Домашнее задание 4
Построение пайплайна получения генетических вариантов

1. https://www.ncbi.nlm.nih.gov/sra/SRX9196450 - результат секвенирования (набор ридов)
2. Скрипт на bash для получения результатов samtools flagstat - script1.sh
   Скрипт для разбора результатов samtools flagstat и получения % (алгоритм оценки качества картирования) - script2.sh
   Скрипт для freebayes - script3.sh
3. Результат команды flagstat - output\flagstat_result
4. У меня был фреймворк GNU Make, на линукс системах и wsl он уже предустановлен, поэтому специальных инструкций не требуется. Но вот на всякий случай:
   a. Чтобы скачать и установить GNU Make в Debian (А я всё делала в виртуальной машине Debian, но эта инструкция подойдет для всех Linux систем), нужно:
      i. Открыть терминал и обновить список пакетов с помощью команды *sudo apt-get update*.
      ii. Установить GNU Make с помощью команды *sudo apt-get install make*. Это загрузит и установит пакет GNU Make и его зависимости.
      iii. Проверить успешную установку с помощью команды *make --version*. Она выведет версию GNU Make, установленную в системе.
      iv. Также команда make содержится в метапакете build-essential, для его установки нужно выполнить команду *sudo apt-get install build-essential*.
      v. После установки рекомендуется перезагрузить компьютер, чтобы изменения вступили в силу.
   b. GNU make прост в использовании, необходим для автоматизации сборки различных приложений, в том числе для написания пайлпайнов. Его основной файл - *Makefile*, а запускается он с помощью команды *make*.

c. Можно запустить тестовый HelloWorld с помощью команды *make -f hello_makefile* (-f указывает, какой конкретно файл нужно загрузить)
d. Есть команда *make -n -f hello_makefile*, которая выводит список инструкций без их выполнения

5. Результаты работы пайплайна и лог-файлы содержатся в папке output.
6. Код пайплайна содержится в файле Makefile.
7. Сначала выполняется команда *make clean*, затем *make,* а для визуализации - *make visualize*
8. Визуализация в файле output/pipeline.png

Для визуализации пайплайна в GNU Make можно использовать несколько подходов, в зависимости от того, какой именно инструмент или фреймворк вы хотите использовать для визуализации. GNU Make сам по себе не предоставляет встроенных инструментов для визуализации, однако существуют различные инструменты, которые можно интегрировать с Make, чтобы генерировать графическое представление зависимостей в виде диаграмм.
Я использовала make с Graphviz
Graphviz — это популярный инструмент для визуализации графов, который можно использовать для генерации диаграмм зависимостей из файлов Make.
Для того чтобы интегрировать его с GNU Make, можно использовать директиву .NOTPARALLEL и make-параметры, чтобы экспортировать информацию о зависимостях в формате, понятном Graphviz.

В Makefile добавила правило для генерации .dot файла, который будет использоваться для визуализации.

```
.PHONY: visualize

visualize: pipeline.dot
   dot -Tpng -o pipeline.png pipeline.dot

pipeline.dot: Makefile
   echo "digraph G {" > pipeline.dot
      awk '/^[^#][^ \t]*:/ {print "\""$1"\" -> \"" $$2 "\";"}' $(MAKEFILE_LIST) >> pipeline.dot
   echo "}" >> pipeline.dot
```

Команда make visualize генерирует файл pipeline.png с диаграммой зависимостей.

1. Использованный способ визуализации:
Для визуализации пайплайна с использованием GNU Make, используется инструмент Graphviz для генерации графа зависимостей в формате .dot. Этот инструмент визуализирует пайплайн как Directed Acyclic Graph (DAG), где:

- Вершины (узлы) представляют различные задачи или шаги в пайплайне.
- Ребра (стрелки) показывают зависимости между задачами, то есть какой шаг должен быть выполнен до другого.

В процессе работы с Makefile, создается граф, который отображает:

- Зависимости между целями (targets) и их предшествующими шагами.
- Как одна задача зависит от выполнения предыдущей.

Преимущества:

1. Является естественным представлением сложных пайплайнов с множеством шагов и зависимостей.
2. Подходит для параллельной обработки, так как можно запускать независимые задачи одновременно.
3. Прост в масштабировании: легко добавлять новые шаги и зависимости.

Блок-схема алгоритма:
Блок-схема — это графическое представление алгоритма, в котором:

- Блоки представляют различные этапы вычислений или действий.
- Ребра соединяют блоки, указывая на порядок выполнения или логику перехода между блоками (например, ветвления).

Использование в алгоритмах: Блок-схема используется для визуализации последовательности шагов алгоритма или процесса, например, для описания бизнес-логики или вычислительного алгоритма.

Преимущества:

1. Хорошо подходит для отображения последовательности операций в алгоритмах с условиями и циклами.
2. Легко воспринимается человеком, так как отображает логику выполнения алгоритма.