201, 202, 206, 227, 246
  - odgłoszenia zdalne, 226
- git push ../svn, 319
- git push -f, 217
- git push gałąź\_zdalna, 370
- git push --mirror origin, 236
- git push origin, 202
- git push -u origin master, 236, 370
- git rebase, 170, 210, 212
  - a łączenie, 176
  - konflikt łączenia, 172
  - odniesiony do łączenia, 178
- git rebase --abort, 172
- git rebase --continue, 172
- git rebase -i, 172
- git rebase --interactive, 174
- git rebase --onto, 171, 241
- git rebase --preserve-merges, 180
- git rebase --skip, 172
- git reflog, 114, 190
- git reflog delete, 192
- git reflog expire, 192, 354
- git remote, 196, 204
  - zdalne konfigurowanie, 220
- git remote add -f, 206
- git remote add origin, 221
- git remote add origin adresarchithuba, 236
- git remote add upstreamrepo, 225
- git remote add url, 370
- git remote prune, 221
- git remote rename, 221
- git remote rm, 221
- git remote show origin, 221
- git remote show pilot, 224
- git remote update, 205, 206, 221
- git remote update --prune, 221
- git rerere, 366
- git reset, 158, 167
  - użycie, 158
  - zastosowanie opcji, 158
- git reset --hard, 158, 160, 167
- git reset --hard HEAD, 143
- git reset --hard ORIG\_HEAD, 143, 218
- git reset HEAD, 159
- git reset HEAD@{1}, 191
- git reset --mixed, 158
- git reset --soft, 158, 160, 161, 164
- git revert, 166, 167, 168
- git rev-list, 330
  - odzyskiwanie starej wersji pliku, 333
  - wyciąganie według daty, 331
  - zastosowania, 331
- git rev-list --no-merges -v od..do, 262
- git rev-parse, 80, 83, 191, 305
- git rm, 38, 65
  - pliki konfliktowe, 142
- git rm --cached, 66
- git rm -f, 67
- git rm nazwa pliku, 67
- git send-email, 255, 264
- git show, 37, 85
- git show --pretty, 57
- git show-branch, 88, 104
- git show-graph, 41
- git show-ref, 201
- git stash, 181, 188, 351
- git stash --all, 185
- git stash apply, 183
- git stash branch, 186
- git stash drop, 183
- git stash --include-untracked, 185
- git stash list, 183
- git stash -p, 185
- git stash --patch, 185
- git stash pop, 182
  - katalog roboczy, 183
- git stash save, 181, 182
- git stash show, 184
- git stash show -p, 184
- git stash show --stat, 184
- git status, 34, 35, 60, 225
  - pliki konfliktowe, 136
- git submodule, 295, 297

- git submodule add, 298, 301
- git submodule foreach, 302
- git submodule init, 298, 302
- git submodule status, 302
- git submodule summary, 302
- git submodule update, 112, 298, 302
- git svn, 309, 315
- git svn clone, 310
- git svn clone -r, 316
- git svn create-ignore, 322
- git svn dcommit, 312, 316, 320
- git svn dcommit -n, 321
- git svn fetch, 313
- git svn rebase, 314
- git svn show-ignore, 322
- git svn --stdlayout, 318
- git symbolic-ref, 81
- git tag, 57
- git update-index, 142
- git update-ref, 364
- git update-ref -d, 364
- git —version, 25
- git whatchanged --since, 360
- git write-tree, 53
- git.i386, 24
- git/ref, 80
- git/refs/heads/ref, 80
- git/refs/ref, 80
- git/refs/remotes/ref, 80
- git/refs/remotes/ref/HEAD, 80
- git/refs/tags/ref, 80
- GIT\_AUTHOR\_EMAIL, 35
- GIT\_AUTHOR\_NAME, 35
- GIT\_EDITOR, 34
- git-all.i386, 24
- git-arch, 23
- git-arch.i386, 24
- git-core, 23
- git-cvs, 23
- git-cvs.i386, 25
- git-daemon, 199, 231, 232
  - posadowienie na serwerze, 232
- git-daemon -interpolated-path, 235
- git-daemon.i386, 25
- git-daemon-run, 24
- git-daemon --export-all, 232
- git-debuginfo.i386, 25
- git-doc, 23
- git-email, 24
- git-email.i386, 25
- git-grep, 361
- git-gui, 23
- git-gui.i386, 25
- git-http-backend, 234
- GitHub, 235, 367
  - Advanced Search, 382
  - archiwum kodu powszechnie dostępnego, 367
  - czerpanie z otwartych źródeł, 392
  - dodanie pilota, 236
  - edytor kodu, 386
  - Explore, 382
  - graf powiązań, 375
  - hiperłącza podmodułów, 308
  - hiperodsyłacz, 380
  - historia zatwierdzeń, 371
  - inicjatywa gospodarcza, 395
  - interfejs do archiwów, 367
  - interfejs programowania aplikacji, 390
  - interfejs REST API, 390
  - kanal informacyjny, 373
  - kodowanie społeczne na źródłach otwartych, 372
  - kodowanie społeczne na źródłach zamkniętych, 391
  - komentarze na poziomie wierszy, 372
  - lista użytkowników, 236
  - metki, 389
  - modele kodowania, 393
  - most do systemu Subversion, 388
  - nazwa użytkownika, 369
  - obserwatorzy, 373
  - odnajdywanie
    - kodu, 382
    - projektów, 382
    - użytkowników, 382
  - oglądanie stanowiska, 236
  - organizacje, 390
    - zespoły, 390
  - osobista kopia kodu, 374
  - otwarty standard pełnomocnictw, 391
  - podanie nazwy archiwum, 235
  - podsumowanie, 396
  - powiadomienia, 379
  - przełącznik upubliczniania i prywatyzacji
    - repo, 392
  - przeszukiwanie witryny, 383
  - przycisk powiadamiania, 379
  - rozdzielnia, 235
  - rozwidlanie projektów, 254

## GitHub

- rozwidlenia, 372, 374
- selektor organizacji, 390
- strona eksploracyjna, 382
- strona poszukiwawcza, 382
- strony, 384
- SVN, 388
- Tags, 389
- tworzenie archiwum, 235, 369
  - dodanie pilota, 370
  - informacje o nowym archiwum, 369
  - wypchnięcie lokalnej treści, 370
  - zaaranżowanie README, 370
- upublicznianie archiwów, 235
- wikisy, 383
  - redagowanie, 385
- witryna, 368
- wybór poziomu kontrolowanego dostępu, 236
- wypychanie treści, 236
- zainicjowanie archiwum, 236
- założenie konta, 367
  - wybór rodzaju, 368
- zamówienia ciągnięcia, 372
  - obsługiwanie, 377
  - przygotowywanie, 376
  - zatwierdzenia, 379
- GitHub Enterprise, 395
- Github.com, 252
- gitk, 23, 88, 355
  - oglądanie łączenia, 89
  - przeglądanie grafu zatwierdzeń, 88
- gitk.i386, 25
- gitlink, 295
  - konfliktowy, 299
  - odsyłanie do obiektów, 297
  - uaktualnianie, 299
- Gitolite, 231
- git-submodule.sh, 297
- git-svn, 23, 27, 309, 314
  - rekonstruowanie pamięci podręcznej, 322
- git-svn.i386, 25
- git-svn-id, 312, 321, 322
- git-top-check, 359
- gituser, 237
- gitweb, 23, 150
- Global Information Tracker, 22
- globalny tropiciel informacji, 22
- głowa, 102
- GNU Interactive Tools, 24
- GnuPG, 57

- GoogleCode, 372
- górne archiwum, 196, 197
- Gradle Multiproject Builds, 302
- graf, 86
  - osiągalność węzłów, 90
- graf zatwierdzeń, 85, 86
  - bez strzałek, 88
  - DAG, 86
  - etykietowany, 87
  - git commit --amend, 170
  - osiągalność zwtwierdzeń, 90
  - przeglądanie, 131
  - użycie gitk do przeglądania, 88
- grep, 279
- gromadzenie zmian, 59
- GUI, 307

## H

- hard, 158, 160
- hash code, 46
- haszowania, 52
- haszowanie SHA1, 46
  - cechy, 54
  - git hash-object, 63
  - jednoznaczność, 52
  - streszczenie, 54
- haszowy ID, 79
- head, 102
- HEAD, 67, 79, 80, 137
- head of the last branch fetched, 81
- HEAD@[2], 163
- HEAD^, 160, 162
- hierarchia plików konfiguracyjnych, 39
- hierarchiczne nazwy odgałęzień, 101
- hierarchie drzewiaste, 55
- historia archiwum, 216
  - usunięcie pliku, 325
- historia głów odgałęzień, 192
- historia odgałęzień, 83
- historia odniesień, 192
- historia podprojektu, 293
- historia zatwierdzeń, 83, 105
  - DAG, 86
  - git cherry-pick, 164
  - git log, 83
  - git reset, 158
  - git revert, 166
  - grafy zatwierdzeń, 85
  - liniowa, 90

- łączenie, 129
  - na GitHubie, 371
  - nieupubliczniona, 157
  - notacja od..do, 84
  - określenie przedziału zatwierdzeń, 84
  - ponowienie historii, 159
  - przedziały zatwierdzeń, 89
  - przeglądanie starych zatwierdzeń, 83
  - rozdzielenie, 89
  - udoskonalanie, 157
  - upubliczniona, 157
  - w warunkach rozproszenia, 242
  - zasady zmieniania, 156
  - zgniatane komentarze, 153
- historia zmian refów, 163
- home directory, 26
- hook, 228, 275
- hooks/post-update, 233
- hosting services, 236
- HTTP, 199
- HTTPS, 199
- hunk staging, 339
- hunks, 336

## I

- i, 174
- iconified, 380
- ID obiektu, 46
- identyfikatory globalnie jednoznaczne, 46
- identyfikator haszowy, 79, 104
  - SHA1, 46, 80
- identyfikator SVN URI, 316
- identyfikator zatwierdzenia, 79
  - git svn, 312
  - git svn dcommit, 322
  - jednoznaczny globalnie, 79
  - w archiwum SVN, 316
- identyfikowanie użytkownika, 311
- identyfikowanie zatwierdzeń, 79, 80
  - nazwy bezwzględne, 79
  - nazwy względne, 81
  - symrefy, 80
- ignore-all-space, 119
- ignored, 60
- ikonizowanie, 380
- implementacja
  - bystra, 199
  - tępa, 199
- importowanie

- biblioteki, 287
- nowego modułu, 288
- podprojektów
  - poleceniem git pull -s subtree, 289
  - przez kopiowanie, 289
- include-untracked, 185
- indeks, 44, 45, 59
  - mechanizm, 48
  - migawkowe ujęcie, 77
  - operacje łączenia plików, 45
  - przechowanie bieżącego stanu, 182
  - stan, 60
  - śledzenie konfliktów, 140
  - uaktualnianie, 53
  - zmiany do wykonania, 45
- index-filter, 324
- index-filter, 352
- inetd, 232
- initial commit, 89
- inlining, 266
- INSTALL, 26
- instalowanie doczepek, 277
- instalowanie Gita, 23
  - biblioteki, 26
  - binarne dystrybucje Linuksa, 23
  - budowanie, 26
  - dystrybucja Debian, 23, 26
  - dystrybucja Fedor, 24
  - dystrybucja Ubuntu, 23
  - dystrybucje binarne, 24
  - katalog macierzysty, 26
  - msysGit, 29
  - pakiety, 23
  - strony HTML, 27
  - strony podręcznika, 27
  - systemy Gentoo, 24
  - uzyskanie wydania źródłowego, 25
  - w katalogu innym niż macierzysty, 27
  - w systemie Cygwin, 28
  - w systemie Windows, 27
    - msysGit, 29
    - wybór wersji, 27
  - wolnostojącej wersji Gita, 29
- interactive, 63, 174
- interakcyjne narzędzia GNU, 24
- interaktywne wystawianie kawałków, 335
- interfejs programisty aplikacji, 307
- interfejs REST API, 390
- interfejs użytkownika, 390
- interpolated-path, 235

## J

JavaScript Object Notation, 391  
jądro systemu Linux, 239  
JBoss Server, 394  
jednolity lokalizator zasobów, 198  
jednowierszowe podsumowania, 311  
Jekyll, 384  
JSON, 391  
już aktualne, 146

## K

K Desktop Environment, 286  
kanał informacyjny, 373  
    katalog  
    .dotest, 268  
    .git, 33, 51  
    .git/hooks, 275  
    .git/logs, 192  
    .git/rebase-apply, 268  
    .git/refs/, 80  
    .git/svn, 322  
    .gitobjects, 51  
    .rr-cache, 366  
    .svn, 311  
    /tmp/Depot, 203  
    contrib, 278  
    czysty, 35  
    nienadzorowany, 361  
    odzyskiwanie, 286  
    skład, 202  
    templates, 278  
katalog roboczy, 33  
    przechowanie bieżącego stanu, 182  
    usuwanie plików, 65  
    zabrudzony, 130  
kawalki, 336, 337  
    podzielenie, 339  
KDE, 286  
kęsy, 336  
kilof, 98, 126  
    przeszukiwanie zatwierdzeń, 362  
klasyfikacje plików, 60  
klon, 193  
    początek, 249  
    Subversion, 388  
klonowanie, 194  
    adres tylko do czytania, 304  
    kodu źródłowego SVN z odgałęzieniami,  
    317

    podmodułów, 305  
    podprojektu, 293  
    reflog, 189  
klonowanie archiwum, 39, 44, 195, 197, 252  
    .gitignore, 70  
    doczepki, 275, 277  
    zdalnego, 214  
kładzenie pliku do indeksu, 62  
kod haszowania, 46  
kodowanie społeczne, 367  
    czerpanie z otwartych źródeł, 392  
    edytor kodu, 386  
    kanał informacyjny, 373  
    komentarze, 377  
    komentarze na poziomie wierszy, 372  
    konwersacja, 377  
    kuźnia, 372  
    mechanizm powiadomień, 379  
    modele kodowania, 393  
    na otwartych źródłach, 372  
    obserwowanie, 372, 373  
    oparte na źródłach zamkniętych, 391  
    organizacje, 390  
    redagowanie kodu, 387  
    rozjemca, 394  
    rozwidlenia, 372, 374  
    wyszukiwanie, 382  
    zamówienia ciągnięcia, 372, 376, 377  
kolizja SHA1, 52  
kombinowany dif, 143  
komentarze, 135  
    do zamówienia ciągnięcia, 378  
    kodowanie społeczne, 377  
    wierszy, 378  
komunikat dziennika, 34  
komunikaty zatwierdzeń  
    korekta, 328  
koncepcja chronionego rdzenia, 374  
koncepcje archiwum, 194  
koncepcje Gita, 51  
konfigurowanie synonimu polecenia, 41  
konfigurowanie zdalne, 219  
    git config, 221  
    git remote, 220  
    obróbka ręczna, 222  
konflikt łączenia, 132  
    alternatywne historie archiwum, 218  
    badanie, 136  
    git diff, 133, 138  
    git log, 139

- git rebase, 172
- lokalizowanie konfliktowych plików, 136
- postępowanie, 135
- śledzenie, 140
- zakończenie rozwiązywania, 142
- zatwierdzenia SVN, 316
- konfliktowe pliki, 136
- konsument w dole, 247
- konsument w górze, 246
- konta GitHuba, 369
- kontekst
  - odzyskanie ze skrytki, 183
  - stos, 183
- kontrolowanie wersji, 17
- konwencja wykrywania awarii, 275
- konwencje typograficzne, 13
- konwersja z SVN do Gita, 356
  - ogólne zalecenia, 356
  - usuwanie identyfikatorów zatwierdzeń, 357
  - usuwanie trzonu, 356
- krok łączenia, 212
- krok pobierania, 211
- krok przebazowania, 212
- kuźnia, 372
- kwalifikatory refów, 191

## L

- l, 222
- LDAP, 395
- left-right, 140, 314
- lekkie metki, 57
- LF, 29
- libcurl4-openssl-dev, 26
- libexpat1-dev, 26
- lightweight, 57
- line of development, 12
- linia rozwojowa, 12
- Linux Kernel, 240
- Linux Kernel Mailing List, 22
- Linux®, 15
- list of topically focused commits, 376
- lista kontroli dostępu, 283
- listel, 255
  - wysyłanie, 266
- local, 230
- local-tracking branch, 194
- Location, 234
- login, 311

- lokalizator URL
  - Gita, 198
  - GitHuba, 370
- lokalizowanie konfliktowych plików, 136
- ls, 107
- ls -lsa, 39
- luka semantyczna, 78

## Ł

- łamanie, 266
  - wierszy, 266, 273
- łataj i stosuj, 255
- łatanie, 246
- łaty, 127, 155, 255
  - a łączenie, 273
  - argument z jednym zatwierdzeniem, 262
  - doczepki, 281
  - dystrybucja zmian do przeglądu, 256
  - generowanie, 257
  - git am, 267
  - git apply, 267
  - git format-patch, 257, 267
  - git send-email, 264
  - inlining, 266
  - catalog, 267
  - catalog .git/rebase-apply, 268
  - kontekst, 270
  - łamanie wierszy, 266
  - łata-listel-przegląd-zastosowanie, 256
  - pocztowe ekspediowanie, 264
    - nagłówki, 264, 266
  - postać otwarta, 266
  - powody stosowania, 256
  - przedział zatwierdzeń, 258
  - przesyłka, 257
  - ręczne poprawienie, 343
  - sortowanie topologiczne, 264
  - stosowanie, 267
  - wady, 273
  - wybijanie zatwierdzeń, 256
  - wymiana, 255
  - wystawianie kawałków kodu, 335
  - zapora sieciowa, 256
  - zatwierdzenie korzeniowe, 262
  - złe, 273
- łączenia, 129
  - a operacja rebase, 176
  - a przebazowanie, 180
  - asymetryczność, 132

- łączenia
  - automatyzacja, 151
  - badanie konfliktów, 136
  - baza, 145
  - dcommit, 320
  - dwóch odgałęzień, 130
  - gałęzi zdalnego i lokalnego nadzorowania, 212, 213
  - indeks, 130
  - katalog roboczy, 130
  - konfliktowe, 143
  - konflikty, 132
  - krzyżowe, 145, 148
  - liniowej historii, 153
  - lokalizowanie konfliktowych plików, 136
  - łatanie, 273
  - nasze, 149
  - niezbieżne, 147
  - operacje, 153
  - ośmiornicowate, 148, 212
  - poddrzewo, 149
  - postępowanie z konfliktami, 135
  - przygotowanie, 130
  - przykłady, 129
  - rekurencyjne, 148
  - rozpatrywanie przez Git, 151
  - siatka sytuacyjna, 144
  - specjalne, 149
  - stan wynikowy, 147
  - sterowniki, 151
  - strategie, 144
    - stosowanie, 149
  - szybkie przekazanie, 212
  - trzytorowe, 147
  - wznowienie, 143
  - zakończenie rozwiązywania konfliktu, 142
  - zaniechanie, 143
  - zatwierdzanie, 320
  - zdegenerowane, 146
  - zgniatane, 152
  - zwykle, 147
- łączenie nieużytków, 352
  - automatyczne, 353
  - ręczne, 353
- łączenie odgałęzień, 102
- łączenie trzytorowe, 270

## M

- m, 109
- M, 119

- magazyn obiektów, 44, 46
  - bloby, 44, 47, 49
  - drzewa, 44, 49, 53
  - dzielenie przez wiele archiwów, 230
  - metki, 45, 49, 57, 100
  - obrazy, 49
  - zatwierdzenia, 45, 49, 56, 77, 155
- mail transfer agents, 266
- mail user agent, 264
- maintainer, 238
- Makefile, 26, 365
- manifest Gita, 47
- manpages, 27
- Markdown, 383
- master, 37, 49, 81, 100, 101
- master^, 81
- Maven Multimodule Project, 302
- mbox, 265
- mechanizm atrybutów, 151
- mechanizm indeksu, 48
- mechanizm przetwarzania gałęzi, 323
- Mercurial, 21
- merge, 129
  - merge, 140
- merge commit, 88
- merge driver, 151
- MERGE\_HEAD, 81, 137
- mesh, 239
- metki, 44, 45, 49, 57, 100
  - filtry, 324
  - jako kompresowane pliki archiwalne, 389
  - klonowanie archiwum, 195
  - lekkie, 57
  - skomentowana, 57
- migawka, 77, 78
- migawkowe ujęcie, 77
- mixed, 158
- mode, 360
- model obiektowy Gita, 71
  - łączenia, 152
- model obróbki rozproszonej, 18
- modele kodowania, 393
  - o źródłach otwartych częściowo, 394
  - porucznika i dowódcy, 393
  - scentralizowany, 393
- moniker, 21
- Monotone, 21
- more=num, 105
- most Git-SVN, 389
- msg-filter, 324



--msg-filter, 329  
msgfmt, 26  
msysGit, 27  
  instalowanie, 29  
MTA, 266  
  łamanie wierszy, 273  
MUA, 264  
  łamanie wierszy, 266, 273  
mutt, 265

## N

-n, 258  
nadzorowanie  
  danych z archiwów, 194  
  gałęzi, 211  
  przemianowań plików, 69  
nagłówki poczty elektornicznej, 257  
--name-only, 364  
NAS, 243  
nasze, 149  
native, 24  
nazwy  
  adresowane treścią, 46  
  archiwów zdalnych, 198  
  metek, 57, 80, 100  
  symboliczne, 190  
  ścieżek a treść, 47  
nazwy odgałęzień, 100  
  hierarchiczne, 101  
  reguły, 101  
  sporządzanie wykazów, 104  
  usunięcie, 114  
nazwy zatwierdzeń  
  absolutne, 79  
  bezwzględne, 79  
  identyfikator haszowy, 104  
  względne, 81  
Network Attached Storage, 243  
Network File System, 198  
New Repository, 369  
news feed, 373  
NFS, 198  
nienadzorowane katalogi, 361  
nieopublikowana historia, 157  
niepodzielne zbiory zmian, 78  
--no-ff, 320, 321  
nonbare, 194  
non-fast-forward push problem, 217  
notacja od..do, 84  
notification page, 379

## O

-o, 115, 267  
obchód refów, 345  
obiekty  
  bez odniesień, 352  
  nieosiągalne, 345  
  wiszące, 345, 347, 352  
obiekty Gita, 44, 49, 52  
  bloby, 44, 47, 49  
  drzewa, 44, 49, 53  
  haszowanie treści, 46  
  metki, 45, 49, 57, 100  
  po drugim zatwierdzeniu, 50  
  zatwierdzenia, 45, 49, 56, 77, 155  
object store, 44  
obrazy magazynu obiektów, 49  
obserwatorzy, 373  
obsługa  
  dat, 332  
  kolejki zamówień ciągnięcia, 254  
Octopress, 384  
  strona główna witryny, 386  
octopus, 148, 149  
od..do, 84  
odgałęzienia, 49, 99  
  a metka, 100  
  aktywne, 101  
  bieżące, 194  
  czas istnienia, 103  
  czubek, 102  
  dodawanie i usuwanie w zdalnym  
   archiwum, 226  
  gałąź lokalnego nadzorowania, 197  
  gałąź zdalnego nadzorowania, 195, 197, 205  
  głowa, 102  
  HEAD, 111  
  łączenie, 102, 129  
  łączenie dwóch odgałęzień, 130  
  łączenie więcej niż dwóch odgałęzień, 148  
  łączenie zmian w nowe odgałęzienie, 109  
  master, 49  
  nadzorowania, 222  
   tworzenie, 222  
   względne porównania, 225  
  nadzorujące, 171, 197  
  nazwy, 100  
  nienadzorujące, 197  
  odzyskiwanie, 114  
  organizacja treści archiwum, 102

- odgałęzienia
  - pochodzące z dwu archiwów, 357
  - powody stosowania, 99
  - powrót do stanu pierwotnego, 270
  - przeglądanie, 104
  - przesłanie w przód, 171
  - przygotowanie do łączenia, 130
  - publikowanie, 102
  - rozwojowe, 100
  - równoprawność, 105
  - stan skrytki, 186
  - tematyczne, 100, 197
  - tworzenie, 103
  - usuwanie, 112
  - wyciąganie, 106
  - wykazy nazw, 104
  - z dynamiczną historią, 241
  - zastosowanie, 101
  - zdalne, 197, 226
- odniesienia
  - rejestr odniesień, 189
  - rozpoznawanie, 191
  - specyfikator, 200
- odniesienia do zatwierdzeń, 79
  - domyślne, 79
  - jawne, 79
- odniesienia symboliczne, 80
  - FETCH\_HEAD, 81
  - HEAD, 80
  - MERGE\_HEAD, 81
  - ORIG\_HEAD, 81
- odsyłacze gitlinks, 295
- odśmiecianie, 192, 348, 353, 355
  - obiektów, 353
  - parametry konfigurowania, 354
  - stare refy, 325
- odwołania do archiwów zdalnych, 198
- odzyskiwanie z przebazowania w górze, 358
- ograniczanie ścieżki, 333
  - git diff, 125
- ograniczenia widoczności, 307
- ograniczenie przeszukiwania, 333
- określenie
  - przedziału zatwierdzeń, 84
  - typu obiektu, 346
- onto, 171
- open source, 367
- operacja zdjęcia, 183
- operacje łączenia, 45
- operator różnicy symetrycznej, 314

- oprogramowanie o otwartym kodzie
  - źródłowym, 367
- organizacje, 390
- ORIG\_HEAD, 81, 143, 218
- origin, 196, 208
- origin, 196
- origin/master, 171
- osiągalność, 90
- ośmiornica, 148
- otwarty kod źródłowy, 367
- otwieranie źródeł, 392
- ours, 149
- ours, 141

## P

- p, 84, 140, 184, 185, 336
- pack files, 45
- page scrapping, 390
- pamięć podłączona do sieci, 243
- parent commits, 81
- parent-filter, 324
- partially checkouts, 286
- partnerskie składowanie, 243
- patch, 185
- patch and apply, 255
- patching, 246
- pełnomocnictwa w podmodułach, 305
- pickaxe, 98
- pielegnator, 238, 244, 247
  - a budowniczy, 245
- pierwszeństwa pomijanych plików, 70
- piloty, 193, 196, 198
  - origin, 196
- plama-wyczyść, 283
- pliki
  - .git/config, 39, 196
  - .git/FETCH\_HEAD, 81
  - .git/info/exclude, 71
  - .gitignore, 61, 69, 321, 370
  - .gitmodules, 298
  - ~/ .gitconfig, 39
  - archiwalne, 389
  - dodawanie, 38
  - etc/gitconfig, 40
  - expat.h, 26
  - haszowanie SHA1, 46
  - historia z uwzględnieniem przemieszczeń, 364
  - i drzewa, 53

- importowanie, 289
- INSTALL, 26
- klasyfikacje plików w Gicie, 60
- kładzenie pliku do indeksu, 62
- konfiguracyjne, 39
  - hierarchia, 39
  - konfigurowanie synonimu, 41
- konfliktowe
  - czyszczenie statusu, 142
  - git diff, 139
  - lokalizowanie, 136
  - zatwierdzenia w obu częściach historii, 140
- łączenie trzytorowe, 270
- Makefile, 26
- nadzorowane, 60
- nienadzorowane, 60
- niepołączone, 136
- odtwarzanie starych wersji, 67
- operacja odwrócenia, 168
- operacja resetowania, 167
- ozdyskiwanie starej wersji, 333
- pakowane, 45, 48
- pierwotny, 37
- pierwszeństwo pomijania, 70
- pomijane, 60
- pomijanie w katalogu, 61
- postępowanie za przemieszczonymi plikami, 364
- przechowanie podręczne pliku, 62
- przemianowanie, 38, 67
- przeniesienie funkcji, 78
- przeszukiwanie treści, 362
- README, 370
- reflog, 163
- szczegółowy przegląd, 71
- śledzone, 60
- usuwanie, 65
  - zatwierdzonych plików, 67
  - zbędnych plików edytora, 352
- wydobywanie wersji, 334
- wymazanie, 325
- wystawianie, 62
- wystawianie zmian, 64
- zachowywanie bez nadzorowania, 365

plik-wypełniacz, 370

plug-in, 306

plumbing, 51

płytki klon jednego odgałęzienia, 309

pobieranie, 194

pobranie, 211, 217

pociągnij, 150

początek klonu, 249

pocztowe ekspediowanie łąć, 264

poddrzewa, 149

podkatalog .git/logs/refs/, 192

podkatalog .git/logs/refs/heads, 192

podkawalki, 340

podmoduły, 285, 301

- bezpośrednie aktualizowanie kodu, 304
- duże repo, 307
- git archive, 290
- git pull -s ours, 290
- git pull -s subtree, 291
- git pull -s subtree, 289
- git submodule, 295
- git submodule add, 298, 301
- git submodule foreach, 302
- git submodule init, 298, 302
- git submodule status, 302
- git submodule summary, 302
- git submodule update, 298, 302
- hiperłącza w archiwach GitHuba, 308
- ograniczenia widoczności, 307
- pobranie treści, 302
- polecenia podmodułowe, 301
- przeistaczanie podfolderu, 303
- przygotowywanie, 303
- przypadki użycia, 306
- refy zatwierdzeń, 302
- sprawdzanie haszowań zatwierdzeń, 305
- stany zabrudzenia, 302
- struktura katalogowa podkomponentu, 303
- udostępnienie przykładów kodu z książki, 306
- w Gicie, 295
- wielopoziomowe zagnieżdżanie archiwów, 307
- włączki, 306
- wyciąganie, 294
- wykorzystanie pełnomocnictw, 305
- wyświetlanie łąć zmian, 302
- zalety używania, 302
- zaplecze Gita, 285
- zarejestrowanie nowego, 301
- zmniejszanie rozmiaru archiwum, 307

podpolecenia, 31

- uzyskiwanie dokumentacji, 32

podprojekty

- git submodule, 295
- importowanie poleceniem git pull -s subtree, 289

- podprojekty
  - importowanie przez kopiowanie, 289
  - kierowanie zmian w górę, 293
  - klonowanie, 293
  - wyciąganie przy użyciu skryptów, 293
- podział archiwum, 354
- pojęcia podstawowe, 43
- polecenia
  - podmodułowe, 301
  - powłokowe, 323
  - tworzenie własnych, 359
- ponowne użycie zarejestrowanego rozwiązań, 366
- pop, 183
- poprawianie, 168
- poprawka, 169
- porównywanie
  - gałęzi, 358
  - git diff, 117
  - wersji plików, 118
- portier, 317
- porządek topologiczny łąty, 264
- post hook, 275
- post-applypatch, 281
- post-checkout, 276, 283
- post-commit, 281
- postępowanie za przemieszczonymi plikami, 364
- post-merge, 283
- post-receive, 283
- post-update, 233, 276, 283
- potok filtrujący
  - wykonanie zatwierdzenia, 324
- PowerPC®, 15
- powiadomienia, 379
  - hiperodsylacz, 380
  - opcje ogólnosystemowe, 380
  - powiązane tematycznie, 380
- powielanie archiwum, 44
- praca z wieloma archiwami, 247
- prace w toku, 182
- prawa zatwierdzania, 230
- pre hook, 275
- pre-applypatch, 281
- pre-auto-gc, 283
- pre-commit, 276, 278, 280
- prefix=svn, 318
- prepare-commit-msg, 281
- pre-rebase, 283
- pre-recv, 282
- preserve-merges, 180
- pretty=oneline, 96
- pretty=short, 84
- problem niespiesznego wypychania, 217
- problematyczne pliki, 136
- proces połowienia, 94
- producent w dole, 247
- producent w górze, 247
- projekt bezpieczeństwa Gita, 322
- projekt nadrzędny, 301
- projekty
  - podmoduły, 285
  - rozwidlanie, 252, 374
  - zestawianie, 285
- protokół
  - HTTP, 199, 255
  - HTTPS, 199
  - przesyłania, 255
  - Rsync, 200
- prune-empty, 355
- przebazowanie, 154, 172
  - a git merge, 315
  - a łączenie, 180
  - ciągu zatwierdzeń, 176
  - gałęzi, 358
  - gałęzi lokalnego nadzorowania, 213
  - linearyzacja, 179
  - odgałęzienia, 176
    - na którym dokonano łączenia, 178
  - odzyskiwanie z przebazowania w górze, 358
  - przepisywanie zatwierdzenia, 180
  - reflog, 189
  - w górze, 359
  - z zabrudzonym katalogiem roboczym, 351
  - zatwierdzeń, 170
- przebieg na sucho, 361
- przechowanie podręczne pliku, 62
- przedział zatwierdzeń, 89, 261
  - analiza, 261
  - git cherry-pick, 166
  - git diff, 122
  - interpretacja, 91
  - łąty, 258
  - początek i koniec, 90
  - różnica symetryczna, 92
  - temat..master, 91
  - wykluczenie końca, 92
  - wykluczenie początku, 92
  - X..Y, 90

- przeгляд
  - modelu obiektowego, 71
  - partnerski, 256
  - plików, 71
- przeглядanie
  - kodu, 376
  - odgałęzień, 104
  - zatwierdzeń, 36, 83
- przeistaczanie podfolderu w podmoduł, 303
- przekształcenia w Gicie, 351
- przemianowywanie plików, 67
  - bezpośrednio, 38
  - pośrednio, 38
  - problemy ze śledzeniem, 69
  - śledzenie, 68
- przeróbka, 262
- przerywany tok pracy, 181
- przesłanie w przód, 171
  - gałęzi rozwojowej, 171
- przeźrenie nazw ref, 80
- przeźrenie
  - .NET, 302
  - robocza, 247
- przesyłka, 257
- przeszukiwanie, 94
  - archiwum, 361
  - git grep, 362
  - metoda połowienia, 94
- przygotowanie do łączenia, 130
- przygotowywanie podmodułów, 303
- przywracanie
  - reflog, 189
- publikowanie
  - oddzielenie od zatwierdzeń, 242
  - odgałęzień, 102
- publikowanie archiwum, 194, 230
  - Location, 234
  - na koncie GitHub, 235
  - wskazówki, 236
  - wydawca, 247
  - z anonimowym dostępem do czytania, 231
  - z anonimowym dostępem do pisania, 235
  - z kontrolowanym dostępem, 230
  - za pomocą demona HTTP, 233
  - za pomocą demonów HTTP, 235
  - za pomocą procesu git-daemon, 232
  - za pomocą Smart HTTP, 234
  - za pośrednictwem Gita, 235
- publishing a repository, 194
- pull request, 376

- Pull Request, 376
- push, 194
- push in, 297

## R

- r, 316, 318
- RCS, 17, 21
- rdzenny protokół Gita, 199
  - eksportowanie archiwów, 232
- realna historia, 156
- rebasing, 154
- receivepack, 235
- recurse-submodules, 305, 307
- recursive, 147, 150
- ref, 80
  - aktualizowanie, 363
  - do miejsca przeznaczenia, 200
  - do źródła, 200
  - kwalifikatory oparte na datach, 191
  - odmiany nazw, 81
  - odzyskiwanie, 114
  - przeźrenie nazw, 80
  - refs/stash, 182
  - symboliczny, 80
  - usuwanie, 363
  - wybór zatwierdzeń, 82
  - zatwierdzenia podmodułu, 305
  - źródłowy, 212
- reference, 230
- reflog, 181, 189
  - a skrytka, 192
  - archiwa zdalne, 195
  - czas przeterminowania, 355
  - gałęzi, 190
  - git filter-branch, 189
  - nazwa symboliczna, 190
  - obiekty nieosiągalne, 347
  - odśmiecianie, 192
  - odtworzenie utraconego zatwierdzenia, 345
  - operacje powodujące uaktualnienie, 189
  - refy, 190
  - uaktualnienie, 189
  - włączenie, 192
  - wyłączenie, 192
  - wymuszenie przeterminowania, 192
- refs/, 80
- refs/heads/, 195
- refs/heads/\*, 211
- refs/remotes/, 195

- refs/stash, 182
- refspec, 196, 200
  - git fetch, 201
  - git push, 201, 226
  - przepływ danych, 200
  - składnia, 200
  - znak gwiazdki, 200
  - znak plus, 200
- refspecem, 198
- reguły .gitignore, 70
- rejestr odniesień, 164, 181, 189
  - przechowywanie, 192
- rekonstruowanie
  - utraconego zatwierdzenia, 345
  - zatwierdzeń, 355
- rekord CNAME, 384
- rekurencyjna, 147
- relacja przodka, 86
- release, 238
- remote, 193
- remote origin, 202
- remote-tracking branches, 194
- reorganizacja archiwum, 249
- repository, 43
- repozytorium, 20
- rerere, 366
- rerere.enabled, 366
- resolve, 147, 149
- REST API, 390
- reuse recorded resolution, 366
- revision control system, 17, 21
- rm, 65
- root, 262, 264
- rozbiór daty, 332
- rozdzielnia, 235
- rozgąłęzianie, 252
- rozjemca, 394
- rozproszony model rozwojowy, 229, *Patrz też*
  - warunki rozproszenia
- rozproszony system kontroli wersji, 129
  - rozdzielność zatwierdzeń i publikowania, 242
- rozwiązanie konfliktu łączenia, 142
- rozwiąż, 147
- rozwidlanie, 252, 374
  - historii archiwum, 216
  - projektu, 253
    - w GitHubie, 254
  - usuwanie rozbieżności, 253
- rozwidlenia
  - ponowne włączenie, 254

- równoprawność odgałęzień, 105
- różnica symetryczna, 92
  - git diff, 124
- różnice, 115
  - git diff, 116
  - porównanie wprowadzania w systemach Subversion i Git, 126
  - wykaz, 115
- Rsync, 200

## S

- s, 141
- S, 98, 126
- s ours, 290
- s subtree, 292, 293
- save, 182
- SCCS, 20
- scentralizowane archiwum, 237
- SCM, 17
- Secure Hash Function, 19
- Secure Shell, 29
- sed, 357
- semigeneric, 237
- sendmail.smtpserver, 265
- sendmail.smtpserverport, 265
- sendmail, 265
- separatory słów, 192
- serwer, 229
  - bit grupowego bezpiecznego umocowania, 237
  - konta na wpół ogólnych użytkowników, 237
  - posadowienie git-deamon, 232
  - przełącznikowy SMTP, 265
- SHA1, 19, 46, 54
  - identyfikator haszowy, 80
  - ref, 80
- shared, 230
- show branch, 37
- siatka archiwów, 239
- since=, 360
- skarbiec, 20, 43
- skierowany graf acykliczny, 86
- skład, 202
- skomentowana metka, 57
- skrobanie stron, 390
- skrypt z doczepką, 277
- skrytka, 181
  - chowanie plików, 187
  - przechowanie, 182

- ref stash, 192
- rekonstruowanie, 188
- stos kontekstów, 183
- stos stanów, 182
- ukazywanie stanu, 184
- wpisy, 184
- wyjęcie stanu, 186
- smart, 199
- Smart HTTP, 234
- smudge/clean, 283
- snapshot, 77
- social coding, 367
- soft, 158, 160, 161, 164
- sortowanie topologiczne, 179
  - łaty, 264
- Source Code Control System, 20
- source code manager, 17
- SourceForge, 372
- specyfikator odniesienia, 196, 198, 200
  - fetch, 196
- sprawdzenie systemu plików, 345
- squash, 153
- squash commit, 152
- SSH, 29
- stage, 45, 59
- staged, 34
- staged, 117
- stan indeksu, 60
- stan wynikowy łączenia, 147
- stash, 181
- stat, 85, 119, 184
- stdin, 324
- stdin-do-stdout, 329
- stdlayout, 318
- stdout, 324
- sterowniki difów, 151
- sterowniki łączenia, 151
  - binarny, 151
  - suma, 151
  - tekstowy, 151
- sticky bit, 237
- strategie łączenia, 144, 290
  - już aktualne, 146
  - łączenia specjalne, 149
  - łączenia zdegenerowane, 146
  - łączenia zwykłe, 147
  - nasze, 149, 150
  - ośmiornica, 148
  - poddrzewo, 149, 150
  - rekurencyjna, 147
  - rozwiązań, 147, 149
  - sterowniki łączenia, 151
  - stosowanie, 149
  - szybko do przodu, 146
- streszczenie, 54
- strony GitHuba, 384
- struktura archiwum, 237
  - dzielonego, 237
  - przykłady, 239
  - rozproszonego, 238
  - wiele archiwów w górze, 250
- subdirectory-filter, 323, 325
- subdirectory-filter, 355
- submodule, 285
- subtree, 149
- Subversion, 309
- Subversion Bridge, 356
- superprojekt, 301
- svn:ignore, 321
- SVN, 21, 68, 309
  - a GitHub, 388
  - archiwum portiera, 317
  - gałęzie, 319
  - git svn, 316
  - git svn create-ignore, 322
  - git svn dcommit, 312, 320
  - git svn dcommit -n, 321
  - git svn fetch, 313
  - git svn rebase, 314
  - git svn show-ignore, 322
  - identyfikowanie użytkownika, 311
  - jedno archiwum Gita, 317
  - klonowanie archiwum, 309
  - klonowanie gałęzi, 317
  - komunikaty zatwierdzeń, 311
  - konwersja, 356
    - usuwanie identyfikatorów zatwierdzeń, 357
    - usuwanie trzonu po zaimportowaniu, 356
  - miejsce pochodzenia zatwierdzenia, 312
  - most do systemu, 388
  - nazwa użytkownika, 311
  - obiekty zatwierdzeń Gita, 316
  - obraz historii, 313
  - pliki pomijane, 321
  - plytki klon jednego odgałęzienia, 309
  - pobranie przed zatwierdzeniem, 313
  - ponowne włączanie archiwum, 320
  - praca z systemem, 321
  - przebazowanie łą, 315
  - przebazowanie zmian, 314

## SVN

- rekonstruowanie pamięci podręcznej, 322
- rozwidlenie historii, 313
- svn blame, 315
- utrzymywanie prostoty identyfikatorów zatwierdzeń, 316
- wykrywanie pobranej wersji, 322
- zatwierdzenia, 312, 314
- zgniecione zatwierdzenie łączenia, 316
- zmiana identyfikatorów zatwierdzeń, 312

svn mv, 69

sygnały błędów, 35

symbole uniwersalne, 101

symbolic reference, 80

symref, 80

synonimy poleceń, 41

system

- kontroli kodu źródłowego, 20
- kontroli wersji, 17
- kontroli wydań, 17
- nadzorowania treści, 46
- nadzorowania wydań, 21
- nazw domen, 384
- rozgałęziania, 99
- Subversion, 21, 309
- SVN, 309

szybki przegląd zmian, 360

szybkie przekazanie, 207, 212

szybko do przodu, 146

## Ś

### śledzenie

- konfliktów, 140
- przemianowań, 68
- treści, 46

## T

tag, 33

tag-name-filter, 323, 324

--tag-name-filter cat, 330, 355

tags, 44

--tags, 318

teams, 390

tekstowy sterownik łączenia, 151

termin nieodtworzalności zatwierdzeń, 355

text, 151

textconv, 335

--theirs, 141

three way merge, 270

tip, 102

topic, 197

topic branch, 100

tracked, 60

tracking, 211

tracking branch, 100

transakcje niepodzielne, 19

transplantacja linii rozwojowej, 171

tree-filter, 324, 326

--tree-filter, 352

trees, 44

trunk, 356

--trunk, 318

tryb połowienia, 94

trzon, 356

trzytorowe łączenie, 147

tworzenie

- archiwum początkowego, 33
- archiwum w GitHubie, 369
- archiwum wzorcowego, 203
- doczepki, 278
- gałęzi nadzorowania, 222
- i wyciąganie odgałęzień, 111
- nowych zatwierdzeń, 189
- odgałęzień, 103
- reflog, 189
- własnych poleceń, 359

tylda, 82

typy

- metek, 57
- obiektów Gita, 44

## U

-u, 141

uaktualnianie podmodułów, 294

uaktualnianie wika, 383

UI, 390

unified diff, 115

Unified Resource Locators, 198

Uniform Resource Identifiers, 198

Uniksowy system plików, 47

union, 151

UNIX®, 15

unmerged, 136

--unset, 41

untracked, 60

--untracked, 363



- update, 276, 282
- Updating origin, 205
- uporządkowanie lat, 264
- upstream, 195
- upstream, 225
- upstream consumer, 246
- upstream producer/publisher, 247
- upstream repository, 196
- upublicznianie archiwum, 194 *Patrz też*
  - publikowanie archiwum
- upubliczniona historia, 157
  - zmienianie, 241
- URI, 198
- URL, 198
  - lokalizatory, 198
  - upublicznienie archiwów, 235
- user interface, 390
- user-path, 199
- usługa inetd, 232
- ustalenie kontekstu, 327
- usuwanie
  - odgałęzień, 112
  - plików, 65
    - wymuszanie, 67
    - zatwierdzonych, 67
  - refów, 363
  - zatwierdzeń nieosiągalnych, 192
  - zbędnych plików edytora, 352
- utrzymywanie archiwum, 372
- użytkowanie Gita
  - dodawanie pliku do archiwum, 34
  - katalog roboczy, 33
  - kolejne zatwierdzenie, 36
  - konfigurowanie autora zatwierdzenia, 35
  - pliki konfiguracyjne, 39
  - przeglądanie różnic w zatwierdzeniach, 37
  - przeglądanie zatwierdzeń, 36
  - przemianowywanie plików archiwum, 38
  - tworzenie kopii archiwum, 39
  - tworzenie archiwum początkowego, 33
  - usuwanie plików archiwum, 38
  - wprowadzenie, 33

## V

- VCS, 17
- version, 32
- version control system, 17

## W

- w, 119
- wartość haszowania SHA1, 46, 47
- warunki rozproszenia, 241
  - brak historii prawdziwej, 242
  - przepływ w dół, 244
  - przepływ w górę, 244
  - rozdzielność kroków zatwierdzeń i publikowania, 242
  - rozpoczynanie archiwum, 248
  - zmienianie historii upublicznionej, 241
- watching, 373
- wciąganie na zabrudzone drzewo, 184
- wepchnięcie, 297
- wersja konfliktowa, 140
- węzły zatwierdzeń, 86
- whatchanged, 360
- whitespace characters, 101
- wielopoziomowe zagnieżdżanie archiwów, 307
- wielotorowość prac rozwojowych, 19
- wiersz kodu
  - komentarze, 378
- wikisy, 383
- wikiwpisy, 383
- wildcards, 101
- WIP, 182
- własna przestrzeń robocza, 247
- włączka, 306
- word wrapping, 266
- work in progress, 182
- wpół ogólny użytkownik, 237
- współbieżny system wersji, 21
- wyciąganie, 238
  - gałęzi, 161
    - nadzorowania, 223
  - nowych odgałęzień, 111
  - odgałęzienia wyosobnione HEAD, 111
  - odgałęzień, 106
    - łączenie zmian w nowe odgałęzienie, 109
    - w wypadku niezatwierdzonych zmian, 107
  - podmodułów, 297
  - podprojektów, 293
  - ścieżki, 167
  - według daty, 331
  - przestrogi, 332
- wyciągi częściowe, 286
  - gitlinki, 297
- wydawca, 247

wydobycie  
  podkatalogu w podmoduł, 304  
  struktury katalogowej, 303  
wykaz różnic, 184  
  streszczenia, 116  
  uniksowy dif, 115  
wykazy nazw odgałęzień, 104  
wymiana danych  
  łaty, 256  
wymuszanie  
  odpowiedzialności, 19  
  usunięcia plików, 67  
wymuszone wypchnięcie, 228  
wyoobnione HEAD, 94  
wyoobnione odgałęzienia HEAD, 111  
wyprowadzanie różnic  
  Git, 126  
  Subversion, 126  
wypychanie, 194  
  do archiwum rozwojowego, 228  
  doczepka, 282  
  gałęzi nadzorowania, 226  
  połączonej historii archiwum, 219  
  reflog, 189  
  wymuszone, 228  
  z archiwum, 227  
wystawianie, 34, 59, 62  
  git diff, 117  
  kawałków, 335  
  git add -p, 338  
  kompilowanie bazy kodu, 336  
  plików, 62  
  po kawałku, 339  
  zmian do wykonania, 45  
względne nazwy zatwierdzeń, 81  
wznowienie łączenia, 143  
wzorcowe archiwum, 202  
  dodawanie nowego wykonawcy, 207

## X

-x, 361  
X.org, 240  
xinetd, 232

## Y

yum, 24

## Z

zagnieżdżone archiwa, 301  
  wielopoziomowe, 307  
zagnieżdżone foldery źródłowe, 303  
zająkliwa nazwa, 366  
zamówienia ciągnięcia, 254, 376  
  automatyczne łączenie, 379  
  kolejka, 377  
  ogólnosystemowa, 378  
  komentarze, 378  
  kompletowanie ogłoszenia, 376  
  obsługiwanie, 377  
  przedziały, 377  
  wykaz tematycznie powiązanych  
  zatwierdzeń, 376  
zaniechanie łączenia, 143  
zarządca kodu źródłowego, 17  
zarządzanie  
  odniesieniami symbolicznymi, 81  
  plikami, 59  
zarządzanie archiwum, 229  
  budowniczy, 244  
  pielęgnator, 244  
  praca z wieloma archiwami, 247  
  publikowanie, 230  
  wskazówka, 236  
  serwery, 229  
  struktura archiwum, 237  
  warunki rozproszenia, 241  
  miejsce, 243  
zasada uniksowego narzędziownika, 32  
zasady zmieniania historii, 156  
zastosowanie odgałęzień, 101  
zatwierdzenia, 34, 44, 45, 49, 56, 77, 155  
  a zbiór zmian, 77  
  anulowanie skutków, 166  
  autora zmiany, 34  
  bez odniesień, 114  
  czubek odgałęzienia, 102  
  doczepki, 280  
  git fsck, 345  
  git rebase, 176  
  git rebase -i, 172  
  git svn, 312  
  głowa odgałęzienia, 102  
  grafy, 85  
  historia, 83

ID, 79  
identyfikacja błędnej wersji, 94  
identyfikator, 37  
identyfikowanie, 79  
komentarze, 378  
komunikat dziennika, 34  
komunikaty, 135  
konfigurowanie autora zatwierdzenia, 35  
korzeniowe, 45, 81  
lokalizowanie w archiwum, 331  
łączenia, 88, 142, 148  
migawki, 77, 78  
nazwy bezwzględne, 79  
nazwy wielu rodziców, 82  
nazwy względne, 81  
nieosiągalne  
  usuwanie, 192  
niepodzielne, 78  
niepodzielne zbiory zmian, 78  
obecność w gałęzi, 105  
odgałęzienie, 49  
odniesienia, 79  
odśmiecianie, 355  
pick, 174  
początkowe, 45, 89, 103, 148  
  łaty, 263  
podgląd szczegółów, 37  
podmodułów, 305  
ponowne przyłączanie utraconego  
  zatwierdzenia, 349  
poprawka, 169  
proces, 59  
przebazowanie, 170, 176, 180  
przedziały, 89  
przeglądanie, 36  
przeglądanie różnic, 37  
przekazywanie między górnym a dolnym  
  archiwum, 246  
przenoszenie, 164  
redagowanie metainformacji, 170  
ref, 80  
rekonstruowanie utraconego zatwierdzenia,  
  345  
rodzicielskie, 50, 81  
rozdzielność od publikowania, 242  
różnica symetryczna, 92  
sposoby rekonstruowania, 355  
sprawdzenie gotowości, 118  
stare, 180  
symref, 80  
szczytowe  
  zmiana, 168  
termin nieodtwarzalności, 355  
tylda, 82  
typowanie zmian, 60  
utrata, 346  
w bieżącym odgałęzieniu, 267  
w gałęziach nadzorowania, 225  
w potoku filtrującym, 324  
w SVN, 312  
włączane do HEAD, 81  
wykonanie kolejnego zatwierdzenia, 36  
wyświetlanie różnic, 117  
z więcej niż jednym potomkiem, 89  
zapamiętywane, 57  
zapisywanie komunikatów dziennika, 65  
zgniatane, 152  
zmienianie, 155  
znajdowanie, 93  
zwykle, 148  
zawartość katalogu .git, 51  
zawijanie wierszy, 273  
zbiory zmian  
  niepodzielne, 78  
zbiór poleceń instalatorskich, 51  
zdalne archiwum, 193  
zdalne odgałęzienia, 226  
  przemianowywanie, 227  
zdalny początek, 202  
zdegenerowane scenariusze do łączy, 146  
  już aktualne, 146  
  szybko do przodu, 146  
zespoły, 302, 390  
zestawianie projektów, 285  
  biblioteki, 286  
  gitlinki, 295  
  importowanie kodu do swojego projektu,  
    287  
  submodule, 295  
  wyciąganie podprojektów z użyciem  
    skryptów, 293  
  wyciągi częściowe, 286  
zestawienia różnic, 12  
zgniecenie zatwierdzeń, 174, 175  
zmiany kodzie  
  wystawianie, 335  
zmienianie zatwierdzeń, 155  
  git checkout, 167  
  git cherry-pick, 164  
  git rebase, 170

- zmienianie zatwierdzeń
  - git rebase a łączenie, 176
  - git rebase -i, 172
  - git reset, 158, 167
  - git revert, 166, 167
  - przebazowanie, 170
  - szczytowych, 168
  - zmienianie historii, 157
- znaczniki łączenia, 137, 151
- znajdowanie zatwierdzeń, 93
  - git bisect, 93
  - git blame, 97
  - kilof, 98
- znaki
  - ^, 82
  - ASCII, 101
  - niewidoczne w tekście, 101
- zunifikowany protokół rozbieżności, 115

# PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄZKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW  
w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

# Kontrola wersji z systemem Git

## Narzędzia i techniki programistów



Systemy kontroli wersji przechodzą ciągłą ewolucję. Jeszcze niedawno popularny był system CVS, który został wyparty przez SVN. Ostatnie lata to inwazja rozproszonych systemów kontroli wersji z Gitem na czele. Git pozwala każdemu programiście na posiadanie prywatnego repozytorium i korzystanie z jego dobrodziejstw bez wpływu na inne repozytoria. Jesteś ciekaw, jakie jeszcze zalety ma Git? Jeżeli tak, trafiłeś na idealną książkę, dzięki której błyskawicznie wkroczysz w świat Gita. Na początek krok po kroku przejdiesz przez proces instalacji, a następnie dostosujesz środowisko do swoich potrzeb. W kolejnych rozdziałach poznasz dostępne polecenia oraz nauczysz się zarządzać plikami. Ponadto przekonasz się, jak łatwo można stworzyć odgałęzienia kodu oraz przeglądać różnice pomiędzy wersjami pliku. Git posiada zaawansowane narzędzia do łączenia kodu – będziesz mógł je dogłębnie poznać. Na koniec przeczytasz o zaawansowanych możliwościach systemu Git, takich jak współpraca z SVN. Ta książka jest doskonałym podręcznikiem dla każdego początkującego użytkownika systemu Git, zaawansowani również znajdą tu sporo przydatnych informacji. Wykorzystaj potencjał Gita!

Dzięki tej książce:

- odkryjesz rozproszone systemy kontroli wersji
- poznasz ich zalety
- zainstalujesz i skonfigurujesz system Git
- poznasz system kontroli wersji Git

**Wykorzystaj potencjał systemu kontroli wersji Git!**

**helion.pl**  
księgarnia  
internetowa

Nr katalogowy: 17257



Księgarnia internetowa:  
<http://helion.pl>



Zamówienia telefoniczne:  
**0 801 339900**



**0 601 339900**



**Helion**

Sprawdź najnowsze promocje:

👉 <http://helion.pl/promocje>

Książki najchętniej czytane:

👉 <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

👉 <http://helion.pl/nowosci>

**Helion SA**

ul. Kościuszki 1c, 44-100 Gliwice

tel.: 32 230 98 63

e-mail: [helion@helion.pl](mailto:helion@helion.pl)

<http://helion.pl>



KOD KORZYŚCI

ISBN 978-83-246-8176-1



Cena 69,00 zł