System kontroli wersji

Seweryn Kowalski

Październik 2021

Praca jednoosobowa - problemy

- Chyba każdemu programiście się przydarzyło
  - Dobry kod działający – wprowadzenie dużej ilości poprawek/zmian, zapis kodu, całość przestaje działać, co teraz – jak wrócić do poprzedniej wersji?
  - Skasowanie ważnego plik(ów) przez pomyłkę, mnóstwo pracy i linii kodu diabli wzięli
  - Zmiana struktur pliku, kodu – chęć powrotu do wersji przed zmianami
  - Banalna awaria twardego dysku – dzień przed oddaniem projektu
- Możliwe rozwiązanie
  - Wersjonowanie pliku plik_v1.cxx
    - Ale zmiana nawet jednej linii wymaga nowe wersji

Praca w grupie - problemy

- Czyj komputer ma przetrzymywać „oficjalną” wersję projektu?
- Jak czytać zapisywać zmiany wykonanie przez dowolnego członka grupy
- Co się stanie gdy dwie lub więcej osób chcą edytować ten sam plik
- Co się stanie gdy zrobimy błąd w pliku i zawalimy cały projekt
- Jak dowiedzieć się kto pracował z jakim elementem kodu, kto napisał daną część kodu lub ją zmieniał

Rozwiązanie – system kontroli wersji

- system kontroli wersji: oprogramowanie, które śledzi i zarządza zmianą zestawu plików i zasobów
- Używasz kontroli wersji przez cały czas
  - Wbudowane edytory tekstowe / arkusze kalkulacyjne / oprogramowanie do prezentacji
    - Magiczny przycisk "cofnij" przywraca Ci wersję poprzedzającą twoją ostatnia akcję
  - Wiki są związane z kontrolą wersji, zarządzaniem aktualizacjami i zezwalaniem na użycie poprzednich plików/wersji (rollback)

system kontroli wersji

- Wiele systemów sterowania wersjami jest zaprojektowanych i wykorzystywanych szczególnie do projektów programistycznych
  - przykłady: CVS, Subversion (SVN), Git, Monotone, BitKeeper, Perforce
- Pomagają grupą we wspólnej pracy nad projektami programistycznymi
  - Współdzielą pliki do których wszyscy użytkownicy maja dostęp
  - Utrzymują „obecną” wersje plików i wersje poprzednie
  - Pozwalają zobaczyć „kto – co” zmienił
  - Zarządzają konfliktami w przypadku gdy wielu użytkowników modyfikuje ten sam plik
  - Oczywiście nie zostały stworzone tylko i wyłącznie do projektów programistycznych, działają z dokumentami i np. zdjęciami (pliki binarne) ale najlepiej sprawują się z plikami tekstowymi (zawierającymi kod)

Repozytorium

- Repozytorium (repo) miejsce przetrzymywania kopii wszystkich plików
  - Nie edytuje się plików bezpośrednio w repozytorium
  - Edytuje się lokalną/bieżąca wersje roboczą
  - Wprowadza się zmiany plików do repozytorium (commit)
- Rozwiązania:
  - Jedno (centralne) repozytorium dla wszystkich użytkowników (CVS, SVN)
  - Każdy użytkownik ma własna kopie repozytorium (Git, Mercurial)

- Pliki w katalogu roboczym muszą być dodane do repozytorium tak by mogły być śledzone

Co dodajemy do repo

- Wszystko na z czego składa się nasz projekt:
  - Pliki źródłowe (.c, .cpp, .java, .py itp.)
  - Pliki służące do budowy projektu (Makefile, build.xml)
  - Inne pliki niezbędne do budowy projektu
- Nie dodajemy plików które mogą być łatwo utworzone
  - Object files (.o)
  - Pliki wykonywalne (.exe)

Lokalizacja repozytorium

- Może być przetrzymywanie w dowolnej lokalizacji
  - Na lokalnym dysku/komputerze
    - Dobre rozwiązanie gdyż tylko chcemy mieć kontrole nad wersja w przypadku naszego własnego projektu
  - Jednak zazwyczaj wymagamy:
    - Aby repozytorium działało 24/7
    - Aby system plików był zabezpieczony (awaria dysków - RAID)
  - Opcje:
    - GitLab, GitHub

GitHub

- GitHub.com – miejsce do przetrzymywania repozytoriów Git – online
- Wiele projektów open source np. Linux Kernel
- Free space dla open source project, w przypadku prywatnych projektów trzeba zapłacić

Git pomoc

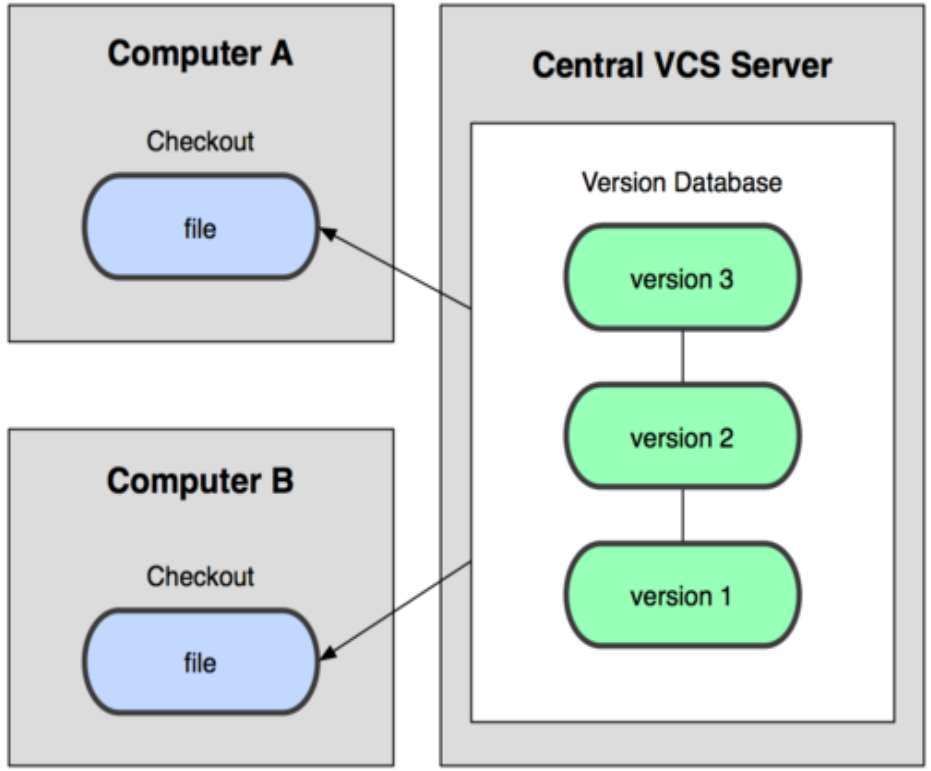
- Linia poleceń (`<verb> = config, add, commit, etc`)
  - `git help <verb>`
  - `git <verb> --help`
  - `man  git-<verb>`
- Git website: http://git-scm.com/
  - Free on-line book: http://git-scm.com/book
  - Reference page for Git: http://gitref.org/index.html
  - Git tutorial: http://schacon.github.com/git/gittutorial.html
  - Git for Computer Scientists:
    - http://eagain.net/articles/git-for-computer-scientists/

Git

- Historia
  - Utworzone przez Linusa Torvaldsa,
    - twórca Linuksa, w 2005 roku
  - wyszedł ze społeczności programistycznej Linux’a
  - przeznaczony do zarządzania wersjami w jądra Linux’a
- Zalety Git:
  - Prędkość
  - Wsparcie dla rozwoju nielinearnego (tysiące równoległych gałęzi)
  - W pełni dystrybuowany
  - Potrafi skutecznie radzić sobie z dużymi projektami

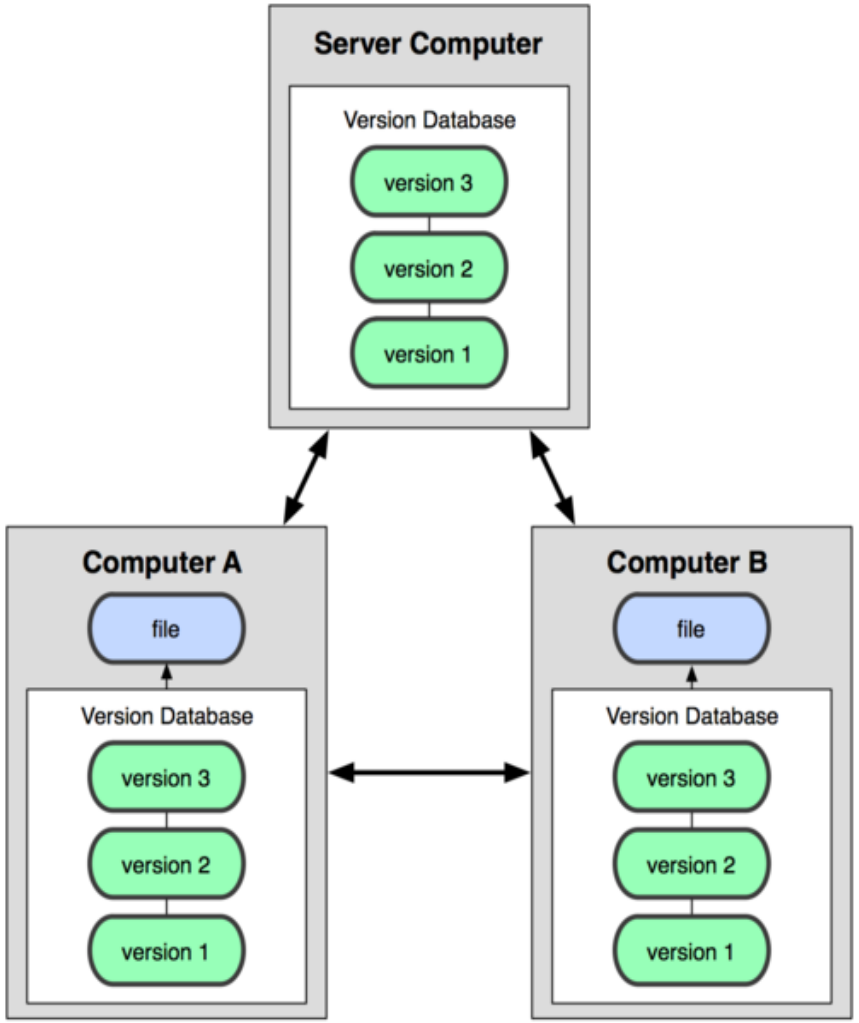
Scentralizowany system kontroli

- Subversion, CVS, Perforce, itd.
  - Centralne repozytorium serwera (repo) posiada "oficjalną kopię" kodu
  - serwer utrzymuje historię wersji repo
- Robisz "checkouts" z tego centralnego repozytorium do lokalnej kopii
  - Dokonujesz lokalnych modyfikacji
  - Twoje zmiany nie są wersjonowane
- Kiedy skończysz, ty robisz „check in" z powrotem do serwera
  - Co zwiększa/zmienia wersję repo



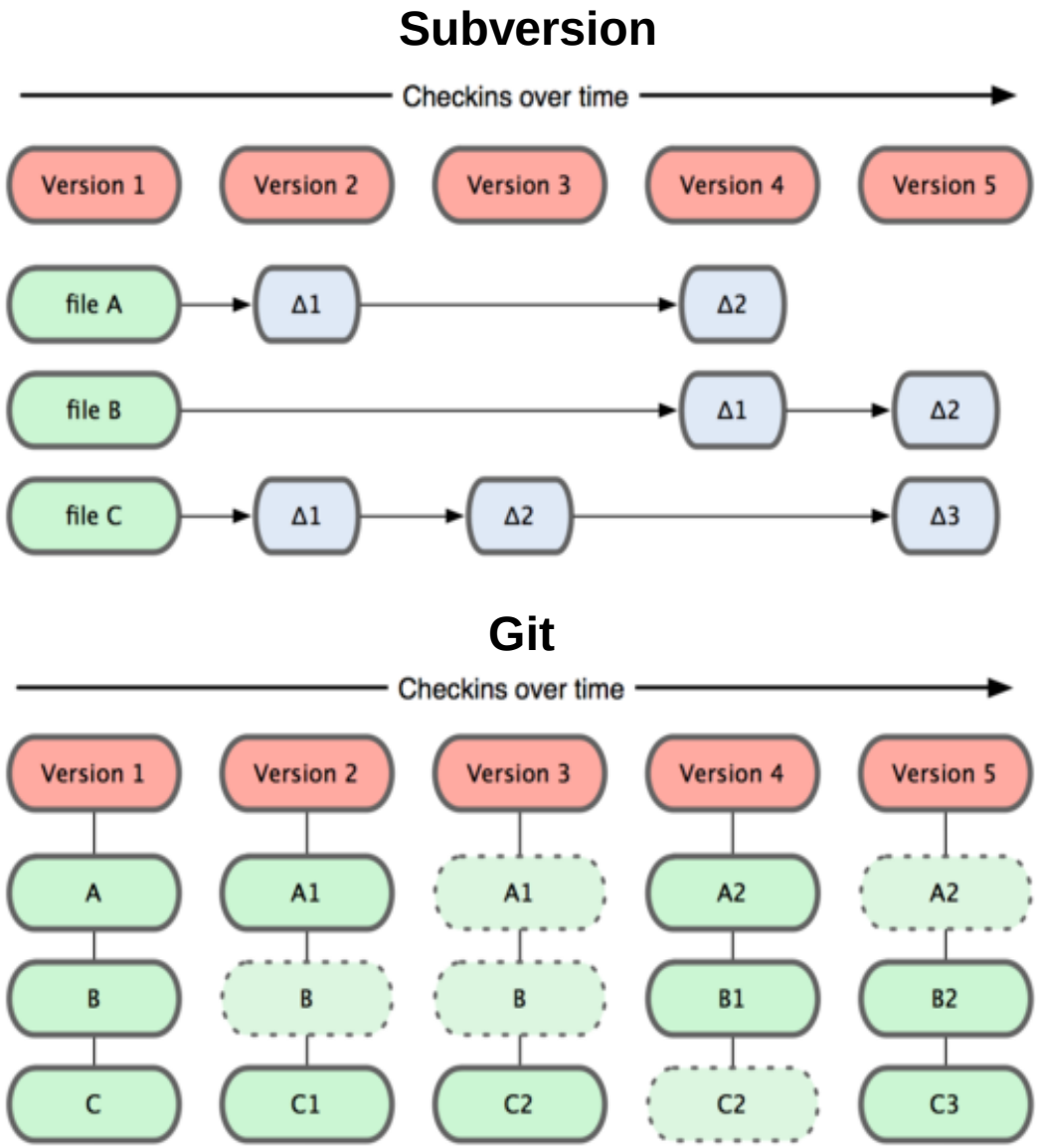
Rozproszony system kontroli wersji

- git, mercurial , itp.,
- Nie "checkout„ z centralnego repo
  - „clone" i „pull" zamiany z repozytorium
- Lokalne repo jest dokładna kopią tego co jest na serwerze zdalnym
  - Twoje repo jest "tak samo dobry" jak zdalne
- Wiele operacji ma charakter lokalny:
  - check in/out z lokalnego repo
  - commit - wprowadzenie zmiay w lokalnym repo
  - lokalne repo przechowuje historię wersji
- Gdy jesteś gotowy, możesz wprowadzić „push" zmiany na serwer



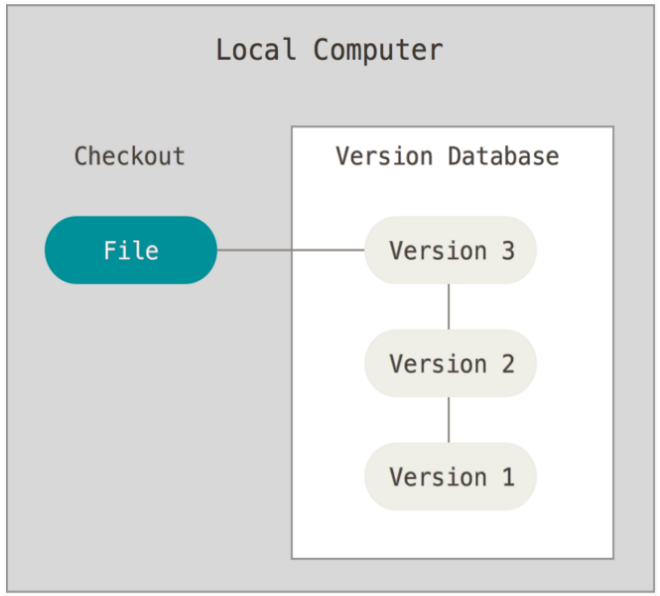
Snapshot

- Centralizowane VCS śledzą każdy plik
- Git przetrzymuje informacje o całym projekcie:
  - Każda wersja to kopia całego projektu
  - Mimo że część plików pozostaje nie zmieniona

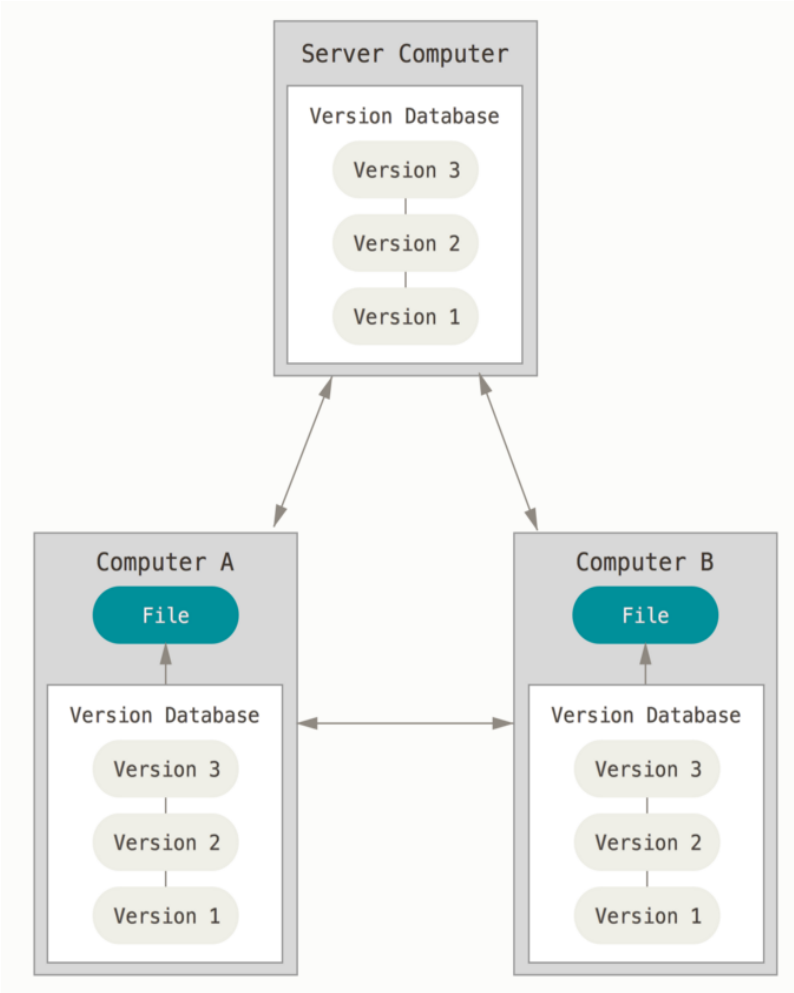


Jak używać Git

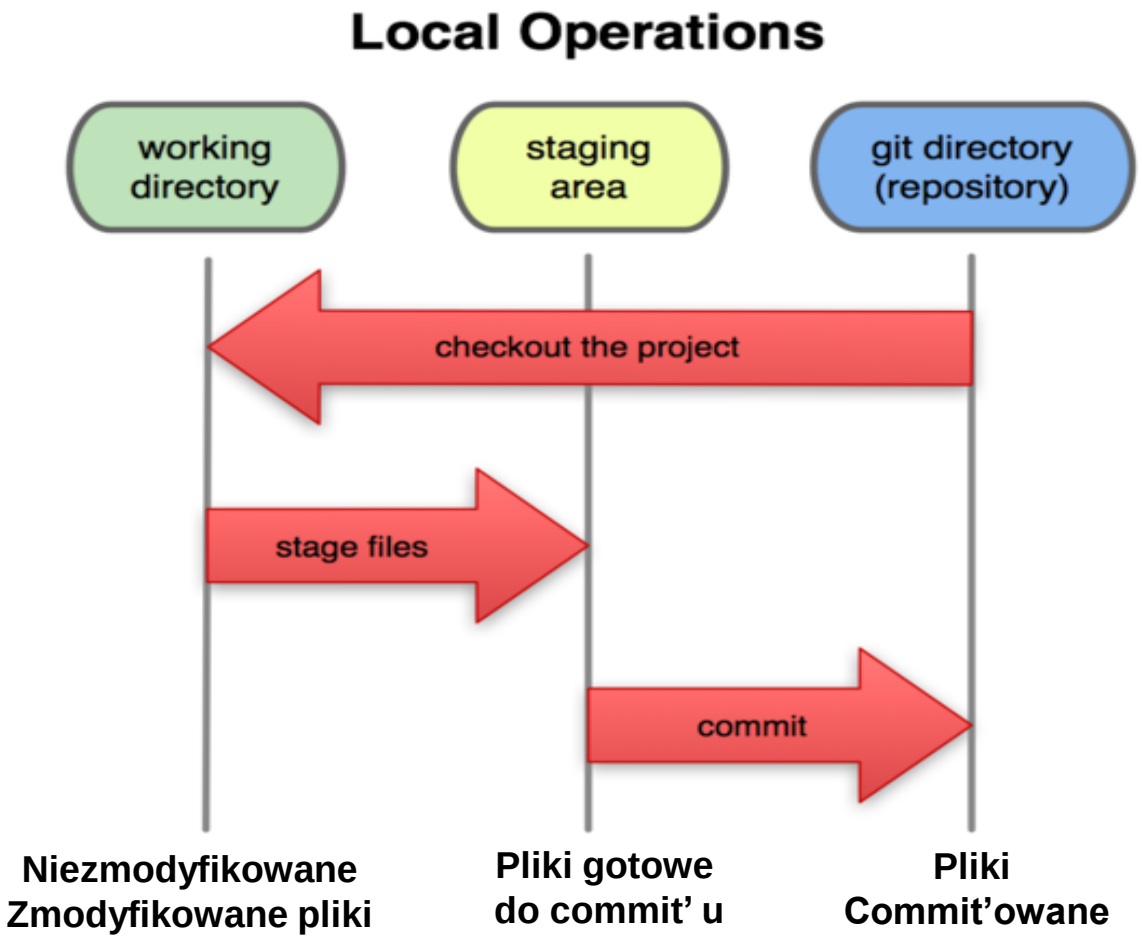
Jeden użytkownik
Jeden komputer



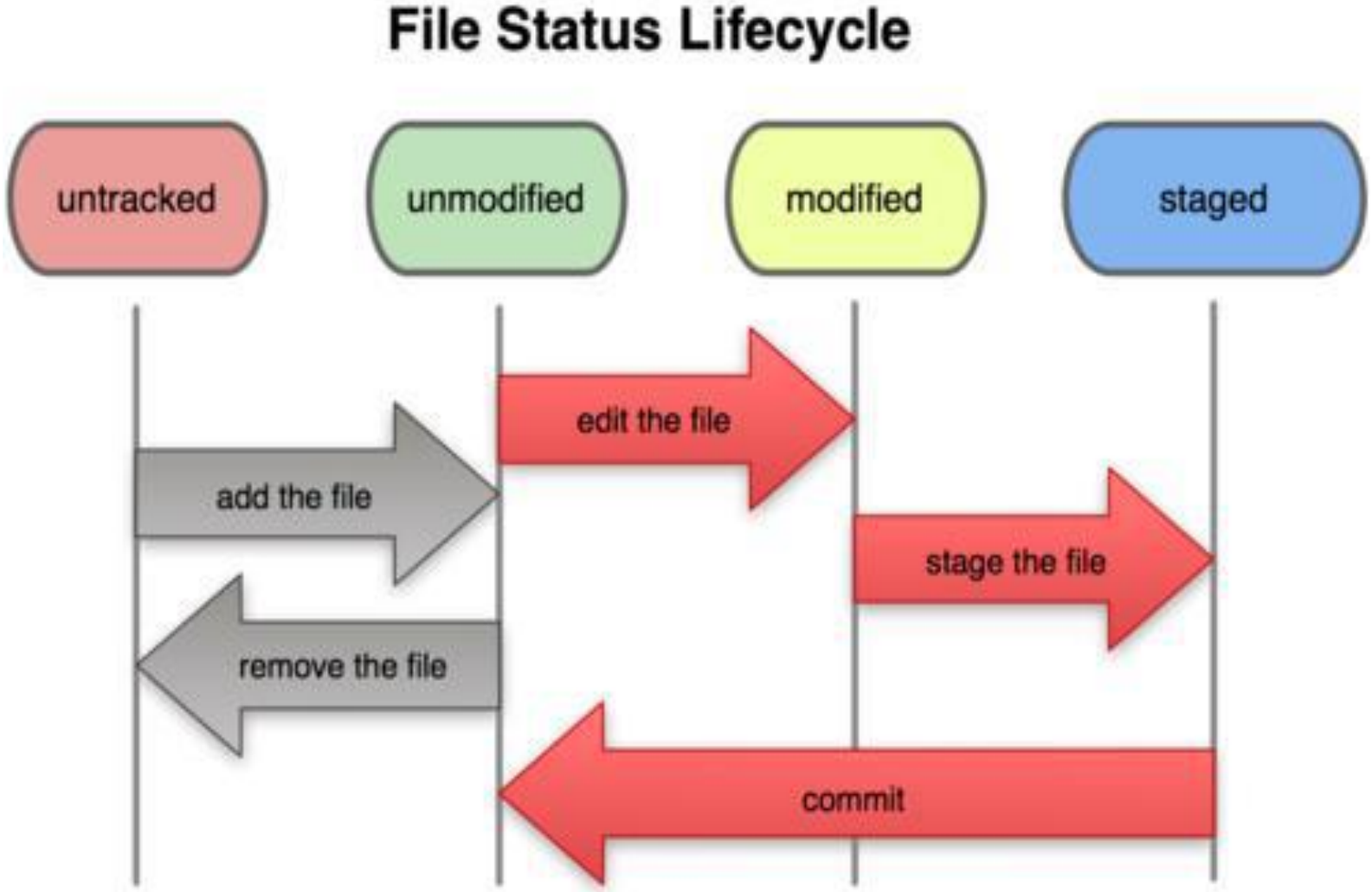
Wielu użytkowników
Wiele komputerów



Lokalna praca z repozytorium



Git – cykl życia pliku



Basic Workflow

- Zmodyfikuj pliki w katalogu roboczym
- Stage files – przygotuj je do commit’u
- Zrób commit – bierze pliki takie jak są i ładuje je katalogu Git (lokalna kopia repozytorium)

Git checksum

- W SVN każda modyfikacja centralnego repozytorium zwiększa numer wersji
- W Git użytkownicy maja własne lokalne repozytoria przed wysłaniem ich na serwer
- Git generuje unikatowy SHA-1 hash (40 znaków w układzie szesnastkowym) dla każdego commit'u
- Odnosimy się więc raczej do tego ID a nie numeru wersji
- Często widzimy tylko pierwsze 7 znaków:
  - 1677b2d Edited first line of readme
  - 258efa7 Added line to readme
  - 0e52da7 Initial commit

Początkowa konfiguracja Git

- Ustawienie nazwy i adresu email powiązanego z commit'ami
  - `git config --global user.name "Bugs Bunny"`
  - `git config --global user.email bugs@gmail.com`
  - Można to sparwadzić `git config –list`
- Ustawianie typu edytora dla wprowadzania informacji o commit'ach
  - `git config --global core.editor emacs`
  - Domyślnie `vim`

Tworzenie repozytorium

- Dwa scenariusze – nie stosuje się ich równocześnie
  - Nowe lokalne repo w katalogu roboczym
    - `git init`
      - To tworzy .git katalog
      - Teraz można wprowadzać „commit'ować" pliki do tego repozytorium
    - `git add` *`filename`*
    - `git commit –m "`*`commit message`*"
  - Klonowanie zdalnego repo do lokalnego katalogu
    - `git clone` *`url localDirectoryName`*

Git polecenia

Polecenie	opis
`git clone url [dir]`	Tworzenie kopii zdalnego repozytorium
`git add file`	Dodaje zmiany we wskazanym pliku do commita
`git commit`	Ładowanie zmian – nowa wersja
`git status`	Wyświetlenie listy zmienionych plików
`git diff`	Szczegółowe wyświetlenie zmian
`git pull`	Pobranie najnowszych zmian z aktywnego brancha zdalnego
`git push`	Wysłanie zmian do zdalnego repozytorium

Git - praca

- Po raz pierwszy dodajemy plik, który ma być śledzony, a za każdym razem, zanim załadujemy plik, musimy go dodać do obszaru staging:
  - `git add foo.cxx bar.h`
- Ładowanie plików ze staging do repo:
  - `git commit –m "Fixing bug #22"`
- Odwołanie zmian w pliku przed ich załadowaniem (commit'em) :
  - `git reset HEAD -- filename` (unstages the file)
  - `git checkout -- filename` (undoes your changes)
- To wszystko jest wykonywane na lokalnej kopii repozytorium

- Sprawdzanie stanu plików w katalogu roboczym
  - `git status`
- Sprawdzenie co było zmodyfikowane
  - `git diff`
- Lista zmian w staged area
  - `git diff –cached`
- Wyświetlić log lokalnego repozytorium
  - `git log`

Przykład

- [rea@attu1 superstar]$ **emacs rea.txt**
- [rea@attu1 superstar]$ **git status**
- *no changes added to commit*
- *(use "git add" and/or "git commit -a")*
- [rea@attu1 superstar]$ **git status -s**
- *M rea.txt*
- [rea@attu1 superstar]$ **git diff**
- *diff --git a/rea.txt b/rea.txt*
- [rea@attu1 superstar]$ **git add rea.txt**
- [rea@attu1 superstar]$ **git status**
- *# modified: rea.txt*
- [rea@attu1 superstar]$ **git diff --cached**
- *diff --git a/rea.txt b/rea.txt*
- [rea@attu1 superstar]$ **git commit -m "Created new text file"**