

# Architektura dużych projektów bioinformatycznych

Bartek Wilczyński

[bartek@mimuw.edu.pl](mailto:bartek@mimuw.edu.pl)

<http://www.mimuw.edu.pl/~bartek>

**Wykład 1. - Wprowadzenie**

28. II. 2018

# Cele do wykonania

- Zapoznanie się z narzędziami do współpracy nad tworzeniem oprogramowania (git, github itp.)
- Przegląd powszechnie dostępnych projektów implementujących istotne rozwiązania dla bioinformatyki
- Rola otwartego kodu dla wiarygodności i reprodukowalności badań naukowych
- Praktyczne zajęcia pozwalające “oswoić” się z wybranym dużym projektem bioinformatycznym
- Wykonanie projektu zespołowego

# Plan wykładów

- 1.Wprowadzenie
- 2.Projekty BioX (Perl, Python, Java, itp...)
- 3.Oprogramowania typu LIMS (Galaxy, Base itp)
- 4.Workflows (Taverna, Galaxy, itp)
- 5.Przeglądarki genomowe (UCSC, Ensembl, itp)
- 6.Wizualizacja (Cytoscape, Circos, itp)
- 7.Środowiska do obliczeń naukowych (Matlab, Mathematica, Bioconductor itp)
- 8.Wizualizacja danych – wyzwania i rozwiązania
- 9.-15. Referaty studentów

# Plan laboratoriów

## Zagadnienia:

- systemy kontroli wersji (lokalne (na przykładzie SVN i rozproszone np. GIT)
- Python (modyfikacje istniejących projektów/tworzenie własnych, automatyzacja testów, współpraca online)
- Galaxy (tworzenie workflow'ów w Galaxy)

## Zaliczenie:

- mniejsze zadanie indywidualne
- Większe zadanie grupowe

# Kryteria zaliczenia

- Zaliczenie laboratorium
  - Suma punktów z zadań daje proponowaną ocenę z ćwiczeń
- Zaliczenie wykładu
  - Referat na ocenę
- Średnia ocena z ćwiczeń i referatu daje punkt wyjścia do egzaminu ustnego, na którym można ocenę zmienić.
- Bez ustnego maksymalnie oc. Dobra
- Na ustnym głównie pytania dotyczące wykładu.

# Proponowane tematy referatów

- Przegląd narzędzi open-source do:
  - Biologii strukturalnej (białka, RNA, leki)
  - Filogenetyki
  - Chemoinformatyki
  - Hurtownie danych (Biomart itp)
  - Genomiki funkcjonalnej (chip-seq itp)
  - Machine learning (klasowanie, klasyfikacja itp)
  - Przetwarzanie obrazów (z mikroskopów, skanerów itp)
  - Własne sugestie...?

# Schemat dobrego referatu

- Elementarne wprowadzenie do tematyki i roli w niej oprogramowania (co badamy, po co badamy, dlaczego potrzebujemy do tego niebanalnego oprogramowania)
- Przegląd typowych problemów w danej dziedzinie i przykłady popularnego oprogramowania do ich rozwiązywania, z uwzględnieniem:
  - Licencji (open-source, komercyjne)
  - Sposobu zarządzania kodem
  - Krótkiej historii projektu
  - Funkcjonalności

# Parę słów o OpenSource

- Free software to nie tylko Open Source
- Projekt GNU
- Licencje GNU – GPL w różnych wersjach
- Czy Free Software to “Public Domain”?
- Czy można zarobić na Open Source?
- A co z patentami?
- I co to ma wspólnego z bioinformatyką?

# Ciekawe daty z historii oprogramowania bioinformatycznego

- 1965 – macierze PAM
- 1970 – słowo bioinformatyka (Hogeweg)
- 1981- algorytm Smitha-Watermana
- 1982 - Genbank
- 1987 – serwer bioinfo w EMBL (początki EMBOSS)
- 1988 - Algorytm Clustal
- 1991 - BLAST
- 1996 – BioPerl (release 0.7 w 2000)
- 1999 – BioPython
- 2001 – Bioconductor
- 2005 – Galaxy project

# Systemy kontroli wersji

- SCCS – source code control system (1972) – na systemy mainframe
- RCS – revision control system (1982) – na komputery osobiste – kontrola na poziomie pojedynczego pliku
- CVS – concurrent versioning system (1986) - “nakładka” na RCS, pozwala śledzić całe projekty, gałęzie kodu, pierwsze implementacje klient-serwer przez telnet i ssh
- SVN – Subversion (2000) – “lepszy” CVS – pozwala na zmianę nazw i przenoszenie plików, nieco lepsza obsługa gałęzi, własny serwer svn, nadal centralne zarządzanie historią

# Czemu rozproszona kontrola wersji?

- Centralny system wymaga administracji
  - Każda gałąź kodu wymaga przechowywania w tym samym miejscu
  - Przydział praw nie pozwala na łatwy udział osób spoza kręgu zaufania
  - Powstaje pojedynczy punkt (serwer), którego awaria rodzi ogromne problemy
- Rozproszony system pozwala na kopiowanie całej historii w różne miejsca i rozwój ich zupełnie niezależnie
  - Konieczne bardzo efektywne algorytmy do łączenia gałęzi kodu
  - Każda zmiana jest podpisywana elektronicznie, aby uniemożliwić “nadpisanie” kodu bez wiedzy użytkownika dokonującego “merge”
  - Nadal istnieje zwykle “oficjalne” repozytorium, ale łatwo je odtworzyć w razie awarii.

# Rozproszone systemy kontroli wersji

- Bitkeeper (1998) – komercyjny system pozwalający na swobodne dzielenie się zmianami pozwalający na łatwe łączenie gałęzi, używany przez zespół rozwijający linuksa do 2005, kiedy naruszenia licencji spowodowały konflikt i konieczność zmiany
- GIT (2005) – system open-source stworzony przez Linusa Torvalds'a na potrzeby projektu GNU linuks dla zastąpienia bitkeepera i obecnie bardzo popularny dzięki serwisom t.j. github i bitbucket
- Wiele innych alternatyw open source : bazaar, mercurial, GNU Arch, Veracity Monotone, także bitkeeper (opensource od 2016)