Wykład 1: Wprowadzenie do systemu Unix/Linux

Prowadzący

Dr inż. Bartosz Kozak,

bartosz.kozak@upwr.edu.pl,

konsultacje Piątek 10:00-11:00, pokój 422,

Katedra Genetyki, Hodowli Roślin i Nasiennictwa

Wydział Przyrodniczo - Technologiczny

Program kursu

1. Podstawy pracy w systemie Linux

Program kursu

1. Podstawy pracy w systemie Linux
2. Wprowadzenie do RNAseq

Program kursu

1. Podstawy pracy w systemie Linux
2. Wprowadzenie do RNAseq
3. Analiza jakości odczytów

Program kursu

1. Podstawy pracy w systemie Linux
2. Wprowadzenie do RNAseq
3. Analiza jakości odczytów
4. Mapowanie odczytów

Program kursu

1. Podstawy pracy w systemie Linux
2. Wprowadzenie do RNAseq
3. Analiza jakości odczytów
4. Mapowanie odczytów
5. Analiza ekspresji

Program kursu

1. Podstawy pracy w systemie Linux
2. Wprowadzenie do RNAseq
3. Analiza jakości odczytów
4. Mapowanie odczytów
5. Analiza ekspresji
6. DE - analiza różnicowa

Program kursu

1. Podstawy pracy w systemie Linux
2. Wprowadzenie do RNAseq
3. Analiza jakości odczytów
4. Mapowanie odczytów
5. Analiza ekspresji
6. DE - analiza różnicowa
7. Analiza GO

Bioinformatyka

Czym jest Bioinformatyka ?

Czym się zajmuje Bioinformatyka ?

Bioinformatyka

Czym jest Bioinformatyka ?

Czym się zajmuje Bioinformatyka ?

- Bioinformatyka jako nauka biologiczna
  Wykorzystanie komputerów do analizy danych biologicznych

Bioinformatyka

Czym jest Bioinformatyka ?

Czym się zajmuje Bioinformatyka ?

- Bioinformatyka jako nauka biologiczna
  Wykorzystanie komputerów do analizy danych biologicznych
- Bioinformatyka jako informatyka (*computer science*)
  Tworzenie oprogramowania do analizy danych biologicznych

Linux

Systemy operacyjne z rodziny GNU/Linux (potocznie Linux) są klonami systemu komputerowego Unix opracowanego pod koniec lat 60-tych XXw. System ten stworzony został z myślą o komputerach wykorzystywanych przez wielu użytkowników jednocześnie. Dlatego jest to dominujący system operacyjny w przypadku komputerów serwerowych oraz superkomputerów. Oprogramowanie wykorzystywane w analizach bioinformatycznych, ze względu na zapotrzebowanie dużych mocy obliczeniowych pisane jest z myślą o wykorzystaniu na serwerach i pracą pod kontrolą systemu operacyjnego wykorzystywanego na tego typu urządzeniach **Linux**.

Linux

Wiele popularnych programów bioinformatycznych (szczególnie wykorzystywanych w analizie danych NGS w tym w analizach RNAseq) nie posiada plików binarnych przeznaczonych dla systemów Windows oraz macOS. Programy, które posiadają pliki binarne dla systemów Windows oraz macOS i tak tworzone były z myślą o pracy pod kontrolą Linuxa i dlatego najlepiej działają pod kontrolą tego systemu.

Linux

Większość programów wykorzystywanych w analizie danych NGS (w tym w analizach RNAseq) działa w środowisku wiersza poleceń systemu Linux ponieważ w takim środowisku najczęściej pracuje się na serwerach. W związku z tym znajomość podstaw pracy w środowisku wiersza poleceń jest **niezbędna** do przeprowadzenai analiz RNAseq.

Dostęp do systemu Linux

1. Instalacja na własnym komputerze (rekomendowany)
2. Instalacja na własnym komputerze jako maszyna wirtualna (rekomendowany)
3. Wykorzystanie wbudowanego wiersza poleceń
   - terminal dla systemu macOS
   - WSL/WSL2 dla systemu Windows 10
4. Emulatory (Cygwin, bash for git) starsze wersje systemu Windows
5. Dostęp do komputera serwerowego
   - VPS *Virtual Private Server* (AWS, DigitalOcean)
   - Serwer KGHRiN (**rekomendowany**)

Dostęp do konta na serwerze KGHRiN

Aby uzyskać dostęp do konta na serwerze należy użyć protokołu ssh (*secure shell*). W tym celu w terminalu lokalnego komputera należy wpisać poniższy kod, zastępując **username** nazwą użytkownika, a **hostname** nazwą(adresem) serwera.

```
ssh username@hostname
```

Dostęp do konta na serwerze

Wyświetlony zostanie kompunikat z prośbą o podanie hasła. Po wpisaniu hasła (wpisane znaki nie będą wyświetlane) nawiązane zostanie połączenie, i wyświetlony zostanie prompt. Od tego momentu mamy dostęp do naszego konta i możemy korzystać z zasobów serwera.

```
[username@hostname ~]$
```

Test

Wszystkie polecenia proszę wykonać w terminalu

Tekst wprowadzany do plików tekstowych można kopiować

Test

- Proszę utworzyć katalog **Test_1** w katalogu domowym
- W katalogu **Test_1** proszę utworzyć 4 podkatalogi
  - **Polecenie_I**
  - **Polecenie_II**
  - Dwa katalogi o dowolnej nazwie
- W katalogu **Polecienie_I** utworzyć 3 pliki: (*tekst.txt*, *skrypt.sh*, *test.txt*)
- Do pliku *tekst.txt* proszę wprowadzić tekst:

  "Informatyka – interdyscyplinarna dziedzina nauki, wykorzystująca metody i narzędzia informatyczne do rozwiązywania problemów z nauk biologicznych. Informatyka obejmuje rozwój metod obliczeniowych, służących do badania struktury, funkcji i ewolucji genów, genomów i białek."

Test

- Do pliku *skrypt.sh* proszę wprowadzić następujący tekst:

```
wc -l
```

- Polecenie

```
wc -l
```

pozwala na wyświetlenie liczby linii w pliku tekstowym. Proszę zmodyfikować plik *spkrypt.sh* tak, aby można było go uruchomić, oraz aby po uruchomieniu pozwalał na wyświetlenie liczby linii w pliku *tekst.txt*

Test

- W pliku test.txt proszę napisać zdanie :

  "Ponadto odpowiada za rozwój metod wykorzystywanych do zarządzania i analizy informacji biologicznej, gromadzonej w toku badań genomicznych oraz badań prowadzonych z zastosowaniem wysokoprzepustowych technik eksperymentalnych."

- Zawartość plików *tekst.txt* oraz *test.txt* proszę przenieść do nowego pliku *definicja.txt*
- W pliku *definicja.txt* proszę zamienić słowo "Informatykańa "Bioinformatyka"(bez otwierania pliku w edytorze tekstowym)

Test

- Całą zawartość katalogu Polecenie_1 proszę skopiować do katalogu Polecenie_2
- Proszę usunąć z katalogu Polecenie_1 pliki *tekst.txt* oraz *test.txt*
- Proszę przejść do katalogu Polecenie_2

Podstawowe komendy - Nawigacja

```
pwd
cd
pwd
cd ~
pwd
cd /home/
pwd
```

Podstawowe komendy - Nawigacja

```
cd ..
pwd
ls
ls -r
ls -r -a
ls -ra
ls --reverse --all
```

Podstawowe komendy - Pliki i Katalogi

```
cd ~
mkdir katalog_test
mkdir -p nauka/katalog1
cd katalog_test
touch plik1
cp plik1 plik2
ls
mv plik2 smiec
cp smiec ../nauka/katalog1/
rm smiec
cd ..
rm -r katalog_test
```

Podstawowe komendy - Pliki i Katalogi

```
cd nauka/katalog1/
cat Bioinformatyka jest cool! > tekst1.txt
cat tekst1.txt
cat /etc/passwd
more /etc/passwd
less /etc/passwd
```

Podstawowe komendy - Pliki i Katalogi

Do edycji plików tekstowych można wykorzystać edytory dostępne z wiersza poleceń **nano** oraz **vim**. Pliki tekstowe możemy także edytować za pomocą edytora strumieniowego **sed**.

Ćwiczenie 1:

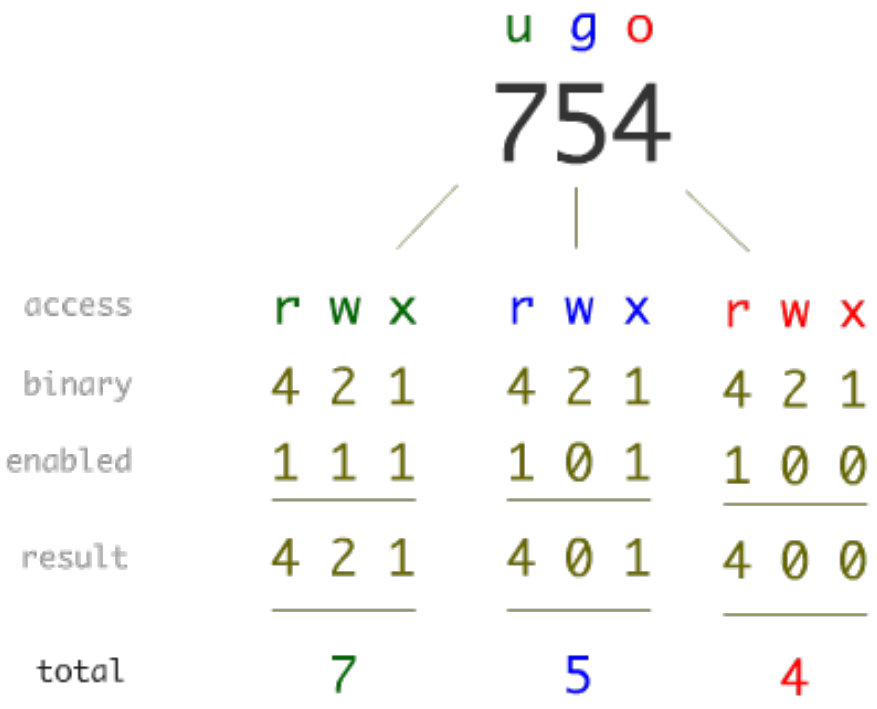
W katalogu domowym utworzyć katalog **cwiczenie_1**, przejść do katalogu **cwiczenie_1**. Za pomocą dowolnego edytora wprowadzić tekst:

> *Informatyka – interdyscyplinarna dziedzina nauki wykorzystująca metody i narzędzia informatyczne do rozwiązywania problemów z nauk biologicznych.*

do pliku cwiczenie1.txt

Pliki i Katalogi - Prawa dostępu

```
ls -l
```



Pliki i Katalogi - Prawa dostępu

```
chmod 777 cwiczenie1.txt
ls -l
chmod 400 cwiczenie1.txt
ls -l
chmod 300 cwiczenie1.txt
cat cwiczenie1.txt
chmod 644 cwiczenie1.txt
```

BASH - podstawowe komendy

```
whoami
date
wc -l cwiczenie1.txt
whereis blastn
grep -c ">" seq1.fa
```

Informacje na temat komend

```
man man
man ls
man wc
man grep
```

Informacje na temat komend

[opcja]	opcja nie jest wymagana (można ją pominąć)
<opcja>	opcja jest wymagana (nie można jej pominąć)
element ...	element może być powtórzony (nieskończenie wiele razy)
opcja1|opcja2	można użyć opcji 1 lub opcji 2, nie można ich użyć razem
opcja	tekst kursywą wskazuje informacje, które muszą być podane poprzez odpowiednią wartość. Jest to opcja lub parametr, który musi być zastąpiony wartością

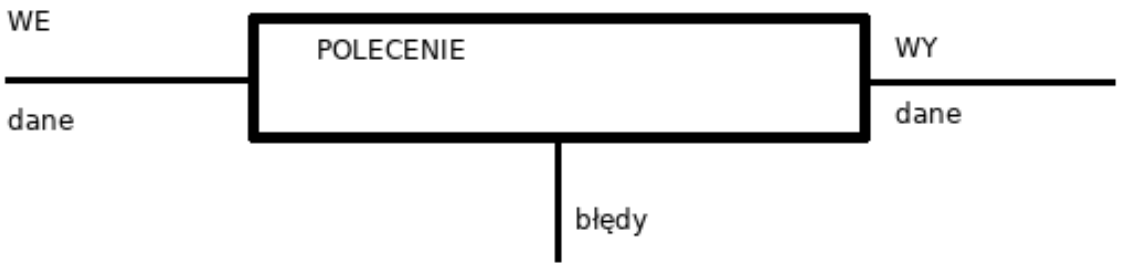
Papline = potokowanie

Programy pisane dla systemów unixowych tworzone są według zasady KISS (ang. *Keep It Simple, Stupid*), tłumaczonej jako "nie komplikuj, głupku". Każdy program powinien wykonywać jedno zadanie, ale powinien wykonywać to zadanie ekstremalnie dobrze. Następnie użytkownik może łączyć poszczególne programy w ciągi tzw. "papline", w których wynik działania jednego programu przekazywany jest do kolejnego. Zastosowanie takiego podejścia pozwala na osiągnięcie bardzo efektownych (i efektywnych) rezultatów przy zastosowaniu stosunkowo prostych narzędzi. Poszczególne programy stanowią 'sayklocki za pomocą których możemy zbudować zaawansowane narzędzie. Umożliwia ono wykonanie zaawansowanych analiz lub operacji, której żaden z elementów składowych nie byłby w stanie wykonać samodzielnie, a napisanie osobnego programu wykonującego te operacje wymagałoby dużego nakładu czasu i pracy.

Podstawowe wejście (Standard Input) i podstawowe wyjście (Standard output)

Dane wprowadzone do programu nazywamy "Podstawowym wejściem", natomiast dane generowane przez program nazywamy śtandardowym wyjściem". Standardowe wyjście domyślnie przekazywane jest do terminala (wyświetlenia na ekranie monitora jako tekst), może ono jednak zostać przekierowane do kolejnego programu i stać się śtandardowym wejściem"dla tego programu lub do pliku i zostać zapisane.

Podstawowe wejście (Standard Input) i podstawowe wyjście (Standard output)



Papline = potokowanie

Przykładem może być użycie polecenia **date**. Wynik działania tego programu czyli aktualna data będzie właśnie standardowym wyjściem - dane generowane przez program (polecenie).

```
date
sob, 6 paz 2018, 20:06:18 CEST
```

Papline = potokowanie

Standardowy błąd to informacja generowana przez program, która nie jest zasadniczą informacją generowaną przez program. Może to być komunikat o błędzie lub inne informacje generowane przez program. Przykładem może być użycie polecenia **date** z niewłaściwą opcją np *dzisjest*. Spowoduje to wyświetlenie informacji o błędzie. Standardowe błędy także domyślnie są przekierowywane do terminalu (wyświetlane w oknie terminala, jako tekst).

```
date dzisjest
date: bledna data:     dzisjest
```

Papline = potokowanie

Standardowe wejście (czyli dane, które wprowadzamy do programu standardowo pochodzą z klawiatury). Mieliśmy okazję zapoznać się z wprowadzaniem danych tą metodą na poprzednich ćwiczeniach, kiedy używaliśmy komendy cat . Standardowe wejście może być jednak pobrane z pliku lub z standardowego wyjścia innego programu, uruchamianego przez działaniem programu, w którym dane stanowią standardowe wejście.

Przekierowywanie

"Standardowe wyjście" może być przekierowane do pliku. Możemy zapisać w pliku wynik działania programu, zamiast wyświetlać ten wynik w oknie terminala. Jeżeli chcemy przekierować standardowe wyjście do jednego pliku, a standardowy błąd do innego możemy to zrobić podając cyfry: 1 - odpowiadającą standardowemu wyjściu lub 2 - odpowiadającą standardowemu błędowi oraz nazwę pliku, do którego chcemy zapisać nasz output. Symbol > spowoduje zapisanie do pliku (jeżeli plik istnieje, to jego zawartość zostanie wyczyszczona i pojawi się w nim output). Zastosowanie symbolu » spowoduje dopisanie do pliku (aktualna zawartość pliku zostanie zachowana).

Przekierowywanie

Przykład:

```
date 1>wyniki1 2>blad1
cat wyniki1
cat blad1
date dzisjest 1>wyniki2 2>blad2
cat wyniki2
cat blad2
```

Przekierowywanie

Jeżeli chcemy zapisać jedynie standardowe wyjście do pliku możemy pominąć cyfrę 1.

Przykład:

```
uname > system.txt
cat system.txt
```

Przekierowywanie

Standardowe wejście również ma skojarzoną cyfrę i jest to cyfra 0. Jeżeli program ma czytać z pliku, czyli jeżeli chcemy, żeby standardowe wejście stanowił plik (a nie klawiatura), stosujemy symbol $>$. Podobnie, jak w przypadku standardowego wyjścia, przy standardowym wejściu także możemy pominąć cyfrę 0.

Przykład:

```
cat < system.txt
```

Możemy także zastosować równoczesne przekierowanie dla standardowego wejścia i wyjścia.

Przykład:

```
cat < system.txt > nowy.txt
cat nowy.txt
```

Łączenie programów w papline

Polecenie **cat** wyświetla podany tekst w terminalu. Standardowym wejściem dla tego programu będzie tekst podany przez użytkownika (możemy jednak zastosować przekierowanie np. z pliku). Standardowym wyjściem będzie ten sam tekst wyświetlony w terminalu. Możemy jednak przekierować standardowe wyjście programu **cat**, czyli tekst wprowadzony przez użytkownika do kolejnego programu. W naszym przykładzie programem tym będzie **cowsay**. Program ten przekształca dane wejściowe (tekst), wyświetlając podany tekst wraz z dorysowaną za pomocą znaków ASCII krową (lub innym zabawnym rysunkiem). Standardowe wyjście programu **cowsay** możemy następnie przekierować do kolejnego programu **lolcat**.

Łączenie programów w papline

Program ten generuje standardowe wyjście jako kolorowy tekst (którym jest standardowe wejście tego programu).Łączenie programów, czyli przekierowanie standardowego wyjścia jednego programu do kolejnego dokonuje się poprzez zastosowanie tz. operatora pipe (|).

Łączenie programów w papline

Przykład:

```
cat < system.txt > nowy.txt
cat > krowa.txt
Bioinformatyka jest cool!
cat < krowa.txt | cowsay | lolcat
< Bioinformatyka jest cool! >
 ------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Łączenie programów w papline

Jak widzimy efekt działania tych programów pozwala uzyskać ciekwe rezulaty, znacznie "potężniejszeńiż przy zastosowaniu pojedynczych programów z papline.

Czym jest skrypt?

Jest to program komputerowy napisany w języku skryptowym, wykonywany wewnątrz aplikacji (interpretatora). Interpretator wykonuje instrukcje w kolejności podanej w skrypcie. Zastosowanie instrukcji warunkowych pozwala na ominięcie pewnych kroków lub powtórzenie innych.

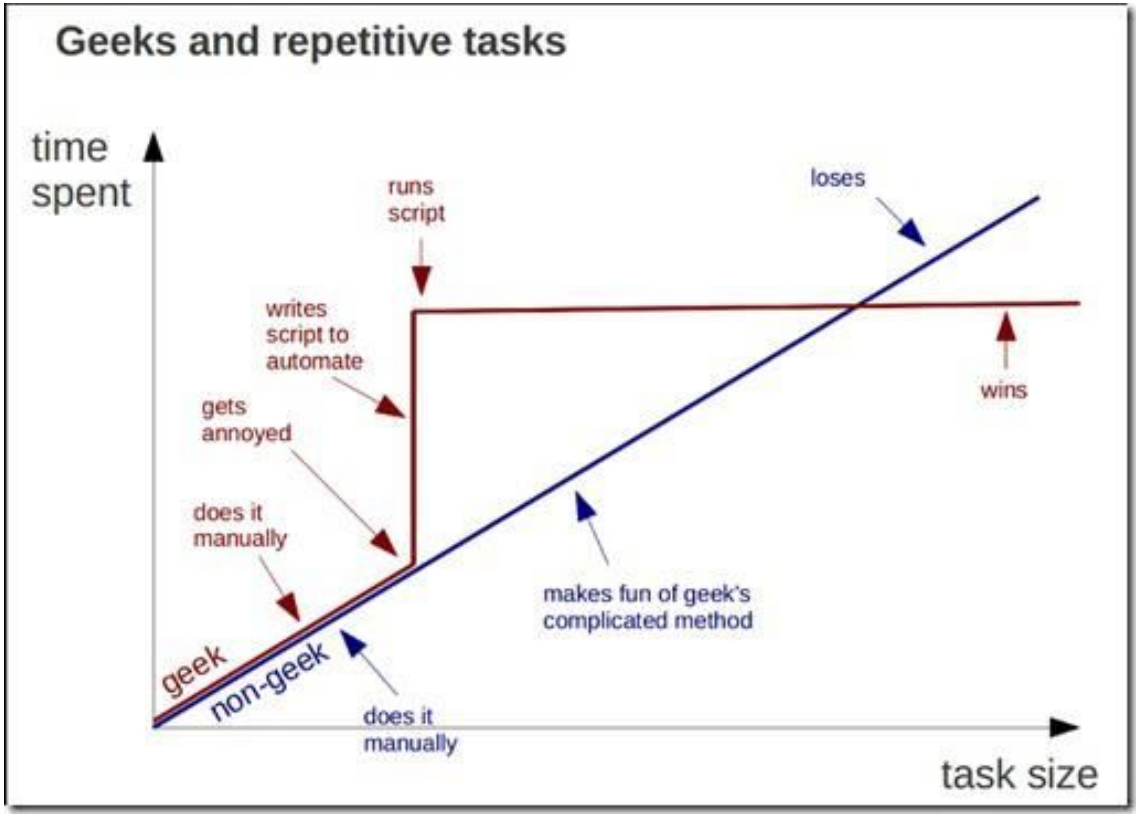
Czym jest skrypt?

Do popularnych języków skryptowych możemy zaliczyć:

- Bash - domyślny język powłoki wielu systemów Unixowych (macOS, Linux), dostępny także w systemie Windows 10 (tylko wersja 64 bit)
- Python - bardzo popularny jezyk ogólnego zastosowania, szczególnie w ostatnich latach, coraz częściej stosowany w bioinformatyce; posiada rozbudowaną bibliotekę ułatwiającą analizę bioinformatyczną - Biopython
- Perl - obecnie traci na popularaności, wiele wczesnych narzędzi bioinformatycznych napisanych jest właśnie w tym języku

W jakim celu tworzymy skrypty?

Skrypty tworzone są, aby zautomatyzować wykonywanie powtarzalnych czynności. Np. zmiana formatu danych. Wykorzystanie skryptów pozwala zaoszczędzić czas.



Budowa skryptu

Skrypt powinien zawierać:

- Ścieżka do interpretatora (system musi wiedzieć jakiego programu użyć do wykonania skryptu – prawidłowej interpretacji poleceń) - “Shebang”
- Instrukcje (polecenia danego języka)
- Komentarze (pomagają zrozumieć kod innym użytkownikom)

bash - Hello World

Tworzymy nowy plik o nazwie **msp.sh**. Ponieważ skrypt tworzymy w języku bash, musimy podać lokalizację interpretatora bash. W tym celu użyjemy programu **which**. Po wpisaniu **which** bash system poda nam lokalizację programu bash . Powinna to być ścieżka */bin/bash*. Teraz możemy otworzyć nasz skrypt dowolnym edytorem (np. **nano**) *nano msp.sh* i utworzyć nasz pierwszy skrypt. Zaczniemy od najprostszego przykładu. Nasz skrypt wyświetli prosty komunikat tekstowy na ekranie. Do wpisywania tekstu na ekranie służy polecenie echo 'Przykładowy tekst' . Komentarze zaznaczamy stawiając znak # , natomiast pierwszą linijkę (Shebang) rozpoczynamy kombinacją znaków #! , a następnie podajemy lokalizację naszego interpretatora. Nasz skrypt powinien wyglądać następująco:

bash - Hello World

```
#!/bin/bash
# Pierwsza linijka zawiera informacje
#o lokalizacji interpretatora, kolejne
#zasadnicza czesc skryptu
#(te polecenia zostan  wykonane)
echo 'Hello World!'
echo 'Pierwszy skrypt napisany na zajeciach'
```

Wykorzystanie poleceń bash w skrypcie

W skryptach bashowych możemy także wykorzystywać polecenia wiersza poleceń takie jak np. whoami , date , ls lub cd .

```
#!/bin/bash
echo "Skrypt napisany przez $(whoami)"
echo "Uruchomiony w dniu $(date +%D)."
echo "Papline nadal dziala!" | cowsay
```