PRACOWNIA INFORMATYCZNA



Wykład 1: Wiersz poleceń i filozofia Uniksa

dr inż. Bartosz Czech

Katedra Genetyki · Uniwersytet Przyrodniczy we Wrocławiu

bartosz.czech@upwr.edu.pl

Plan i cel

Forma kursu

Tryb **blended** — większość nauki online, we własnym tempie, na Moodle.

Element	Opis
2 wykłady wprowadzające	Na żywo, na starcie — ogólne wprowadzenie do całości (to jeden z nich)
10 modułów online	Samodzielna praca na Moodle: materiały i zadania
Zaliczenie pisemne	Na żywo, na osobnym spotkaniu na koniec kursu (termin ustalany z grupą)

> Wykłady wprowadzające dają obraz całości i wspólny język. Właściwa praca odbywa się w modułach — w nich ćwiczy się konkretne narzędzia.

Zakres materiału

1. Terminal i bash: powłoka, potoki i filtry tekstowe
2. Przetwarzanie tekstu: `sed` i `awk`
3. Skrypty powłoki: podstawy
4. Skrypty powłoki: elementy zaawansowane
5. Bazy danych, formaty i interwały genomowe
6. Reprodukowalność: Git, Conda i Docker
7. Python w bioinformatyce: parsowanie sekwencji i Biopython
8. Python w praktyce: budowa narzędzi wiersza poleceń
9. Snakemake: budowa reprodukowalnych pipeline'ów
10. Wizualizacja danych w R: `ggplot2` i raporty Quarto

Fundament to warsztat bash (moduły 1–4), dalej dane (5), reprodukowalność (6), Python (7–8), automatyzacja (9) i wizualizacja (10).

Platforma Moodle

Cały kurs na uczelnianej platformie **Moodle** — poza dwoma wykładami na żywo, jedyne potrzebne miejsce.

- **Logowanie** — tym samym kontem co poczta uczelniana; kurs pojawia się po zapisaniu na przedmiot
- **Materiały** każdego modułu: wprowadzenie, przykłady i dane do zadań
- **Zadania** — treści i miejsce oddania rozwiązań
- **Ogłoszenia** — komunikaty prowadzącego i terminy oddania

Reguła praktyczna

Warto włączyć powiadomienia z kursu i zaglądać na Moodle regularnie — tam obowiązują terminy oddania zadań.

Zawartość kursu na Moodle

- **Moduł 0** — przygotowanie środowiska: instalacja dla każdego systemu, paczki danych, edytor
- **Moduły 1–10** — właściwy kurs: materiały, dane, treści zadań i miejsce oddania

> Slajdy obu wykładów wprowadzających (PDF) są dostępne na stronie bczech.github.io/teaching — nie trzeba niczego przepisywać w trakcie.

Rytm pracy w semestrze

Kurs jest samodzielny, ale wymaga **systematyczności** — najlepiej pracować regularnie przez cały semestr.

- Kolejne moduły są budowane na poprzednich, więc warto przerabiać je po kolei
- Na jeden moduł warto zarezerwować **kilka godzin** pracy własnej rozłożonych na tydzień
- Rozwiązania oddaje się na bieżąco, żeby nie kumulować zaległości

Reguła praktyczna

Najczęstszą przyczyną kłopotów nie jest trudność materiału, lecz odkładanie modułów na koniec. Fundament (moduły 1–4) trzeba przerobić na bieżąco, bo kolejne części z niego korzystają.

Praca z modułem

1. **Materiały** — wprowadzenie z przykładami do samodzielnego uruchomienia
2. **Zadania** — lista zadań albo mini-projekt na dostarczonych danych
3. **Oddanie** — rozwiązanie przesłane na Moodle w terminie

Do oddania

Plik(i) wskazane w module — zwykle jeden plik tekstowy z użytymi poleceniami lub skryptami i istotnymi fragmentami wyników; w Modułach 6, 8 i 10 dwa pliki. Nazwa według wzoru `M#_nazwisko_nrAlbumu` (np. `M3_kowalski_123456.txt`).

> Każdy moduł kończy się wynikiem ocenianym w skali 0–4 pkt; punkty z modułów wchodzą do oceny końcowej.

Zasady zaliczenia

Ocena końcowa: 100 pkt

- **Moduły**: 10 × 0–4 pkt (ocena całościowa) — razem 40 pkt
- Moduł oddany po terminie — **0 pkt**
- **Zaliczenie pisemne** — 60 pkt

Próg zaliczenia: **50/100 pkt**; decyduje suma punktów

Skala ocen

Suma punktów	Ocena
≥ 90	5,0
80–89	4,5
70–79	4,0
60–69	3,5
50–59	3,0
< 50	2,0

Terminy oddania modułów — w Moodle; zaliczenie pisemne na koniec kursu.

Zaliczenie pisemne

- Termin: do ustalenia ze studentami
- Zakres: całość kursu, od wiersza poleceń po pipeline'y i wizualizację
- Forma: pytania koncepcyjne oraz czytanie i pisanie poleceń i fragmentów skryptów
- Języki: bash, awk i Python

Tryb pracy i konsultacje

Tryb pracy

- Kurs samodzielny — bez formalnej listy obecności na modułach
- O ocenie decyduje suma punktów z modułów i zaliczenia pisemnego ($\geq 50/100$)

Konsultacje: poniedziałek 14:00–15:30, pokój 306 w Katedrze Genetyki
(proszę o wcześniejszy kontakt mailowy: `bartosz.czech@upwr.edu.pl`)

Pomoc: forum przedmiotu na Moodle oraz konsultacje. Pytania o materiał i zadania warto zadawać na bieżąco, a nie tuż przed terminem.

Literatura i materiały

Podręczniki

- V. Buffalo — *Bioinformatics Data Skills* (O'Reilly, 2015)
- M.G. Sobell — *Linux. Programowanie w powłoce. Praktyczny przewodnik* (Helion, 2014; rozdziały o `sed`/`awk`)

Darmowe, dostępne online

- Software Carpentry — *The Unix Shell*
- Kross — *The Unix Workbench*
- Janssens — *Data Science at the Command Line*
- MIT — *The Missing Semester*
- explainshell.com — rozkłada polecenie na części

Plan dzisiejszego wykładu

1. Po co bioinformatykowi wiersz poleceń
2. Terminal i powłoka (przypomnienie)
3. Instalacja i konfiguracja środowiska
4. Budowa polecenia i praca w terminalu (przypomnienie)
5. Praca z plikami i katalogami (przypomnienie)
6. Filozofia Uniksa: potoki i filtry
7. Dane i formaty w bioinformatyce
8. Dobre praktyki i typowe pułapki

1. Po co bioinformatykowi wiersz poleceń

Czym jest ten przedmiot

Pracownia Informatyczna — praktyczny warsztat codziennej pracy bioinformatyka z danymi.

- Celem nie jest programowanie dla samego programowania, lecz **warsztat**: sprawne i powtarzalne przetwarzanie danych biologicznych
- Punktem wyjścia jest **wiersz poleceń**, a docelowo także skrypty, kontrola wersji, środowiska, kontenery i pipeline'y
- Kurs jest kontynuacją przedmiotu **„Elementy linii poleceń bash"** (I rok); Moduł 1 powtarza i porządkuje podstawy, więc nie wymaga ich pamiętania

Cel praktyczny

Umiejętność przetworzenia całego zbioru plików z danymi własnym, powtarzalnym rozwiązaniem — zamiast ręcznej pracy na każdym pliku osobno.

Skala danych w bioinformatyce

- Pojedynczy eksperyment sekwencjonowania to **miliony** odczytów i pliki rzędu **gigabajtów**
- Plik FASTQ z jednej próbki RNA-seq ma zwykle kilka–kilkanaście GB; projekt to dziesiątki takich plików
- Genom człowieka w postaci tekstowej to około **3 miliardy** znaków

Dlaczego nie arkusz kalkulacyjny

Excel zatrzymuje się na ok. milionie wierszy i otwiera cały plik w pamięci. Danych bioinformatycznych zwykle nie da się w nim ani otworzyć, ani przetworzyć.

Dwa sposoby pracy z komputerem

Interfejs graficzny (GUI)

- Praca przez klikanie w okna i ikony
- Intuicyjny na start
- Każda czynność wykonywana ręcznie, raz
- Trudno opisać i powtórzyć krok po kroku

Wiersz poleceń (CLI)

- Praca przez wpisywanie poleceń tekstowych
- Wymaga nauki, ale daje pełną kontrolę
- Zapisane polecenie można powtórzyć
- Łatwo automatyzować i skalować

CLI = *Command Line Interface*, czyli tekstowy interfejs wiersza poleceń.

Przewaga wiersza poleceń

- **Reprodukowalność** — zapisane polecenie jest dokładnym opisem tego, co zrobiono; da się je powtórzyć i sprawdzić
- **Automatyzacja** — tę samą operację można wykonać na tysiącu plików jednym poleceniem
- **Skala** — narzędzia czytają dane strumieniowo, więc rozmiar pliku nie ogranicza pracy
- **Praca zdalna** — serwery i klastry obliczeniowe udostępniają zwykle wyłącznie terminal

> Większość profesjonalnych narzędzi bioinformatycznych w ogóle nie ma interfejsu graficznego — działają tylko z wiersza poleceń.

Reprodukowalność: klikanie a polecenia

Praca przez klikanie

„Otworzyłem plik, zaznaczyłem kolumnę, wybrałem z menu sortowanie, potem odfiltrowałem…”

Po miesiącu nikt — łącznie z autorem — nie odtworzy dokładnie tych kroków.

Praca przez polecenia

Jedna linia, którą można wkleić do publikacji, wysłać koledze i uruchomić za rok z tym samym wynikiem.

```
sort -t$'\t' -k3,3n dane.tsv > posortowane.tsv
```

Praca zdalna: serwery i klastry

- Poważne obliczenia uruchamia się na **zdalnych serwerach** i **klastrach**, nie na własnym laptopie
- Połączenie przez **SSH** daje sam wiersz poleceń, bez okien i myszy
- Pliki przesyła się poleceniami (`scp`, `rsync`); zadania zleca się systemowi kolejkowemu

> Klaster i SLURM — przegląd na Wykładzie 2, w Module 9 teoretycznie (Snakemake na klastrze). W kursie nie korzysta się z klastra.

Ograniczenia arkusza kalkulacyjnego

Excel — nie tylko za mały; bywa wręcz **niebezpieczny** dla danych biologicznych.

- Nazwy genów zamieniane są na daty: SEPT2 $\longrightarrow$ „2 września”, MARCH1 $\longrightarrow$ data
- Identyfikatory tracą zera wiodące, długie liczby zaokrąglane są do notacji wykładniczej
- Zmiany są **ciche** — plik zapisuje się bez ostrzeżenia, a błąd wychodzi dużo później

Udokumentowany problem

Badanie z 2016 roku wykazało, że około **jedna piąta** publikacji genetycznych z załącznikami w Excelu zawierała tak przekłamane nazwy genów. Skala problemu skłoniła później do zmiany części symboli genów.

Dzień pracy: od surowych danych do wyniku

Typowa analiza — **ciąg kroków**; wynik jednego narzędzia jest wejściem następnego.

1. Kontrola jakości surowych odczytów (`FastQC`)
2. Przycięcie odczytów słabej jakości (`fastp`, `cutadapt`)
3. Dopasowanie (mapowanie) odczytów do genomu odniesienia (`bwa`, `STAR`)
4. Wykrycie wariantów (`bcftools`) albo zliczenie odczytów na genach — zależnie od typu analizy
5. Analiza statystyczna i wykresy (R)

Dlaczego to ważne dla kursu

Każdy z tych kroków uruchamia się z wiersza poleceń, a całość spina się w jeden powtarzalny **pipeline**. Takie łączenie narzędzi ćwiczy się przez cały semestr.

Narzędzia tylko w wierszu poleceń

Sztandarowe programy bioinformatyczne — **wyłącznie** z wiersza poleceń.

Narzędzie	Zastosowanie
`bwa`, `bowtie2`, `STAR`	dopasowanie (mapowanie) odczytów do genomu
`samtools`, `bcftools`	obróbka dopasowań i wariantów
`bedtools`	operacje na interwałach genomowych
`salmon`, `kallisto`	kwantyfikacja ekspresji
`seqkit`, `fastp`	obróbka plików FASTA/FASTQ

Nauka wiersza poleceń to warunek wstępny korzystania z tych programów — nie ma dla nich „wersji z oknami”.

2. Terminal i powłoka (przypomnienie)

Geneza: Unix i Linux

- **Unix** powstał na przełomie lat 60. i 70. w Bell Labs jako system dla programistów
- Zbudowano go wokół prostej idei: małe narzędzia, wspólny format tekstowy, łączenie w potoki
- **Linux** (lata 90.) to darmowa, otwarta implementacja tych samych zasad; dziś działa na serwerach i klastrach obliczeniowych, w tym na wszystkich superkomputerach z listy TOP500
- macOS również wywodzi się z Uniksa, więc jego terminal działa niemal identycznie

Dlaczego to nadal aktualne

Zasady sprzed pół wieku okazały się trwałe: te same polecenia działają na laptopie, na serwerze i na superkomputerze. Nauka opłaca się na całą karierę.

Trzy pojęcia na początek

Terminal

Okno do wpisywania tekstu i czytania odpowiedzi. Sam w sobie nic nie liczy.

Powłoka (*shell*)

Program, który odczytuje wpisane polecenia i je wykonuje. Najczęściej `bash`.

Polecenie (*command*)

Konkretna instrukcja, np. „wypisz zawartość katalogu” albo „policz wiersze”.

> Znak zachęty (*prompt*), zwykle `$`, oznacza, że powłoka czeka na polecenie. Polecenie wpisuje się i zatwierdza klawiszem Enter.

Powłoki: bash, sh, zsh

Powłoka to zwykły program — jest ich kilka; różnią się drobiazgami składni, nie ideą.

Powłoka	Uwagi
`sh`	pierwotna, minimalna; wzorzec zgodności
`bash`	domyślna w większości dystrybucji Linuksa; w niej pisze się skrypty kursu
`zsh`	domyślna w macOS (od 2019 r.); w kilku miejscach działa inaczej niż `bash`

3. Instalacja i konfiguracja środowiska

Wspólne środowisko pracy

Na starcie — **jedno spójne środowisko** dla wszystkich, niezależnie od systemu na własnym komputerze.

- Instrukcje instalacji podaje **Moduł 0** na Moodle; ten wykład przedstawia ich przegląd
- Konfigurację wykonuje się samodzielnie, **raz**, na początku semestru
- Formy pomocy przy problemach opisuje ostatni slajd tej części

Cel

Działający terminal uniksowy, menedżer pakietów Conda ze środowiskiem kursu i edytor kodu — ten sam zestaw, na którym oparte są wszystkie moduły.

Trzy elementy warsztatu

Element	Do czego
Terminal uniksowy	Miejsce wpisywania poleceń (powłoka `bash`)
Conda (Miniforge)	Instaluje narzędzia i izoluje środowisko kursu (w nim także Git — od Modułu 6)
Edytor (VS Code)	Pisanie i podgląd skryptów oraz plików tekstowych

Terminal — *gdzie* się pracuje, Conda — *skąd* pochodzą narzędzia, edytor — *czym* pisze się skrypty. Te trzy elementy wystarczą na cały kurs.

Który system operacyjny?

Punkt startu zależy od systemu; cel wspólny — działający **bash** i **conda**.

System	Otwieranie terminala	Kolejne kroki
Linux	aplikacja „Terminal” albo skrót `Ctrl+Alt+T`	od razu instalacja Condy
macOS	Terminal (Programy → Narzędzia) albo Spotlight	od razu instalacja Condy
Windows	terminal Ubuntu z menu Start (po instalacji WSL)	najpierw **WSL** (podsystem Linuksa), potem Conda w Ubuntu

Windows: nie PowerShell, nie CMD

Polecenia kursu są uniksowe, a PowerShell i wiersz poleceń CMD mają inne polecenia. Na Windows pracuje się w terminalu Ubuntu z WSL — w środowisku Linux takim samym jak na serwerach.

Windows: WSL, czyli Linux w Windowsie

WSL (Windows Subsystem for Linux) — pełny Linux wewnątrz Windowsa, bez ręcznie konfigurowanej maszyny wirtualnej i bez osobnego rozruchu.

- Prawdziwy terminal z `bash` i menedżerem pakietów Ubuntu
- Pliki Windows i Linux są wzajemnie dostępne
- WSL 2 sam uruchamia lekką, zarządzaną maszynę wirtualną — stąd wymóg włączonej wirtualizacji
- Zalecany sposób pracy na Windows — ten sam, którego używa się zawodowo

> Po instalacji WSL polecenia z obu wykładów i z modułów działają tak samo jak na Linuksie.

Windows: instalacja WSL

Menu Start → **PowerShell jako administrator**, jedno polecenie:

```
wsl --install -d Ubuntu
```

1. Polecenie instaluje WSL z dystrybucją **Ubuntu**
2. Po instalacji **restart** komputera
3. Przy pierwszym uruchomieniu Ubuntu zakłada się **nazwę użytkownika i hasło** (osobne od konta Windows)
4. Od tej pory terminal Ubuntu otwiera się z menu Start

> PowerShell jest potrzebny tylko w tym jednym momencie — do zainstalowania WSL. Dalsza praca odbywa się w terminalu Ubuntu.

macOS: terminal i narzędzia wiersza poleceń

macOS wywodzi się z Uniksa — terminal jest **wbudowany** (aplikacja *Terminal*; iTerm2 to osobno instalowana alternatywa).

- Domyślna powłoka to `zsh`, a narzędzia systemowe pochodzą z rodziny BSD (częściowo inne opcje niż GNU)
- Bash 5 i narzędzia GNU dostarcza środowisko Conda; ani Homebrew, ani `xcode-select --install` nie są potrzebne

Uwaga na starą wersję bash

Systemowy `bash` w macOS ma wersję 3.2 (z 2006 r.) i nie obsługuje części nowszej składni. Przykłady z modułów wykonuje się w bashu ze środowiska: po `conda activate pracownia` wpisuje się `bash`, a polecenie `exit` wraca do `zsh`.

Linux: niemal gotowe środowisko

Terminal i menedżer pakietów — część systemu; zostaje tylko doinstalować Condę.

- Menedżer pakietów zależy od dystrybucji: `apt` (Ubuntu/Debian), `dnf` (Fedora)
- Aktualizacja systemu: `sudo apt update && sudo apt upgrade`
- `update` odświeża tylko listę dostępnych pakietów; nowsze wersje instaluje `upgrade`

> Na Linuksie droga jest najkrótsza: od razu instalacja Condy (Miniforge). To samo dotyczy Ubuntu w WSL.

Conda: menedżer pakietów i środowisk

Conda — instaluje narzędzia bioinformatyczne i trzyma je w izolowanych środowiskach.

- Instaluje pakiety bez uprawnień administratora i bez ręcznego kompilowania
- Każdy projekt może mieć własne środowisko z własnymi wersjami narzędzi
- Środowisko kursu powstaje z jednego pliku, więc wszyscy instalują ten sam zestaw narzędzi — to podstawa reprodukowalności

Miniforge

W kursie: wariant Miniforge — lekki, darmowy, z otwartym kanałem `conda-forge` jako domyślnym.
Szerzej o Condzie — w Module 6.

Instalacja Miniforge

Instalator: `github.com/conda-forge/miniforge` (Linux, także w WSL, i macOS), uruchamiany w terminalu:

```
bash Miniforge3-$(uname)-$(uname -m).sh
```

1. Instalator prowadzi przez kolejne kroki (akceptacja licencji, katalog instalacji)
2. Na koniec należy zgodzić się na inicjalizację powłoki
3. Terminal trzeba **zamknąć i otworzyć ponownie**, żeby zmiany zadziałały
4. Po ponownym otwarciu na początku znaku zachęty pojawia się `(base)`

> Napis `(base)` oznacza, że Conda działa, a aktywne jest jej środowisko bazowe.

Środowisko kursu

Osobne środowisko dla kursu, zamiast środowiska bazowego — z pliku `environment.yml` z Modułu 0:

```
conda env create -f environment.yml
conda activate pracownia
```

- `conda env create -f environment.yml` — tworzy środowisko `pracownia` z kompletem narzędzi całego kursu
- `conda activate pracownia` — wejście do środowiska; znak zachęty pokazuje wtedy `(pracownia)`
- Jeśli środowisko `pracownia` już istnieje, uzupełnia się je poleceniem `conda env update -n pracownia -f environment.yml`

Reguła praktyczna

Przed pracą nad modułem warto sprawdzić, czy w znaku zachęty widać `(pracownia)`. Jeśli go nie widać, środowisko aktywuje się poleceniem `conda activate pracownia`.

VS Code — edytor kodu

- Visual Studio Code — darmowy, standard w pracy zawodowej
- Składnia, podgląd plików i wbudowany terminal w jednym oknie
- Windows: instalacja w systemie Windows i rozszerzenie WSL — praca wprost na plikach w Linuksie
- Skrypty z uniksowymi końcami wierszy LF (pasek stanu); przy CRLF bash zgłasza `$'\r': command not found`
- Rozszerzenia Python i R — druga połowa kursu

Weryfikacja instalacji

Sprawdzenie, czy narzędzia odpowiadają — w aktywnym środowisku `pracownia`:

```
bash --version      # powinno pokazać wersję 5.x
conda --version     # np. conda 26.x
git --version       # np. git version 2.5x
python --version    # Python 3.11.x
```

- Każde polecenie powinno wypisać **numer wersji**, a nie komunikat o błędzie
- `command not found` oznacza, że dany element nie jest jeszcze zainstalowany albo aktywny

Najczęstsze problemy instalacyjne

Objaw	Zwykła przyczyna
`wsl` nie działa	wyłączona wirtualizacja w BIOS lub brak aktualizacji Windows
`conda: command not found`	nie otwarto terminala ponownie po instalacji
brak `(base)` w wierszu	powłoka nie została zainicjalizowana
`bash` wciąż 3.2 (macOS)	środowisko `pracownia` nie jest aktywne
`prefix already exists`	środowisko `pracownia` już istnieje — `conda env update`

> Większość problemów rozwiązuje ponowne otwarcie terminala i sprawdzenie, czy środowisko jest aktywne.

Pomoc przy instalacji

Problemy z konfiguracją zdarzają się często — ważne, żeby rozwiązać je, **zanim** zaczną się moduły.

- **Moduł 0 na Moodle** — instrukcje instalacji dla każdego systemu
- **Spotkania wprowadzające** — pytania o problemy z konfiguracją
- **Forum przedmiotu** — opis: system, polecenie i pełna treść błędu
- **Konsultacje** — z własnym laptopem, gdy problemu nie udało się rozwiązać zdalnie

Reguła praktyczna

Konfigurację środowiska warto zrobić w pierwszym tygodniu. Problem instalacyjny wykryty na dwa dni przed terminem Modułu 1 trudno rozwiązać na czas.

4. Budowa polecenia i praca w terminalu (przypomnienie)

Pierwsze polecenia

Kilka nieszkodliwych poleceń do oswojenia się z terminalem — każde coś wypisuje:

```
whoami        # nazwa zalogowanego użytkownika
date          # bieżąca data i godzina
echo "Witaj"  # wypisz podany tekst
clear         # wyczyść ekran terminala
```

- Żadne z nich niczego nie zmienia w systemie — służą do oswojenia się z wpisywaniem poleceń
- `echo` przyda się później do wypisywania wartości i budowania komunikatów w skryptach
- `clear` czyści zapełniony ekran

Anatomia znaku zachęty

Sam znak zachęty niesie informacje, jeszcze przed wpisaniem polecenia:

```
bartek@laptop:~/dane/fasta$
```

Fragment	Znaczenie
`bartek`	nazwa użytkownika
`laptop`	nazwa komputera (własny czy zdalny)
`~/dane/fasta`	katalog roboczy (~ to katalog domowy)
`$`	koniec znaku zachęty; dalej wpisuje się polecenie

Rzut oka na znak zachęty pokazuje użytkownika, katalog i maszynę — warto wyrobić nawyk jego czytania.

Anatomia polecenia

Polecenie ma stały wzorzec: **nazwa**, potem **opcje**, potem **argumenty**.

```
grep -c ">" probka_C.fasta
```

Część	Przykład	Rola
Polecenie	`grep`	co zrobić (tu: wyszukaj)
Opcja	`-c`	jak zmodyfikować działanie (tu: policz pasujące wiersze)
Argument	`">"`	na czym operować (tu: wzorzec do znalezienia)
Argument	`probka_C.fasta`	wejście (tu: plik)

Pomoc do dowolnego polecenia: `man grep` albo `grep --help`.

Opcje krótkie i długie

Wiele opcji ma dwie równoważne postacie — krótką (jedna litera) i długą (czytelną nazwę).

Krótka	Długa	Działanie (`grep`)
`-c`	`--count`	policz pasujące wiersze
`-i`	`--ignore-case`	ignoruj wielkość liter
`-v`	`--invert-match`	pokaż wiersze *nie*pasujące
`-n`	`--line-number`	dopisz numery wierszy

- Krótkie opcje można łączyć: `grep -in` to to samo co `grep -i -n`
- Długie są czytelniejsze w skryptach — od razu widać, co robią

Pomoc w terminalu

Nie trzeba pamiętać wszystkich opcji — trzeba wiedzieć, gdzie je sprawdzić.

Sposób	Do czego
`man polecenie`	pełny podręcznik (wyjście klawiszem `q`)
`polecenie --help`	szybkie streszczenie opcji
`tldr polecenie`	praktyczne przykłady użycia (po instalacji: `conda install tldr`)
explainshell.com	rozkłada dowolne polecenie na części

Reguła praktyczna

Przed skopiowaniem polecenia z internetu warto sprawdzić w `man` albo na explainshell, co dokładnie robią jego opcje. To odróżnia rozumienie od zgadywania.

Skróty klawiaturowe i historia poleceń

Skrót	Działanie
`Tab`	dokończ nazwę pliku lub polecenia po kilku pierwszych literach
↑ / ↓	przewijaj wcześniejsze polecenia z historii
`Ctrl+R`	przeszukaj historię poleceń po fragmencie
`Ctrl+C`	przerwij działające polecenie
`Ctrl+A` / `Ctrl+E`	skocz na początek / koniec wiersza
`Ctrl+L`	wyczyść ekran

Reguła praktyczna

`Tab` i strzałka w górę oszczędzają przepisywania i chronią przed literówkami w długich nazwach plików.

5. Praca z plikami i katalogami (przypomnienie)

System plików i ścieżki

Pliki tworzą drzewo katalogów; plik wskazuje się jego **ścieżką**.

- **Katalog roboczy** — bieżące położenie w drzewie; pokazuje je polecenie `pwd`
- **Ścieżka bezwzględna** zaczyna się od `/` — pełny adres od korzenia, np. `/home/ala/dane/probka.fasta`
- **Ścieżka względna** liczona jest od katalogu roboczego, np. `fasta/probka.fasta`

Skróty warte zapamiętania

~ to katalog domowy, `.` to katalog bieżący, `..` to katalog nadrzędny (o poziom wyżej).

Poruszanie się po katalogach

```
pwd              # pokaż katalog roboczy
ls -lah          # wypisz zawartość
cd fasta         # wejdź do katalogu fasta
cd ..            # cofnij się o poziom wyżej
```

- `pwd` — pokaż katalog roboczy (ang. *print working directory*)
- `ls` — wypisz zawartość: `-l` szczegóły, `-a` też pliki ukryte, `-h` czytelne rozmiary
- `cd` — wejdź do katalogu; `cd ..` cofa o poziom, samo `cd` wraca do katalogu domowego

Reguła praktyczna

Przed uruchomieniem polecenia na plikach warto sprawdzić `pwd` i `ls` — czy bieżący katalog jest właściwy i czy pliki mają spodziewane nazwy.

Podglądanie zawartości plików

Polecenie	Do czego
`cat plik`	wypisz całą zawartość
`less plik`	przeglądaj stronami (wyjście klawiszem `q`)
`head plik`	pierwsze 10 wierszy (`head -n 3` — trzy)
`tail plik`	ostatnie 10 wierszy (`tail -n +2` — od drugiego)
`wc -l plik`	policz wiersze

> Wielkie pliki (gigabajty) ogląda się przez `less` albo `head`, a nie przez `cat`, który wypisałby na ekran setki tysięcy wierszy.

Tworzenie i porządkowanie

Polecenie	Działanie
`mkdir wyniki`	utwórz katalog
`touch plik.txt`	utwórz pusty plik (lub odśwież datę)
`cp zrodlo cel`	skopiuj plik
`cp -r kat1 kat2`	skopiuj cały katalog
`mv stara nowa`	przenieś lub zmień nazwę

`mv` służy zarówno do przenoszenia, jak i do zmiany nazwy — to ta sama operacja.

Usuwanie plików — ostrożność

```
rm plik.txt
rm -r stary_katalog
```

- `rm` usuwa plik
- `rm -r` usuwa katalog z zawartością

Nie ma kosza

W terminalu `rm` kasuje nieodwracalnie — plik nie trafia do kosza. Szczególnie groźne jest `rm -rf` z błędną ścieżką.

Reguła praktyczna

Przed usunięciem warto wypisać to samo poleceniem `ls`, żeby zobaczyć, co dokładnie zniknie. Dwukrotne sprawdzenie zajmuje mniej czasu niż odtwarzanie utraconych danych.

Znaki wieloznaczne (globbing)

Powłoka potrafi dopasować wiele plików naraz według wzorca — bez wypisywania każdego z osobna.

Wzorzec	Dopasowuje
`*.fasta`	wszystkie pliki z rozszerzeniem `.fasta`
`probka_?.txt`	`probka_A.txt`, `probka_B.txt`, …(jeden znak)
`probka_[AB].txt`	tylko `probka_A.txt` i `probka_B.txt`
`dane/*`	cała zawartość katalogu `dane`

```
ls *.fasta          # wypisz wszystkie pliki FASTA
wc -l probka_*.txt  # policz wiersze w kilku plikach naraz
```

Cudzysłowy i spacje w nazwach

Powłoka rozdziela polecenie spacjami, więc spacja w nazwie pliku bywa źródłem błędów.

Problem

```
cat moja probka.txt
```

Powłoka widzi *dwa* argumenty: `moja` i `probka.txt`.

Rozwiązanie

```
cat "moja probka.txt"
```

Cudzysłów scala nazwę w jeden argument.

Reguła praktyczna

Najprościej nie używać spacji w nazwach plików roboczych — zamiast nich podkreślenie (`moja_probka.txt`). Oszczędza to kłopotów w całym kursie.

Uprawnienia w skrócie

Każdy plik ma zapisane, kto może go czytać, zmieniać i uruchamiać. Pokazuje to `ls -l`.

```
-rwxr-xr-- 1 ala grupa 2048 paź 13 17:54 analiza.sh
```

- Pierwszy znak to typ: `-` zwykły plik, `d` katalog
- Dalej trzy trójki praw — właściciela, grupy i pozostałych: `r` — odczyt, `w` — zapis, `x` — uruchamianie
- Żeby uruchomić własny skrypt, trzeba nadać mu prawo wykonywania:

```
chmod +x analiza.sh
```

Pełniej uprawnienia (także zapis ósemkowy) omawia Moduł 1; w skryptach (Moduł 3) przydaje się `chmod +x`.

Szukanie plików: find

Gdy położenie pliku jest nieznane, `find` przeszukuje drzewo katalogów według kryteriów.

```
find . -name "*.fastq"        # pliki .fastq w bieżącym katalogu i niżej
find dane -type d             # tylko katalogi wewnątrz dane/
```

- Pierwszy argument to miejsce startu (`.` — katalog bieżący)
- `-name` dopasowuje nazwę (wzorzec w cudzysłowie), `-type f` pliki, `-type d` katalogi
- Szuka rekurencyjnie, więc zagląda też do podkatalogów

Reguła praktyczna

`find` dobrze łączy się z potokiem: listę znalezionych plików można przekazać dalej i przetworzyć wszystkie naraz.

Ile miejsca zajmują dane: du i df

Dane sekwencyjne bywają wielkie — warto sprawdzać zajętość dysku, zanim skończy się miejsce.

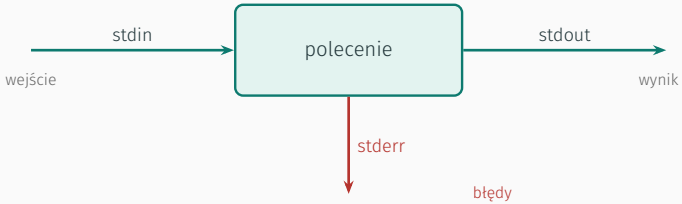
Polecenie	Do czego
`du -sh katalog`	rozmiar katalogu w czytelnej formie
`du -sh *`	rozmiar każdego elementu bieżącego katalogu
`df -h`	wolne miejsce na zamontowanych dyskach

Opcja `-h` (*human-readable*) pokazuje rozmiary jako `4,0K`, `2,3G` zamiast liczby bajtów — dużo czytelniej przy dużych plikach.

6. Filozofia Uniksa: potoki i filtry

Trzy strumienie każdego polecenia

Polecenie czyta z jednego wejścia i pisze do dwóch osobnych wyjść.



- **stdout** i **stderr** to dwa *różne* wyjścia — wynik osobno, komunikaty o błędach osobno
- Dzięki temu można zapisać wynik do pliku, a błędy nadal widzieć na ekranie

Przekierowania: kierowanie strumieni

Zapis	Znaczenie
`polecenie > plik`	zapisz wynik do pliku (nadpisuje)
`polecenie >> plik`	dopisz wynik na koniec pliku
`polecenie < plik`	podaj plik jako wejście
`polecenie 2> bledy.log`	przekieruj komunikaty błędów

```
grep ">" probka_C.fasta > naglowki.txt
```

Uwaga

Pojedyncze > nadpisuje plik bez pytania. Do dopisania na końcu pliku służy >>.

Małe narzędzia, jeden cel

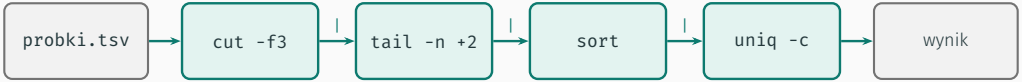
Filozofia Uniksa

Każde narzędzie robi jedną rzecz dobrze. Narzędzia porozumiewają się tekstem. Złożone zadania powstają przez łączenie małych narzędzi, nie przez budowę jednego wielkiego programu.

- Zamiast jednego programu „do wszystkiego” — kilkadziesiąt małych, wyspecjalizowanych narzędzi
- Wspólnym interfejsem jest zwykły **tekst** — wyjście jednego narzędzia jest wejściem następnego
- Ten sam zestaw narzędzi działa na pliku z tabelą, z sekwencjami i z wariantami

Potok: łączenie narzędzi znakiem |

Znak **potoku** (`|`) kieruje wynik jednego polecenia wprost na wejście następnego, bez pliku pośredniego.



Każdy człon dostaje strumień tekstu, przetwarza go i przekazuje dalej. Tak z prostych elementów powstaje złożone przetwarzanie.

Potoki a duże dane

Potok nie zapisuje plików pośrednich między krokami — dane **płyną** przez kolejne narzędzia.

- Większość filtrów (`grep`, `cut`, `wc`) przetwarza strumień **wiersz po wierszu** i od razu podaje dalej
- Dzięki temu plik większy niż pamięć komputera i tak da się przetworzyć
- **Wyjątek:** `sort` musi zobaczyć całe wejście, zanim wypisze pierwszy wiersz; przy dużych danych zapisuje pliki tymczasowe (domyślnie w `$TMPDIR` lub `/tmp`; zmiana: `-T`)

Reguła praktyczna

To dlatego jeden potok potrafi policzyć odczyty w wielogigabajtowym FASTQ, choć Excel nie otworzyłby nawet ułamka takiego pliku.

Klasyczne filtry tekstowe

Narzędzie	Do czego
`grep`	znajdź wiersze pasujące do wzorca
`cut`	wytnij wybrane kolumny
`sort`	posortuj wiersze (`-n` liczbowo, `-k` po kolumnie)
`uniq`	scal powtórzenia (`-c` zlicz)
`tr`	zamień lub usuń znaki
`wc`	policz wiersze, słowa, znaki

Z tych elementów w Module 1 buduje się wieloetapowe potoki.

Potok: liczba próbek na grupę

Tabela `probki.tsv` ma kolumny `id`, `plik`, `grupa`. Liczba próbek w każdej grupie:

```
cut -f3 probki.tsv | tail -n +2 | sort | uniq -c
```

Wynik:

```
      2 kontrola
      2 stres
```

- `cut -f3` — wytnij trzecią kolumnę → `tail -n +2` — pomiń nagłówek
- `sort` — uporządkuj → `uniq -c` — policz powtórzenia

grep i wzorce

`grep` wyszukuje wiersze pasujące do wzorca. Wzorce mogą być dokładne albo opisane wyrażeniem regularnym.

```
grep "^>" probka_C.fasta
```

```
>geneC1
>geneC2
>geneC3
>geneC4
```

- `^>` to wzorzec: znak `>` na początku wiersza (kotwica `^`)
- W FASTA nagłówki zaczynają się od `>`, więc to prosty sposób, by je wszystkie wypisać
- `grep -c "^>"` zamiast wypisywać — policzy sekwencje w pliku

Sortowanie po kolumnie

`sort` domyślnie porządkuje alfabetycznie, całymi wierszami. Opcje pozwalają sortować liczbowo i po wybranej kolumnie.

```
sort -t$'\t' -k3,3n dane.tsv    # po 3. kolumnie, liczbowo
```

- `-t$'\t'` — ustaw tabulator jako separator kolumn
- `-k3,3n` — klucz: tylko kolumna 3, porównywana liczbowo (`n`); samo `-k3` to klucz od kolumny 3 do końca wiersza
- `-r` — odwróć kolejność (malejąco)

Reguła praktyczna

Brak sortowania liczbowego (`n`) to częsta pomyłka: bez niego `sort` porównuje znak po znaku i `10` wypada przed `2`.

Przykład: najczęstsze chromosomy w VCF

Które chromosomy mają najwięcej wariantów? Odpowiedź daje potok złożony z poznanych już elementów:

```
grep -v "^#" warianty.vcf | cut -f1 | sort | uniq -c | sort -rn
```

- `grep -v "^#"` — odrzuć wiersze nagłówka VCF
- `cut -f1` — weź pierwszą kolumnę (chromosom) → `sort | uniq -c` — policz
- `sort -rn` — ustaw malejąco, żeby najliczniejsze były na górze

sed i awk — zapowiedź modułu 2

sed

Edytor strumieniowy. Podmienia, usuwa i drukuje wybrane wiersze — przydatny do zamian tekstu w locie.

awk

Mały język do danych w kolumnach. Filtruje, liczy i formatuje wiersz po wierszu; rozpoznaje pola `$1`, `$2`, …

Przykład: średnia długość z kolumny 2 (bez wiersza nagłówka)

```
awk -F'\t' 'NR>1 { suma += $2; n++ } END { print suma/n }' dane.tsv
```

Szczegóły `awk` i `sed` — w Module 2.

Łączenie poleceń i kody wyjścia

Każde polecenie po zakończeniu zwraca **kod wyjścia**: `0` oznacza sukces, wartość różną od zera — błąd. Powłoka pozwala uzależnić kolejne kroki od tego kodu.

Zapis	Znaczenie
`A ; B`	wykonaj A, potem B (niezależnie od wyniku A)
`A && B`	wykonaj B **tylko jeśli** A zakończyło się sukcesem
`A || B`	wykonaj B **tylko jeśli** A zawiodło

Przykład

`fastqc probka.fastq && echo "gotowe"` — komunikat pojawi się dopiero, gdy analiza się powiedzie.

Więcej mocy potoku: tee i xargs

`tee`

Rozgałęzia strumień: zapisuje wynik do pliku *i* przekazuje go dalej potokiem. Przydatne do zachowania kroku pośredniego.

`xargs`

Buduje polecenia z listy na wejściu — pozwala uruchomić to samo narzędzie na wielu plikach naraz.

`tee` — w Module 1; `xargs`, także z równoległym uruchamianiem (`-P`) — w Module 4.

7. Dane i formaty w bioinformatyce

Przewaga formatów tekstowych

Ogromna większość formatów w bioinformatyce to zwykły **tekst**, a nie zamknięty format binarny.

- **Czytelność** — plik można obejrzeć `head`-em i od razu zrozumieć strukturę
- **Uniwersalność** — te same narzędzia (`grep`, `cut`, `awk`) działają na każdym z nich
- **Strumieniowość** — plik czyta się kawałek po kawałku, więc rozmiar nie jest przeszkodą

> Dlatego warsztat z wiersza poleceń przenosi się wprost na dane biologiczne: umiejętność przetwarzania zwykłej tabeli wystarcza też do pracy z VCF czy GFF.

FASTA — sekwencje

Najprostszy format sekwencji. Każdy rekord to wiersz nagłówka (zaczyna się od >) i sekwencja pod nim.

```
>geneC1
ATGCTAGCTAGCTTACGATCGATCGTAGCTAGC
>geneC2
ATGGGCTTAAGCTAGCATCGATCG
```

- Przechowuje DNA, RNA albo białka
- Nagłówek to identyfikator (i często opis) sekwencji
- Policzenie sekwencji to po prostu policzenie wierszy z >

FASTQ — sekwencje z jakością

Format surowych odczytów z sekwenatora. Każdy odczyt to **cztery** wiersze.

```
@odczyt_001
ATGCTAGCTAGCTTACGATCG
+
IIIIIIIIIHHHHGGGGFFFE
```

- Wiersz 1: identyfikator (zaczyna się od `@`)
- Wiersz 2: sekwencja Wiersz 3: separator `+`
- Wiersz 4: **jakość** każdej zasady, zakodowana znakami (skala Phred)

> Skoro odczyt zajmuje cztery wiersze, liczba odczytów to liczba wierszy podzielona przez cztery.

Przykład: liczba odczytów w FASTQ

Skoro jeden odczyt to cztery wiersze, liczbę odczytów daje jeden potok — także dla pliku spakowanego:

```
zcat probka.fastq.gz | wc -l        # liczba wierszy
```

- `zcat` wypisuje zawartość pliku `.gz` bez rozpakowywania go na dysk
- `wc -l` liczy wiersze; liczbę odczytów daje podzielenie wyniku przez **cztery**
- Ten sam plik może mieć dziesiątki milionów odczytów, a potok i tak czyta go strumieniowo

Reguła praktyczna

To typowa *kontrola jakości na wejściu*: przed rozpoczęciem analizy warto sprawdzić, ile odczytów faktycznie zawiera każda próbka.

Formaty tabelaryczne — krótki przegląd

Format	Zawartość
TSV/CSV	zwykłe tabele (kolumny rozdzielone tabulatorem/przecinkiem)
GFF/GTF	anotacja genomu: gdzie leżą geny, egzony, cechy
BED	interwały genomowe: chromosom, początek, koniec
VCF	warianty: pozycje różnic względem genomu odniesienia
SAM/BAM	odczyty dopasowane do genomu (BAM to binarna wersja SAM)

Formaty i narzędzia do nich (`samtools`, `bcftools`, `bedtools`) omawia Moduł 5.

VCF z bliska — warianty

Format opisujący **różnice** próbki względem genomu odniesienia. Zaczyna się od wierszy nagłówka (`##`), potem następuje tabela.

```
#CHROM  POS    ID   REF  ALT  QUAL  FILTER  INFO
chr1    10583  .    G    A    52.4  PASS    DP=31
chr1    12783  .    A    G    38.1  PASS    DP=18
```

- Jeden wiersz to jeden wariant: gdzie (`CHROM`, `POS`) i jaka zmiana (`REF`→`ALT`)
- Osiem kolumn obowiązkowych; `INFO` to adnotacje wariantu, np. `DP` — głębokość pokrycia
- To wciąż tabela tekstowa, więc `grep` i `awk` działają tu tak samo jak na zwykłym TSV
- Policzenie wariantów na chromosomie to ten sam potok, co liczenie próbek w grupie

GFF/GTF — anotacja genomu

Format opisujący, gdzie na genomie leżą cechy: geny, egzony, sekwencje regulatorowe. Dziewięć kolumn rozdzielonych tabulatorem.

```
chr1  havana  gene   11869  14409  .  +  .  gene_id "ENSG..."
chr1  havana  exon   11869  12227  .  +  .  gene_id "ENSG..."
```

- Kluczowe kolumny: chromosom, typ cechy, początek, koniec, nić (+/−)
- Współrzędne liczone od 1, przedział obustronnie domknięty
- Zliczenie genów to policzenie wierszy, w których trzecia kolumna to `gene`

BED — interwały genomowe

Minimalistyczny format do opisu **fragmentów genomu**. W najprostszej postaci trzy kolumny.

```
chr1    1000    5000
chr1    8000    9500
chr2    200     1750
```

- Kolumny: **chromosom**, **początek**, **koniec** (opcjonalnie nazwa, wynik, nić)
- Uwaga: współrzędne liczone **od 0**, koniec wyłączony — inaczej niż w GFF
- Narzędzie `bedtools` liczy na tych interwałach części wspólne, pokrycia i sąsiedztwa

Typowa pułapka

BED liczy od zera, GFF od jedynki. Pomylenie konwencji przesuwa wynik o jedną pozycję, a taki błąd łatwo przeoczyć, bo nie wywołuje żadnego komunikatu.

SAM/BAM — dopasowania do genomu

Wynik **dopasowania (mapowania) odczytów** do genomu odniesienia, ang. *alignment*. Każdy wiersz to jeden odczyt i miejsce, w które trafił.

- **SAM** to wersja tekstowa; **BAM** to jej skompresowana, binarna postać (mniejsza i szybsza)
- BAM jest binarny, więc nie ogląda się go wprost — do podglądu i filtrowania służy `samtools`
- Zawiera m.in. nazwę odczytu, chromosom, pozycję i jakość mapowania

Reguła praktyczna

`samtools view plik.bam | head` zamienia binarny BAM na czytelny tekst i pokazuje pierwsze dopasowania — dalej działają już zwykłe filtry tekstowe.

Pliki skompresowane: .gz

Dane sekwencyjne bywają ogromne, więc trzyma się je spakowane — najczęściej z rozszerzeniem `.gz` (np. `probka.fastq.gz`).

Polecenie	Działanie
`gzip -k plik`	spakuj (tworzy `plik.gz`; `-k` zachowuje oryginał)
`gunzip -k plik.gz`	rozpakuj (`-k` zachowuje plik `.gz`)
`zcat plik.gz`	wypisz zawartość bez rozpakowywania
`zless`, `zgrep`	wersje `less`/`grep` dla plików `.gz`

Reguła praktyczna

Bez `-k` polecenia `gzip` i `gunzip` zastępują plik wejściowy — przy surowych danych to ryzyko. Do podglądu wystarczy `zcat probka.fastq.gz | head`, bez rozpakowywania na dysk.

Systemowy `zcat` w macOS nie czyta plików `.gz`; w środowisku `pracownia` działa (poza nim: `gzcat` lub `gunzip -c`).

Wspólny warsztat dla różnych formatów

Reguła praktyczna

Skoro większość formatów to tekst w kolumnach, ten sam warsztat — potoki, `grep`, `cut`, `awk`, skrypty — obsługuje je wszystkie. Nauka narzędzi opłaca się raz i procentuje przy każdym formacie.

- Policzenie sekwencji w FASTA, wariantów per chromosom w VCF czy cech w GFF to warianty *tej samej* operacji
- Dlatego kurs zaczyna się od fundamentu (M1–M4), a wyspecjalizowane narzędzia przychodzą później

8. Dobre praktyki i typowe pułapki

Typowe pułapki początkującego

- Spacje w nazwach plików — powłoka rozdziela argumenty spacją; zamiast nich **podkreślenie**
- Ślepe `rm -rf` — brak kosza, kasuje nieodwracalnie; najpierw `ls`
- Pojedyncze `>` zamiast `>>` — nadpisuje plik bez ostrzeżenia
- Praca w złym katalogu — polecenie działa w innym katalogu niż zamierzony; `pwd` rozwiewa wątpliwość
- Puste zmienne w ścieżkach — `rm -rf "$KAT"/*` przy pustym `KAT` usuwa zawartość katalogu głównego; ochrona: `"${KAT:?}"` lub `set -u`

> Wypadki zwykle biorą się z pośpiechu, nie z braku wiedzy. Chwila sprawdzenia przed groźnym poleceniem to zawodowy nawyk.

Dobre nawyki od pierwszego dnia

- Sprawdzenie przed działaniem — `ls` i `pwd` przed uruchomieniem czegokolwiek na plikach
- Budowanie potoku stopniowo — człony dokładane po jednym, z kontrolą wyniku po każdym
- Uzupełnianie Tab i historia poleceń — mniej literówek, szybsza praca
- Proste nazewnictwo — bez spacji, bez polskich znaków, z rozszerzeniem mówiącym o formacie
- Zapisywanie działających poleceń — zalążek przyszłego skryptu i dokumentacji analizy

Organizacja katalogu projektu

Prosty, powtarzalny układ — uporządkowany projekt oszczędza godzin szukania:

```
projekt/
├── dane/        # surowe dane (tylko do odczytu)
├── wyniki/      # to, co wyprodukowały analizy
├── skrypty/     # polecenia i skrypty
└── README.md    # co jest czym i jak uruchomić
```

- Surowe dane tylko do odczytu — wszystko, co powstaje z ich przetwarzania, trafia do `wyniki/`
- `README.md` — opis projektu: skąd dane, jak powstał wynik

Nazewnictwo plików

Dobra nazwa mówi, co jest w środku, i dobrze współpracuje z powłoką.

Dobrze

```
probka_01_kontrola.fastq.gz
liczba_wariantow_chr1.tsv
2026-03-01_analiza_ekspresji.R
```

Źle

```
Próbka 1 (kopia).txt
nowy plik FINAL v2.csv
wyniki???.dat
```

Reguła praktyczna

Bez spacji i polskich znaków, konsekwentne rozszerzenia, daty w formacie `RRRR-MM-DD` (wtedy sortują się chronologicznie).

Zestawienie poleceń z wykładu

Orientacja i pliki

- `pwd`, `ls`, `cd` — położenie, zawartość, zmiana katalogu
- `cat`, `less`, `head`, `tail` — podgląd
- `mkdir`, `cp`, `mv`, `rm` — porządki
- `find`, `du`, `df` — szukanie, miejsce

Potoki i filtry

- `grep` — szukaj wierszy
- `cut` — wytnij kolumny
- `sort`, `uniq -c` — porządkuj, zliczaj
- `wc -l` — policz wiersze
- `|`, `>`, `>>` — łącz i zapisuj

Opcje i przykłady większości tych poleceń — w materiałach Modułu 1 na Moodle.

Podsumowanie

Praca w ciągu semestru

- **Środowisko z Modułu 0** — skonfigurowane przed Modułem 1 (terminal, Conda, edytor)
- **Praktyka** — samo czytanie nie wystarczy; każde polecenie trzeba samodzielnie uruchomić
- **Podstawy bez pomijania** — Moduły 1–4 to fundament, na którym opiera się reszta
- **Błąd to informacja** — najpierw warto przeczytać komunikat; `--help` i `man` są pod ręką
- **Pytania na bieżąco** — na konsultacjach i forum kursu; lepiej dopytać, niż długo tkwić w problemie
- **Notatki z działających poleceń** — z czasem staną się podręcznym zestawieniem na całe studia

> Narzędzia z tego kursu zostają na całą karierę: te same potoki obsłużą pracę magisterską, doktorat i realne projekty badawcze.

Do zapamiętania

- Bioinformatyka to praca z danymi na dużą skalę — **wiersz poleceń** daje reprodukowalność, automatyzację i dostęp do serwerów
- Polecenie ma stały wzorzec: **nazwa, opcje, argumenty**; czyta ze `stdin`, pisze na `stdout` i `stderr`
- Filozofia Uniksa: **małe narzędzia** łączone **potokiem** (`|`) w złożone przetwarzanie
- Większość formatów biologicznych to **tekst**, więc jeden warsztat obsługuje je wszystkie

Na następny wykład

Wykład 2: Reprodukowalność i nowoczesny warsztat

- Reprodukowalność analizy i skrypty bash
- Kontrola wersji (Git), środowiska (Conda) i kontenery (Docker)
- Python i narzędzia wiersza poleceń, pipeline'y Snakemake, wykresy i raporty w R i Quarto
- Klaster obliczeniowy i SLURM — przegląd

Dziękuję za uwagę

dr inż. Bartosz Czech · bartosz.czech@upwr.edu.pl