GIT

Jakub Springer
Software Development Academy

Cel: jesteś git i „chcesz do grypsujących”

- Zrozumieć jak działa GIT i jakie jest jego miejsce we współczesnym programowaniu
- Poczuć się swobodnie w konsoli
- Poznać kilka narzędzi: gitk, GitHub
- Przetestować poznane procedury w praktyce

VCS/SCM : Czym jesteś, a jeśli już to na ile?

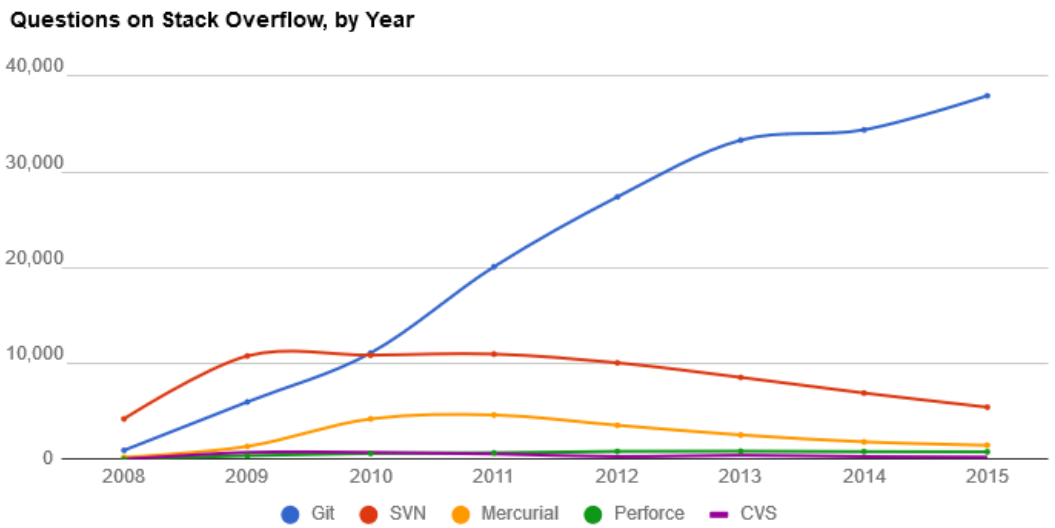
- Version control system – system kontroli wersji
- Source control management
  - Zadania:
    - Zarządzanie zmianami dokonanymi w plikach
    - Przywołanie dowolnej wcześniejszej wersji każdego pliku
    - Udostępnianie dokonanych zmian
    - Identyfikacja osoby odpowiedzialnej za daną zmianę
    - Łączenie zmian
    - Kopia zapasowa
  - Podział:
    - Lokalne: RCS
    - Scentralizowane: Subversion (Svn)
    - Rozproszone: Git, Mercurial (Hg)

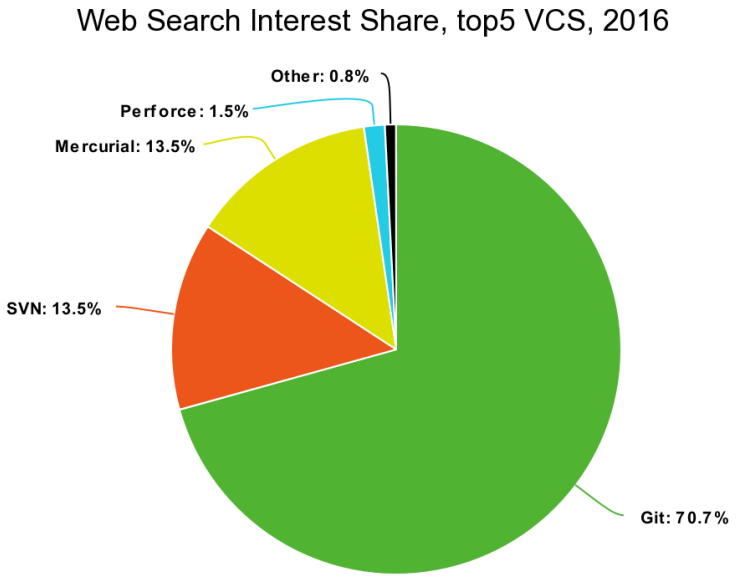
Git- historia

- Zainicjowany przez Linusa Torvaldsa do zarządzania wersjami kodu źródłowego jądra Linuxa (2005 rok)
- Wstępna wersja powstała w trzy dni

  (Start: 3-04-2005, self-hosting:7-04-2005)
- Zarządzał kodem jądra Linuksa 16-06-2005

Popularność





Źródło:1

G. I. T.

- Kiedy jesteś w dobrym humorze i chóry anielskie sławią twój postęp w projekcie: "global information tracker"
- Kiedy coś nie działa: "goddamn idiotic truckload of sh*t"

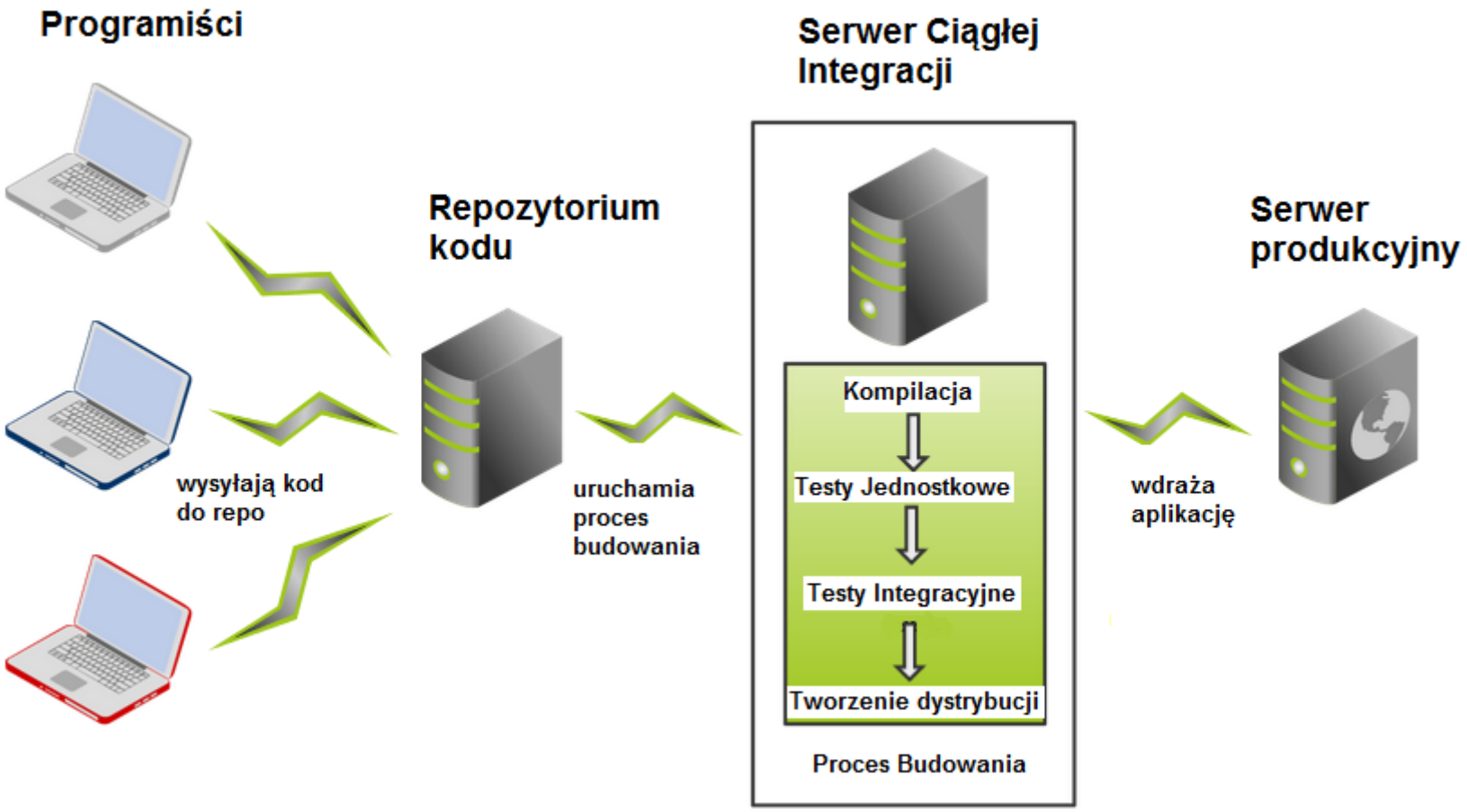
Dlaczego git?

- Szybki (szczególnie jeśli większość twojego projektu stanowi kod)
- Umożliwia pracę offline
- Zapewnia spójność danych
- Nie trudniejszy od pozostałych SCM
- Pozwala przywrócić stan plików z przeszłości
- Wiele narzędzi ułatwiających i automatyzujących pracę
- Duże wsparcie społeczności, mnóstwo poradników, kursów, samouczków

Najwięksi niegitowi gracze

- Mercuriala używają np: Facebook, Mozilla, Nginx, NetBeans
- Subversion (SVN) jest wykorzystywany przez: Backblaze, FreeBSD, Mono, and SourceForge
- Perforce był używany w Google, ale stworzyli własny system wersjonowania (Piper) w 2015 roku
- Bazaar: Canonical

„Ciągła integracja” a git



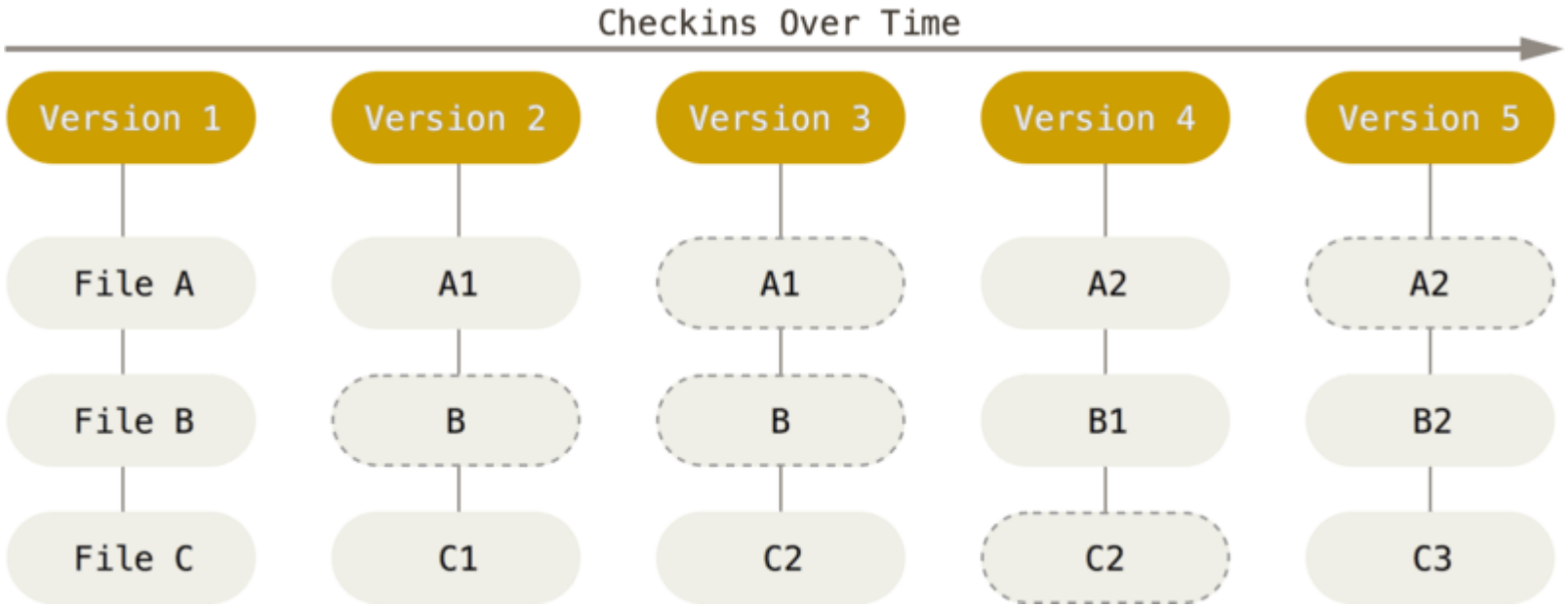
Źródło: 4

Kilka pojęć

- Funkcja skrótu, funkcja mieszająca lub funkcja haszująca – funkcja przyporządkowująca dowolnie dużej liczbie krótką, zawsze posiadającą stały rozmiar, niespecyficzną, quasi-losową wartość, tzw. skrót nieodwracalny.
- Kryptografia klucza publicznego (asymetryczna)- to rodzaj kryptografii, w którym używa się zestawów dwu lub więcej powiązanych ze sobą kluczy, umożliwiających wykonywanie różnych czynności kryptograficznych.

Jak to działa? Migawki (snapshots)

- Git traktuje dane podobnie jak zestaw migawek (ang. snapshots) małego systemu plików. Za każdym razem jak tworzysz commit lub zapisujesz stan projektu, Git tworzy obraz przedstawiający to jak wyglądają wszystkie pliki w danym momencie i przechowuje referencję do tej migawki.

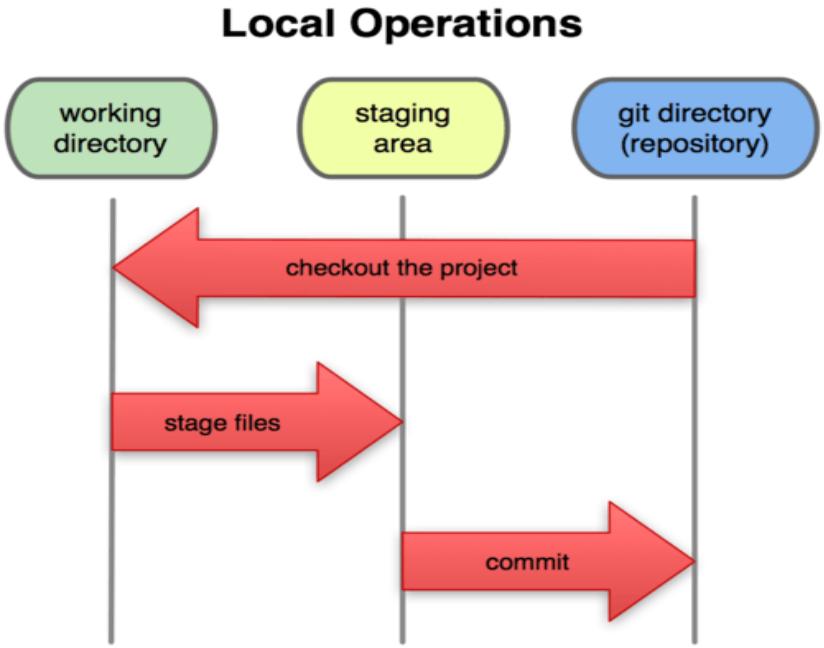


Źródło:2

Trzy stany plików w lokalnym repozytorium

zatwierdzony, zmodyfikowany i śledzony

- Katalog roboczy – Working directory
- Przechowalnia/indeks - Staging area/cache/index
- Katalog git



Git lokalnie

- Dokonujesz modyfikacji plików w katalogu roboczym.
- Oznaczasz zmodyfikowane pliki jako śledzone, dodając ich bieżący stan do przechowalni - staging. (*add*)
- Dokonujesz zatwierdzenia (rewizja = *commit*), podczas którego zawartość plików z przechowalni zapisywana jest jako migawka projektu w *katalogu* Git. Obliczana jest suma kontrolna (SHA-1)

Suma kontrolna

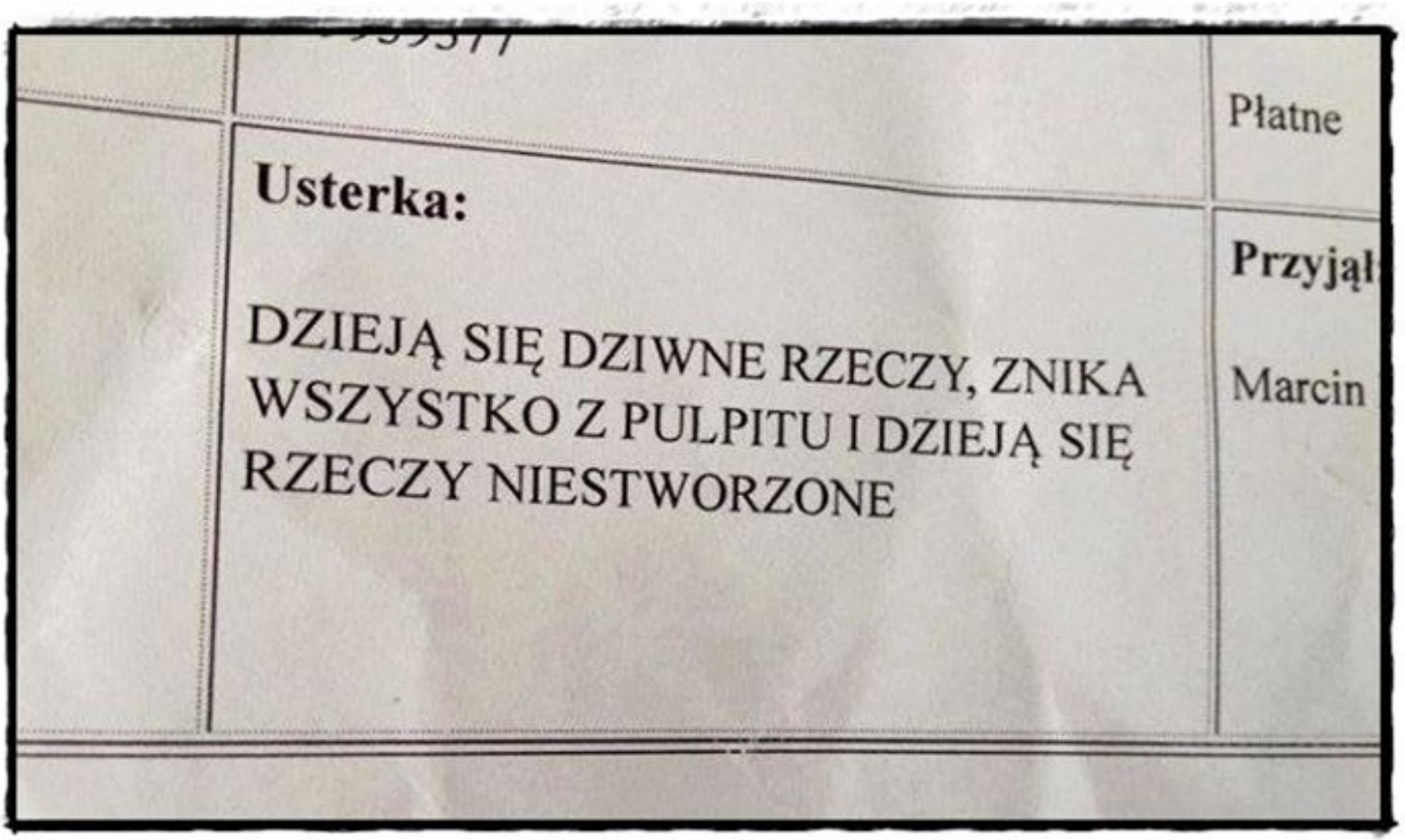
- **Generowana dla każdego obiektu przechowywanego w gicie.**
- **W przypadku rewizji (commit-a) obliczana na podstawie:**
  - sumy kontrolnej rodzica
  - sumy kontrolnej drzewa katalogu
  - autora zmian
  - osoby komitującej
- **Kolizje sumy kontrolnej**

> „Two objects colliding accidentally is exceedingly unlikely. If you had five million programmers each generating one commit per second, your chances of generating a single accidental collision before the Sun turns into a red giant and engulfs the Earth is about 50%.”

Konsola najlepszym przyjacielem

- ls – wypisanie plików i folderów
- pwd – aktualne położenie
- cd xx – zmiana folderu na xx
- mkdir xx – stwórz folder xx
- chown/chmod – zmiana właściciela/uprawnień pliku

Czego się spodziewać?



Płatne

Usterka:

Przyjął

DZIEJĄ SIĘ DZIWNE RZECZY, ZNIKA WSZYSTKO Z PULPITU I DZIEJĄ SIĘ RZECZY NIESTWORZONE

Marcin

Do pracy! Porozmawiajmy z gitem

- Tworzymy nowy katalog: **mkdir firstDir**
- Przechodzimy do niego: **cd firstDir**
- Inicjalizacja: **git init**
- Tworzymy dokument: **firstFile**
- Wypisujemy listę plików: **ls**
- Sprawdzamy status gita: **git status**
- Dodajemy plik do śledzonych: **git add firstFile → git status**
- Pierwsza rewizja! : **git commit -m 'here a commit message'**

Po każdej komendzie czytamy informacje zwrotne od gita. Zazwyczaj ma on do powiedzenia ciekawe rzeczy (dopóki go nie wyciszymy '-q').

Podstawowe komendy w pracy lokalnej

- git config (--system, --global, --local)
  - user.name krzys
  - user.email krzys@gmail.com
  - user.name
  - --list
- git status
  - -s
- git diff
  - --cached
- git add
  - <filename>
  - -A lub .
  - -p

Podstawowe komendy c.d.

- git reset –-soft <filename>

  HEAD~

- git commit

  -m

  --amend

- git log
- git checkout -- <filename>

.gitignore

- Lista plików, które git ma pominąć podczas wypisywania zmian
- Można ją stworzyć na trzech poziomach (system, global, local)
- Każda linia określa szablon (pattern) dla nazw plików, które mają być ignorowane
- Alternatywy:
  - .git/info/exclude
  - git config core.excludesFile
- Komenda do sprawdzenia, czy git ignoruje plik:
  git check-ignore <filename>

„The purpose of gitignore files is to ensure that certain files not tracked by Git remain untracked. To stop tracking a file that is currently tracked, use git rm –cached.„

gitk

- Dołączany domyślnie do instalacji gita
- Proste narzędzie do graficznej analizy sytuacji w repozytorium
- Przy dużej ilości gałęzi, rewizji, ułatwia orientację w terenie

Who is the master? Gałęzie

- **git branch**

  --list

  --all

  <new branch name>

- **git checkout <branch name>**

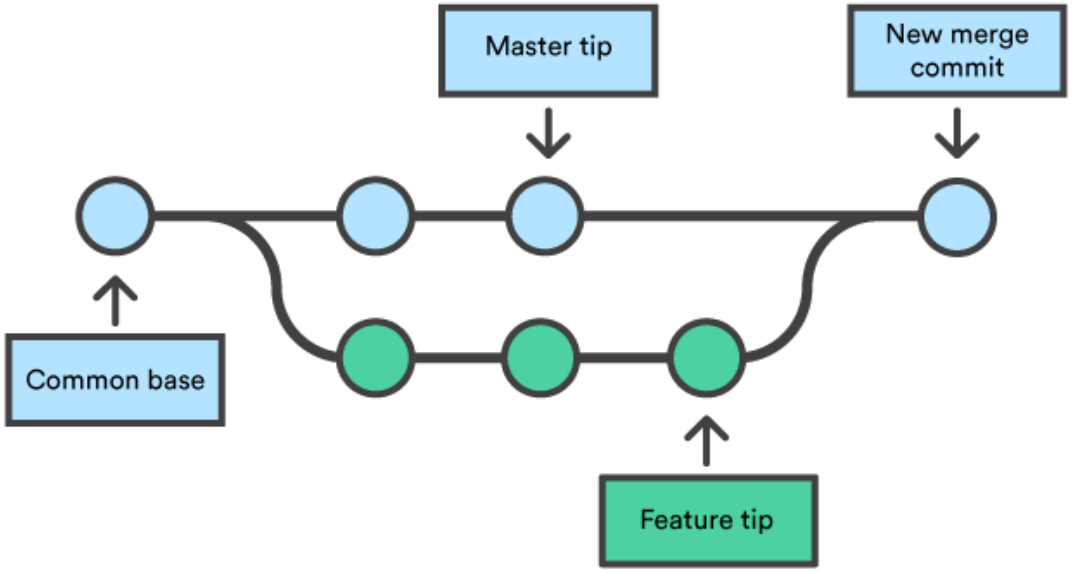
  -b <new branch name>

  <commit hash>

Merge

- *git checkout <branch name>*
- *git merge <branch2 name>*

  --no-ff



Konflikty

- Jeśli dana linia kodu została zmodyfikowana w różny sposób w scalanych gałęziach Git nie będzie wiedział jak połączyć zmiany: wystąpi konflikt, który musi rozwiązać programista
- vimdiff, IDE
- git config merge.tool vimdiff

HEAD i head, czyli co dwie głowy to nie jedna

- HEAD → wskaźnik na lokalną gałąź, na której się znajdujesz
- head → ostatnia rewizja każdej gałęzi (branch tip), szpica
- Wiadomość o ‘Detached HEAD’ po użyciu *git checkout <hash>*

Schowek

- **git stash**

  list <stash>

  show <stash>

  apply <stash>

  pop <stash>

GitHub

- Stwórzcie konto na gitHubie

Praca zdalna- pobieranie repozytorium

- git clone username@host:/path/to/repository

Praca zdalna

- **git remote**

  add origin <url to remote repo>

  -v
- **git pull**
- **git fetch**
- **git push origin master**
- **git fetch + git merge = git pull**
- **git clone /path/to/repository**

Etykiety(tagi)

- Git tag

  -a

  -d

- Tagi: lekkie (light) lub opisane (annotated)
- Domyślnie, polecenie git push nie przesyła twoich etykiet do zdalnego repozytorium.
- Git push origin <tag>
- Git push origin --tags

Hosting repozytorium

- Najpopularniejsze: GitHub, BitBucket, GitLab
- Śledzenie problemów(issue tracking)
- Zarządzanie użytkownikami
- Nadawanie uprawnień ‘per gałąź’
- Zintegrowane z narzędziami Continous Integration (np. Jenkins)

Dobre praktyki

- Jeden projekt → jedno repozytorium
- Stworzyć .gitignore na początku projektu
- Single Responsibility Principle działa również w przypadku gita. Zmiany w commicie dotyczą jednej funkcjonalności, są ze sobą logicznie powiązane
- Korzystamy z tagów (najlepiej opisanych)
- Nie commitujemy bezpośrednio do mastera
- Wypychamy zmiany do zdalnego repo w miarę często (zespół widzi postępy, możliwy jest przegląd kodu, tworzymy w ten sposób kopię zapasową)

Dobre praktyki

- **Tworzymy osobną gałąź do pracy nad konkretną funkcjonalnością (feature branch)**
  - Stworzenie nowej gałęzi zajmuje dokładnie tyle czasu, co zapisanie 41 bajtów w pliku (40 znaków + znak nowej linii).
- **Wykorzystujemy mechanizm ‘pull request’**
- **Komentarze rewizji:**
  - Linie o długości maksymalnie 72 znaki, pierwsza do 50
  - pisane w trybie rozkazującym/teraźniejszym („Fix the bug on main page”-ok, „Fixed ...” - źle!)
  - Zawierają informacje z systemu zarządzania projektem, identyfikator problemy itp

Zadania

- Utwórz konto na gitHub.com
- Utwórz repozytorium zdalne(za pomocą interfejsu użytkownika)
- Zainicjuj repozytorium lokalne w katalogu ‘myFirstRepo’
- Stwórz kilka plików tekstowych i dodaj obrazek do katalogu
- Sprawdź stan repozytorium, dodaj pliki do indeksu
- Stwórz rewizję z wiadomością ‘Initial commit’
- Stwórz etykietę opisaną ‘v0.1’
- Dodaj repozytorium zdalne
- Wyślij pliki do gitHuba
- Zweryfikuj przy użyciu gitk i strony gitHub.com

Zadania 2

- Sklonujcie repozytorium z URL na gitHubie
- Stwórzcie nową gałąź o nazwie ‘nazwiskoPY’
- Dodajcie swoje zmiany do indeksu
- Utwórzcie rewizję z wiadomością wyjaśniającą zmiany i wskazującą, które problemy (issue) zostały rozwiązane

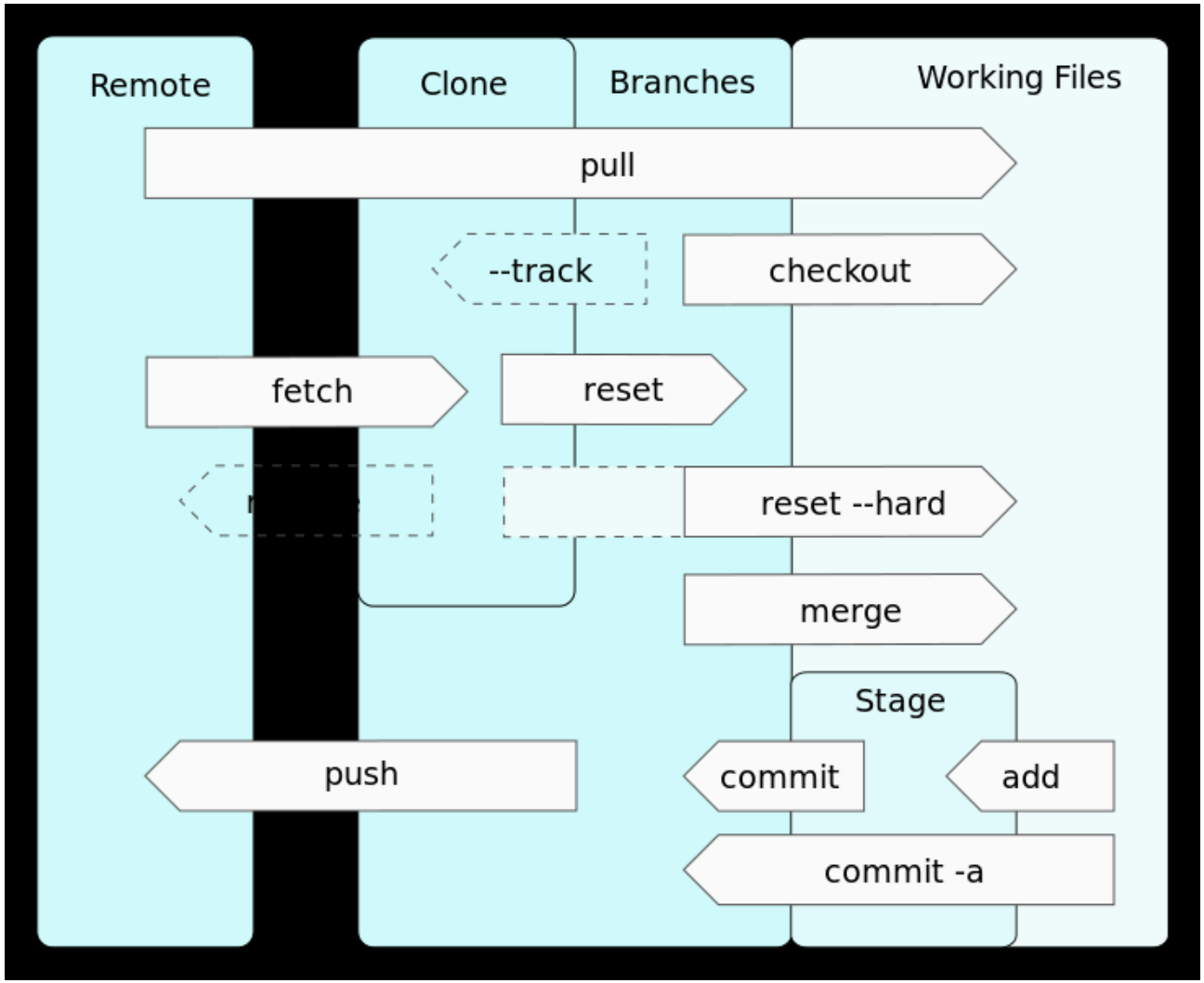
Zadania 3

- Podzielcie się na grupy
- Każda osoba z grupy tworzy nowy projekt Java według szablonu ‘HelloWorld’ w intelliJ
- Z użyciem konsoli każdy inicjuje repozytorium
- Osoba A dodaje tylko klasę Main do indeksu, tworzy .gitignore i dodaje tam automatycznie wygenerowane i niepotrzebne pliki, zatwierdza jako ‘initial commit’ i wypycha do zdalnego repozytorium do mastera
- Pozostałe osoby dodają zdalne repozytorium osoby A jako remote z nazwa ‘osobaAremote’
- Ściągają zdalne repozytorium do swojego lokalnego
- Tworzą featureBranch

Integracja z IntelliJ

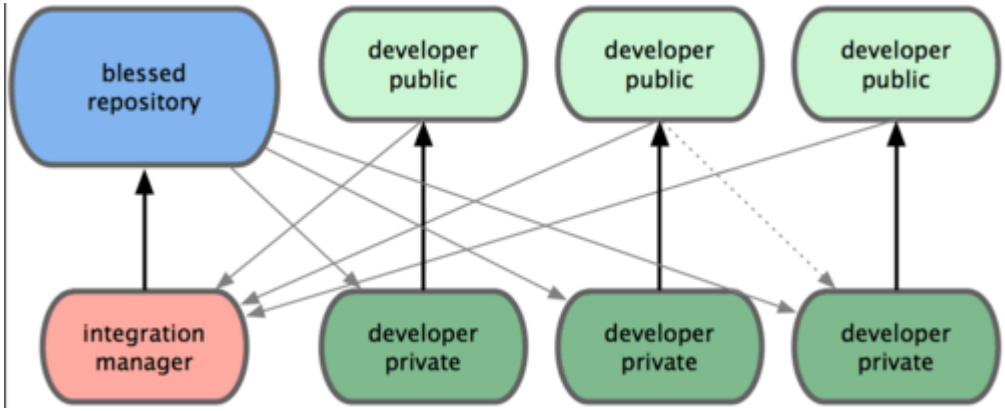
- Otworzyć działający projekt
- Stworzyć repozytorium lokalne
- Dodać/zignorować odpowiednie pliki
- Wypchnąć do swojego zdalnego repozytorium

Operacje


Remote
Clone
Branches
Working Files
pull
--track
checkout
fetch
reset
reset --hard
merge
Stage
push
commit
add
commit -a

Pull request

- Często stosowane
- Zapobiega nienadzorowanym zmianom w repozytorium



Dodatki

- Zdalne repozytorium lokalne
- Pickaxe search: git log -S ‘tekst’
- Uwierzytelnianie z użyciem ssh
- Large files support
- Feature toggles
- Rebase
- Submodules

Źródła

1. https://rhodecode.com/insights/version-control-systems-2016
2. https://git-scm.com/book/pl/v1/Pierwsze-kroki-Podstawy-Git
3. https://git-scm.com/docs/
4. https://es.atlassian.com/git/tutorials/using-branches/git-merge
5. https://github.blog/2017-03-20-sha-1-collision-detection-on-github-com/