Тестовый пайплайн. 18S последовательности

1. Создание бласт БД

```
mkdir -p blast_dbs

for f in ~/Amphipodas/test_18s/*.fasta; do
    base=$(basename "$f" .fasta)
    makeblastdb -in "$f" -blastdb_version 5 -dbtype nucl -hash_index -out "blast_dbs/$base"
done
```

2. Выравнивание на референс

```
mkdir -p blast_results

 for db in blast_dbs/*.nsq; do     base=$(basename "$db" .nsq);     blastn -query
18S_reference.fasta -db "blast_dbs/$base" -out
"blast_results/${base}_vs_18S_reference.txt" -evalue 1e-5 -outfmt 6 -max_target_seqs 10;
done
```

3. Взятие 5 лучших хитов по e-value с помощью python кода ("test_pipeline_step3.py")

```
import os
import pandas as pd

input_dir = "blast_results"
output_dir = "best_hits"

os.makedirs(output_dir, exist_ok=True)

for file in os.listdir(input_dir):
    if file.endswith(".txt"):
        input_path = os.path.join(input_dir, file)
        output_path = os.path.join(output_dir, f"{file.replace('.txt', '_top5.txt')}")

        with open(input_path, "r") as f:
            lines = [line.strip().split("\t") for line in f]

        # сортировка по e-value и взятие 5 первых
        lines_sorted = sorted(lines, key=lambda x: float(x[10]))
        top5 = lines_sorted[:5]

        with open(output_path, "w") as f:
            for line in top5:
                f.write("\t".join(line) + "\n")
```

4. Создание фаста файлов с 5 лучшими последовательностями (для дальнейшего поиска в случае, если 1ая последовательность не подойдет) (“test_pipeline_step4.py”)

```
import os
from Bio import SeqIO

best_hits_dir = "./best_hits"
fasta_dir = "/media/tertiary/transcriptome_assemblies/rnaspades_reassemblies"
output_dir = "./5_hits_seqs_TEST"

os.makedirs(output_dir, exist_ok=True)

hit_files = [f for f in os.listdir(best_hits_dir) if f.endswith("_vs_18S_reference_top5.txt")]

for hit_file in hit_files:
    species_name = hit_file[:-26]  # название вида из названия файла
    fasta_filename = species_name + ".fasta"
    fasta_path = os.path.join(fasta_dir, fasta_filename)
```

```
    if not os.path.exists(fasta_path):
        print(f"Файл {fasta_path} не найден, пропускаем...")
        continue

    # 5 лучших хитов из файла
    best_hits_path = os.path.join(best_hits_dir, hit_file)
    with open(best_hits_path, "r") as f:
        best_hits = [line.strip().split("\t")[1] for line in f.readlines()]  # названия контигов

    output_fasta_path = os.path.join(output_dir, f"{species_name}_18S_top5.fasta")

    # извлечение последовательности
    with open(output_fasta_path, "w") as out_f:
        for record in SeqIO.parse(fasta_path, "fasta"):
            if record.id in best_hits:
                out_f.write(f">Naumenko_18S_{species_name}_{record.id}\n{str(record.seq)}\n")

    print(f"Сохранено в файле {output_fasta_path}")
```

5. Взятие 1ой лучшей последовательности из фасты с 5 лучшими (“test_pipeline_step5.py”)

```
import os
from Bio import SeqIO

folder_path = './5_hits_seqs'
folder_res = './test_1_best_hit'
os.makedirs(folder_res, exist_ok=True)

def get_best_hit(file_path):
    with open(file_path, "r") as handle:
        for record in SeqIO.parse(handle, "fasta"):
            # лучший хит —первый контиг
            return f">{record.id}\n{record.seq}"
    return None

def process_all_files(folder_path):
    files_in_folder = os.listdir(folder_path)
    fasta_files = [file for file in files_in_folder if file.endswith('.fasta')]

    for fasta_file in fasta_files:
        file_path = os.path.join(folder_path, fasta_file)
        best_hit = get_best_hit(file_path)  # первый контиг
        if best_hit:
            new_file_name = f"{os.path.splitext(fasta_file)[0][:-5]}_best.fasta"
            new_file_path = os.path.join(folder_res, new_file_name)
```

```
        with open(new_file_path, "w") as new_handle:
            new_handle.write(best_hit)

process_all_files(folder_path)
```

6. Вариант с mafft+trimal + Blast check identity + выравнивание PairwiseAligner (from Bio.Align) для финальной проверки идентичности не менее 90%. (“test_pipeline_step6.py”)

mafft + trim

```
import os
from Bio import SeqIO


INPUT_DIR = "./test_1_best_hit"
REFERENCE = "18S_reference.fasta"
OUTPUT_DIR = "./processed_results"
MAFFT = "mafft"
TRIMAL = "trimal"

os.makedirs(OUTPUT_DIR, exist_ok=True)

ref_length = len(next(SeqIO.parse(REFERENCE, "fasta")).seq)
print(f"Reference length: {ref_length} bp")

for filename in os.listdir(INPUT_DIR):
    if not filename.endswith("_18S_best.fasta"):
        continue

    species_name = filename.replace("_18S_best.fasta", "")
    input_file = os.path.join(INPUT_DIR, filename)
    output_prefix = os.path.join(OUTPUT_DIR, species_name)

    print(f"\nОбработка {species_name}...")

    combined_file = f"{output_prefix}_combined.fasta"  # временный файл с
объединенными последовательностями
    with open(combined_file, "w") as f:
        # 1й - референс, потом последовательность вида
        ref_seq = next(SeqIO.parse(REFERENCE, "fasta"))
        query_seq = next(SeqIO.parse(input_file, "fasta"))
        SeqIO.write([ref_seq, query_seq], f, "fasta")
```

```
    # mafft
    aligned_file = f"{output_prefix}_aligned.fasta"
    exit_code = os.system(f"{MAFFT} --auto {combined_file} > {aligned_file}")

    if exit_code != 0:
        print(f"ERROR: MAFFT failed for {filename}")
        os.remove(combined_file)
        continue

    # trimal
    trimmed_file = f"{output_prefix}_trimmed.fasta"
    exit_code = os.system(f"{TRIMAL} -in {aligned_file} -out {trimmed_file} -nogaps")

    if exit_code != 0:
        print(f"ERROR: trimal failed for {filename}")
        os.remove(combined_file)
        continue

    if not os.path.exists(trimmed_file):
        print(f"ERROR: Trimmed file not created for {filename}")
        os.remove(combined_file)
        continue

    # проверка длины
    try:
        trimmed_seqs = list(SeqIO.parse(trimmed_file, "fasta"))
        if len(trimmed_seqs) != 2:
            print(f"ERROR: Expected 2 sequences after trimming, got
{len(trimmed_seqs)}")
            os.remove(combined_file)
            continue

        trimmed_length = len(trimmed_seqs[1].seq)  # длина последовательности
вида

        if abs(trimmed_length - ref_length) > 0.1 * ref_length:  # допуск ±10%
            print(f"ERROR: {filename} - bad length after trimming")
            print(f"Trimmed length: {trimmed_length}, Reference: {ref_length}")
        else:
            with open(f"{output_prefix}_processed.fasta", "w") as out:
                SeqIO.write(trimmed_seqs[1], out, "fasta")
                print(f"Saved processed sequence (length: {trimmed_length})")

    except Exception as e:
        print(f"ERROR processing {filename}: {str(e)}")
```

```
    os.remove(combined_file)

print("\nProcessing complete!")
```

7. **DB check** (“test_pipeline_step7.py”)

```
import os
from Bio import SeqIO

INPUT_DIR = "./processed_results"  # для обработанных последовательностей
REFERENCE = "18S_reference.fasta"
OUTPUT_DIR = "./final_results"
BLASTDB_DIR = "./blast_reference_db"
MIN_IDENTITY = 90.0

os.makedirs(OUTPUT_DIR, exist_ok=True)
os.makedirs(BLASTDB_DIR, exist_ok=True)

# BLAST бд  из референса
print("Создаём BLAST базу данных...")
base_name = os.path.splitext(os.path.basename(REFERENCE))[0]
db_command = (
    f"makeblastdb -in {REFERENCE} -blastdb_version 5 "
    f"-dbtype nucl -hash_index -out {BLASTDB_DIR}/{base_name}"
)
os.system(db_command)

print("\nПроверка последовательности...")
for filename in os.listdir(INPUT_DIR):
    if not filename.endswith("_processed.fasta"):
        continue

    input_file = os.path.join(INPUT_DIR, filename)
    output_file = os.path.join(OUTPUT_DIR, filename.replace("_processed", "_final"))

    # BLAST
    blast_output = "temp_blast_result.txt"
    os.system(
        f"blastn -query {input_file} -db {BLASTDB_DIR}/{base_name} "
        f"-out {blast_output} -outfmt '6 pident' -max_target_seqs 1"
    )

    try:
        with open(blast_output) as f:
            identity = float(f.readline().split()[0])
```

```
            print(f"{filename}: идентичность {identity:.1f}%", end=" ")

            if identity >= MIN_IDENTITY:
                with open(input_file) as src, open(output_file, "w") as dst:
                    dst.write(src.read())
                print("Сохранено")
            else:
                print("Слишком низкая идентичность")

        except Exception as e:
            print(f"{filename}: ошибка при проверке ({str(e)})")

        if os.path.exists(blast_output):
            os.remove(blast_output)

    print("\nПроверка закончена!")
```

–сбор прошедших проверку последовательностей в один фаста-файл–

8. **final check** (“test_pipeline_step8.py”)

```
import os
from Bio import SeqIO
from Bio.Align import PairwiseAligner

def load_single_sequence(fasta_file):
    return next(SeqIO.parse(fasta_file, "fasta")).seq

def calculate_identity(seq1, seq2):
    """% идентичности между двумя последовательностями"""
    aligner = PairwiseAligner()
    aligner.mode = 'global'
    alignment = aligner.align(seq1, seq2)[0]

    aln_seq1 = alignment.aligned[0]
    aln_seq2 = alignment.aligned[1]

    matches = 0
    for (start1, end1), (start2, end2) in zip(aln_seq1, aln_seq2):
        for i in range(min(end1 - start1, end2 - start2)):
            if seq1[start1 + i] == seq2[start2 + i]:
                matches += 1

    identity = matches / max(len(seq1), len(seq2)) * 100  # % идентичности
    return identity, alignment
```

```
def process_fasta(input_file, reference_seq, min_identity=90.0):
    sequences_to_save = []  # подходящие последовательности

    for seq_record in SeqIO.parse(input_file, "fasta"):
        seq = seq_record.seq
        identity, alignment = calculate_identity(seq, reference_seq)
        print(f"{seq_record.id}: Идентичность {identity:.1f}%")

        if identity >= min_identity:
            print(f"  - Сохранено (Идентичность выше {min_identity}%)")
            sequences_to_save.append(seq_record)
        else:
            print(f"  - Слишком низкая идентичность (< {min_identity}%)")

    if sequences_to_save:
        with open("filtered_sequences.fasta", "w") as output_file:
            SeqIO.write(sequences_to_save, output_file, "fasta")
        print(f"\nСохранено {len(sequences_to_save)} последовательностей в файл
'filtered_sequences.fasta'.")

# Основной процесс
def main():
    input_fasta = "sequences.fasta"
    reference_fasta = "18S_reference.fasta"
    reference_seq = load_single_sequence(reference_fasta)

    # Обрабатываем файл с последовательностями
    process_fasta(input_fasta, reference_seq, min_identity=90.0)

if __name__ == "__main__":
    main()
```

9. Вариант с UGENE. Итоги и % идентичности референсу

Итого прошли проверки 25 последовательностей для rnaspades и 18 trinity, после удаления последовательностей с отсутствующими координатами и выбора лучшей последовательности для тех, которые прошли проверку в обеих сборках, итоговый файл содержал 26 последовательностей.

Было обнаружено, что последовательности стали похожи на референс и при бласте в NCBI многие из них давали лучший хит с референсной последовательностью.

В связи с этим было решено провести множественное выравнивание контигов с лучшими хитами для этих 26 последовательностей и референсом, с обрезанием по границам референса с помощью UGENE.

Сравнительная таблица идентичности референсу (union_nogaps.fasta и 18S_ugene.fasta):

Sample	Identity_ugene	Identity_nogaps
SRR3467039_Oxyacanthus_sowinskii_18S	99.5	99.5
SRR3467090_Pentagonurus_dawydowi_18S	99.5	99.5
SRR3467037_Oxyacanthus_curtus_18S	99.5	99.5
SRR3467095_Pallaseopsis_kessleri_18S	99.4	99.4
SRR3467063_Eulimnogammarus_cruentus_18S	99.4	99.5
SRR3467052_Sluginella_kietlinskii_18S	99.0	99.0
SRR3467065_Eulimnogammarus_messerschmidtii_18S	98.9	98.9
SRR3467066_Eulimnogammarus_maackii_18S	99.3	99.3
SRR3467055_Eulimnogammarus_violaceus_18S	99.3	99.4
SRR3467067_Eulimnogammarus_sp._18S	99.3	99.3
SRR3467057_Eulimnogammarus_cyaneus_18S	99.3	99.3
SRR3467086_Ommatogammarus_flavus_18S	99.2	99.2
SRR3467048_Carinurus_bicarinatus_18S	99.5	99.5
SRR3467081_Micruropus_parvulus_18S	97.8	97.8
SRR3467077_Macrohectopus_branickii_18S	97.8	97.8
SRR3467083_Micruropus_wahlii_18S	97.5	97.5
SRR3467071_Gmelinoides_fasciatus_18S	97.3	97.4
SRR3467068_Eulimnogammarus_verrucosus_18S	99.0	99.1

SRR3467075_Hyalellopsis_setosa_18S	91.7	91.7
SRR3467073_Hyalellopsis_costata_18S	91.1	91.6
SRR3467089_Homalogammarus_brandtii_18S	99.1	99.2
SRR3467043_Boeckaxelia_carpenterii_18S	99.3	99.3
SRR3467072_Hyalellopsis_carinata_18S	99.1	99.1
SRX1736878_Gammarus_lacustris_18S	97.7	97.8
SRR3467059_Eulimnogammarus_testaceus_18S	94.8	97.0
SRR3467047_Asprogammarus_rhodophthalmus_18S	79.6	94.3

% идентичности референсу рассчитывался с помощью кода (“test_pipeline_step9.py”):

```
from Bio import SeqIO
from Bio.Align import PairwiseAligner

def load_sequence(fasta_file):
    record = next(SeqIO.parse(fasta_file, "fasta"))
    return record.seq

def compute_identity(seq1, seq2):
    aligner = PairwiseAligner()
    aligner.mode = "global"
    alignment = aligner.align(seq1, seq2)[0]

    aligned_seq1 = alignment.aligned[0]
    aligned_seq2 = alignment.aligned[1]

    matches = 0
    for (start1, end1), (start2, end2) in zip(aligned_seq1, aligned_seq2):
        matches += sum(seq1[start1 + i] == seq2[start2 + i] for i in range(min(end1 -
start1, end2 - start2)))

    total_length = max(len(seq1), len(seq2))
    identity = matches / total_length * 100
    return identity

# Пример использования
fasta1 = "18S_reference.fasta"
fasta2 = "fin_file.fasta"
```

```
seq1 = load_sequence(fasta1)
seq2 = load_sequence(fasta2)

identity = compute_identity(seq1, seq2)
print(f"Процент идентичности: {identity:.2f}%")
```