# Czym jest kontrola wersji?

* Grafik chce zachować każdą wersję pliku graficznego
* Programista chce zachować wcześniejszy układ strony internetowej
* Student chce zachować wcześniejsze wersje swojej pracy inżynierskiej

# VCS - Version Control System

* Naprzeciw wymaganiom wyszedł VCS
* Pozwalałby stworzyć kopię projektu
* Wrócić do starszego pliku lub ddtworzyć stan całego projektu
* Porównać wprowadzone zmiany
* Dowiedzieć się, kto i kiedy zmodyfikował pliki, które powodują problemy

Nawet jeśli stracimy część danych – można je łatwo odzyskać

# „Własnoręczna” kontrola wersji

* Własnoręczne kopiowanie plików do katalogów; sprytniejsi - oznaczanie ich datą
* Wady – łatwo zapomnieć gdzie co jest, zmodyfikować błędy plik, skopiować nie te dane
* **Kopia** to inaczej **rewizja (commit)**

Lokalny system kontroli wersji

* Do poradzenia sobie z problemami, programiści stworzyli dawno lokalne systemy kontroli wersji (prosta baza danych, która śledziła zmiany na plikach)
* RCS – używany do tej pory na OS X w Narzędziach Developerskich
* Zapisuje dane różnicowe - czyli zawierające jedynie różnice pomiędzy plikami

# Ale co w przypadku pracy zespołowej?

* Bazą danych nie można było się dzielić w prosty sposób

# Scentralizowane systemy kontroli wersji

* Aby poradzić sobie z tym problemem stworzono scentralizowane systemy kontroli wersji
* Jeden serwer, na którym znajdują się pliki poddane kontroli wersji
* Klienci łączą się z serwerem, by uzyskać dostęp do najnowszych plików
* Przez wiele lat był to standardowy model kontroli wersji
* Gdy kopię wrzuci Michał i Adam - **Na serwer są wysyłane tylko zmiany, a nie cały pliki**

# Scentralizowane systemy kontroli wersji – zalety, wady

* Jest wiele zalet, ale jeszcze więcej wad

# Rozproszone systemy kontroli wersji

* Klienci nie dostają dostępu jedynie do najnowszych wersji plików ale w pełni kopiują całe repozytorium
* Gdy jeden z serwerów, używanych przez te systemy do współpracy, ulegnie awarii, repozytorium każdego klienta może zostać po prostu skopiowane na ten serwer w celu przywrócenia go do pracy

# Krótka historia Gita

* Szybkość
* Prosta konstrukcja
* Silne wsparcie dla nieliniowego rozwoju (tysięcy równoległych gałęzi)
* Pełne rozproszenie
* Wydajna obsługa dużych projektów, takich jak jądro Linuksa (szybkość i rozmiar danych)

# **GIT**

* Każda operacja wykonuje się lokalnie = szybo
  + Dostęp do historii projektu
  + zmiany wprowadzone pomiędzy bieżącą wersją pliku, a jego stanem sprzed miesiąca
  + można zrobić prawie wszystko będąc poza zasięgiem sieci

# Rysunek Bitbucketa

* Gdy w połowie lutego padł bitbucket, każda firma trzymająca tam projekty używające GITa, mogła dalej bez przeszkód pracować
* **GIT nie działa w tle**
* Po **push** zobaczyć na githubie, czy są commity

# Branche

# Gałęzie są używane do rozwijania funkcjonalności odizolowanych od siebie.

# Gałąź master jest domyślną gałęzią

# Używaj innych gałęzi do rozwoju projektu, a kiedy skończysz scalaj je z powrotem z gałęzią główną.

# Branche – brak zmian na masterze

# Pull Requests

* Odbranchowaliśmy się po to, by móc porównać zmiany z masterem
* Te zmiany możemy komuś pokazać – nazywa się to code review
* Na końcu wykonujemy **merge**, czyli dogrywamy zmiany z brancha do mastera

# Code Review

* Zostawiamy miny w kodzie, a druga osoba pomaga nam je rozbroić

Gdy .gitignore nie działa, oznacza to problem z kodowaniem pliku. Trzeba zamienić na utf-8

Komenda usuwająca pliki z gita to (najpierw robimy commita, by status- s było puste, potem robimy tą komendą, potem git add –A, i powinno już działać

git rm -r --cached.