




WBT-BINF2-1.1 Bioinformatyka 2:

Tworzenie skalowalnych potoków analitycznych

Miniconda i Snakemake

Miniconda

 <https://docs.conda.io/en/latest/miniconda.html>

└─ Python 3.9 version



Pandas



komenda 'conda install pkg_name' służy do instalacji pakietów w dystrybucji Anaconda:

```
conda install ipython jupyter numpy pandas matplotlib xlrd
```

czasami należy wskazać również z jakiego kanału (channel, -c) instalujemy dany pakiet

instalacja Snakemake'a:

```
conda install -c conda-forge -c bioconda snakemake
```

Snakemake

Snakemake

*The Snakemake workflow management system is a tool to create
reproducible and scalable data analyses
Workflows are described via a human readable, Python based language*

<https://snakemake.readthedocs.io>

Snakemake: dlaczego jest użyteczny?

Główne zalety Snakemake'a:

- przejrzystość opisu ciągu analitycznego językiem składniowo opartym na języku Python
- możliwość stosowania wstawek w języku Python
- skalowalność: ten sam ciąg analityczny można uruchomić na zwykłym komputerze lub klastrze obliczeniowym
- automatyczne urównoleglanie zadań
- wznowianie analizy w punkcie, w którym ostatnim razem została zakończona
- łączenie ze sobą istniejących, niezależnych narzędzi i własnych skryptów

Snakemake: struktura katalogowa w analizie danych

```
katalog_roboczy/  
├── scripts/  
│   ├── flt.py  
│   └── ...  
├── input/  
│   ├── input_0.in  
│   ├── input_1.in  
│   └── ...  
├── output/  
│   ├── rule_one/  
│   │   ├── first.out  
│   │   └── second.out  
│   ├── ...  
│   ├── rule_two.out  
│   └── rule_three.out  
├── log/  
│   ├── rule_one.log  
│   ├── rule_two.log  
│   └── ...  
├── config.yml  
└── Snakefile
```

⚠ Snakemake samodzielnie tworzy strukturę katalogową konieczną do poprawnego przetworzenia pliku *Snakefile*

Struktura po lewej jest tylko propozycją, niemniej pamiętaj, że porządek ułatwia znacząco życie.

Snakemake: tworzenie plików *Snakefile*

Snakemake: przydatne funkcje

Funkcja	Zastosowanie
<code>glob_wildcards(wyrażenie)</code>	<p>Dopasowuje wyrażenie do nazw plików i zwraca krotkę nazwową (namedtuple) z listami wartości dla znaków zastępczych (wildcards)</p> <pre>seqids, = glob_wildcards('input/{seqid}.fna') many = glob_wildcards('input/{organism}/{seqid}.fna') organisms = many.organism seqids = many.seqid</pre>
<code>expand(</code> <i>wyrażenie,</i> [zip], <i>nazwa=wartości,</i> ..., [allow_missing=True])	<p>Tworzy listę ciągów tekstowych będących efektem uzupełnienia znaków zastępczych o danych nazwach w wyrażeniu podanymi wartościami</p> <pre>expand('output/{organism}/{seqid}.tsv', zip, organism=organisms, seqid=seqids)</pre> <p>Przydatna tam gdzie gdzie zbiegają się niezależne ścieżki analizy (wiele pików wejściowych, jeden wyjściowy)</p>
<code>temp(ścieżka_pliku)</code>	Traktuje określony plik wynikowy jako plik tymczasowy
<code>directory(ścieżka_katalogu)</code>	Traktuje katalog jako plik wynikowy (output)
<code>touch(ścieżka_pliku)</code>	Generuje wynikowy plik zastępczy (dummy output)

Snakemake: bloki opisujące kolejne etapy przetwarzania danych, reguły (rules)

Blok	Funkcja
<code>rule blastn:</code>	Reguła (etap przetwarzania danych). Typem reguły jest <code>checkpoint</code> , używany gdy tworzona jest nieznana liczba plików wynikowych.
<code>params:</code> <code>evaluate = 0.01,</code> <code>db = 'dbs/genomes',</code> <code>cols = 'sseqid evaluate'</code>	Pojedyncza zmienna, krotka (zwykła lub nazwowa, tuple/namedtuple) wartości parametrów uruchomienia
<code>threads:</code> 8	Ilość rdzeni procesora przydzielona do jednego uruchomienia zadania, jeżeli uruchomiane narzędzie działa w trybie wielowątkowym dobrze jest zdefiniować tę wartość
<code>input:</code> <inputcode>'input/query.fna'</inputcode>	Pojedyncza zmienna, krotka (zwykła lub nazwowa) ścieżek do pliku(ów) wejściowego(ych), które istnieją w systemie plików lub są tworzone jako wyjście (output) innej reguły, alternatywnie referencja do funkcji Python
<code>output:</code> <code>'output/blastn.tsv'</code>	Podobnie do input dla pliku(ów) wyjściowych
<code>log:</code> <code>'log/blastn.log'</code>	Podobnie do input dla pliku(ów) log

Snakemake: bloki opisujące kolejne etapy przetwarzania danych, reguły (rules)

Blok	Funkcja
<pre> shell: '''blastn -db {params.db} \ -evalue {params.evalue} \ -outfmt "6 {params.cols}" \ -num_threads {threads} \ -query {input} \ -out {output} \ > {log} 2>&1 ''' </pre>	<p>Polecenie powłoki tekstowej (tutaj Bash), które uruchamia zadanie (istniejące narzędzie lub własny skrypt)</p>
<pre> run: with open(output[0], w) as f: f.write(f'Input {input[0]}\n') f.write(f'E-value {params.evalue}\n') </pre>	<p>Zagnieżdżony kod Python (krótki), dostęp do wartości zapisanych w blokach pliku Snakefile poprzez krotki nazwowe (named tuples) <code>input[0]</code> <code>params.evalue</code></p>
<pre> script: 'scripts/alt_filter.py' </pre>	<p>Uruchomienie skryptu Python z przekazaniem wartości zapisanych w blokach pliku Snakefile w obiekcie <i>snakemake</i> <code>snakemake.input[0]</code></p>

Snakefile: studium przypadku

Snakefile: studium przypadku

```
blastn -evalue 0.01 -query input/gap.fna -db dbs/staph -out output/blastn.tsv \
-outfmt "6 qseqid sseqid qcovs evalve" -num_threads 8
```

```
rule blastn:
    params:
        evalue = 0.01,
        db      = 'dbs/staph',
        cols    = 'qseqid sseqid qcovs evalve'
    threads:
        8
    input:
        'input/gap.fna'
    output:
        'output/blastn.tsv'
    log:
        'log/blastn.log'
    shell:
        '''blastn -db          {params.db}          \
            -evalue          {params.evalue} \
            -outfmt          "6 {params.cols}" \
            -num_threads     {threads}         \
            -query            {input}           \
            -out              {output}          \
            > {log} 2>&1
        '''
```

```
scripts/filter.py --qcovs 100 \
--cols "qseqid sseqid qcovs evalve" \
--input output/blastn_res.tsv \
--output output/filtered.tsv
```

```
rule filter:
    params:
        qcovs = 100,
        cols  = 'qseqid sseqid qcovs evalve'
    input:
        'output/blastn.tsv'
    output:
        'output/filtered.tsv'
    log:
        'log/filter.log'
    shell:
        '''scripts/flt.py --qcovs {params.qcovs} \
            --cols "{params.cols}" \
            --input {input} \
            --output {output} \
            > {log} 2>&1'''
```

Snakefile: studium przypadku

```
rule all:
    input:
        'output/filtered.tsv'
```

```
rule blastn:
    params:
        eval = 0.01,
        db    = 'dbs/staph',
        cols  = 'qseqid sseqid qcovs eval'
    threads:
        8
    input:
        'input/gap.fna'
    output:
        'output/blastn.tsv'
    log:
        'log/blastn.log'
    shell:
        '''blastn -db {params.db} \
                    -eval {params.eval} \
                    -outfmt "6 {params.cols}" \
                    -num_threads {threads} \
                    -query {input} \
                    -out {output} \
                    > {log} 2>&1
        '''
```

⚠ Snakemake odczytując plik Snakefile uruchamia pierwszą regułę a następnie porusza się między regułami zgodnie z istniejącymi między nimi zależnościami.

Zgodnie z konwencją pierwszą regułę nazywa się *all* a jej wejściem jest wyjście ostatniego etapu analizy.

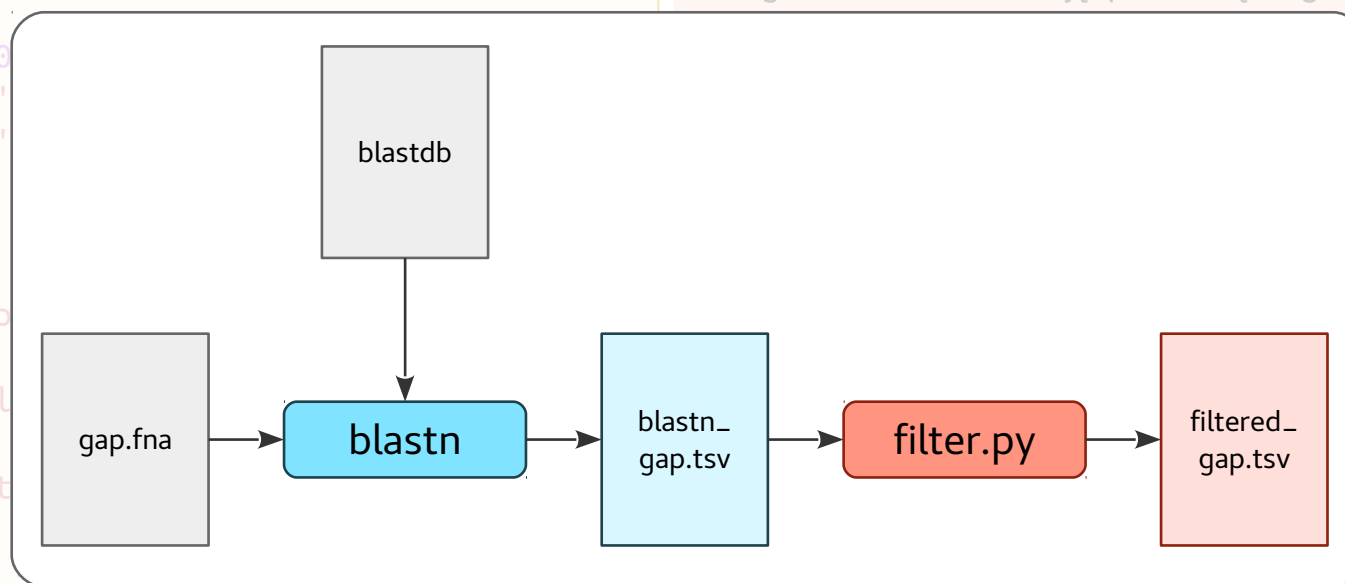
jeżeli reguła blastn jest wcześniej zdefiniowana
rules.blastn.output

```
rule filter:
    params:
        qcovs = 100,
        cols  = 'qseqid sseqid qcovs eval'
    input:
        'output/blastn.tsv'
    output:
        'output/filtered.tsv'
    log:
        'log/filter.log'
    shell:
        '''scripts/flt.py --qcovs {params.qcovs} \
                          --cols  "{params.cols}" \
                          --input  {input} \
                          --output {output} \
                          > {log} 2>&1'''
```

Snakefile: studium przypadku

```
rule all:
    input:
        'output/filtered.tsv'
```

```
rule blastn:
    params:
        evalue = 0.001
        db      = 'blastdb'
        cols    = 6
    threads:
        8
    input:
        'input/gap.fna'
    output:
        'output/blastn_gap.tsv'
    log:
        'log/blastn.log'
    shell:
        '''blastn
```



```
-evalue      {params.evalue} \
-outfmt      "6 {params.cols}" \
-num_threads {threads}       \
-query       {input}          \
-out         {output}         \
> {log} 2>&1
'''
```

⚠ Snakemake odczytując plik Snakefile uruchamia pierwszą regułę a następnie porusza się między regułami zgodnie z istniejącymi między nimi zależnościami.

Zgodnie z konwencją pierwszą regułę nazywa się *all* a jej zadaniem jest zainicjowanie analizy.

...j zdefiniowana

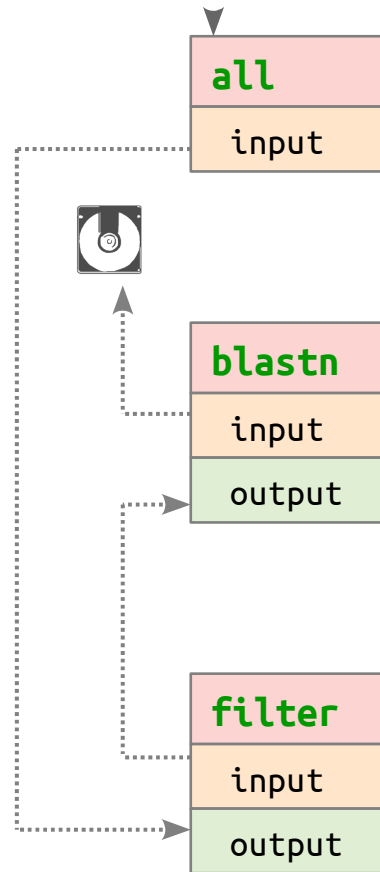
...s evalue'

```

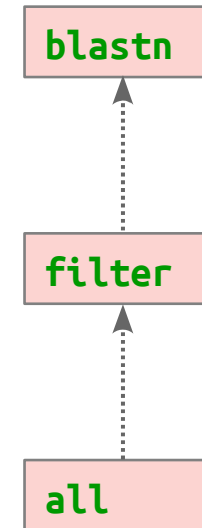
        'log/filter.log'
    shell:
        '''scripts/flt.py --qcovs {params.qcovs} \
                           --cols  "{params.cols}" \
                           --input  {input}         \
                           --output {output}        \
                           > {log} 2>&1'''
```

Snakefile: DAG (directed acyclic graph, skierowany graf acykliczny)

Snakefile

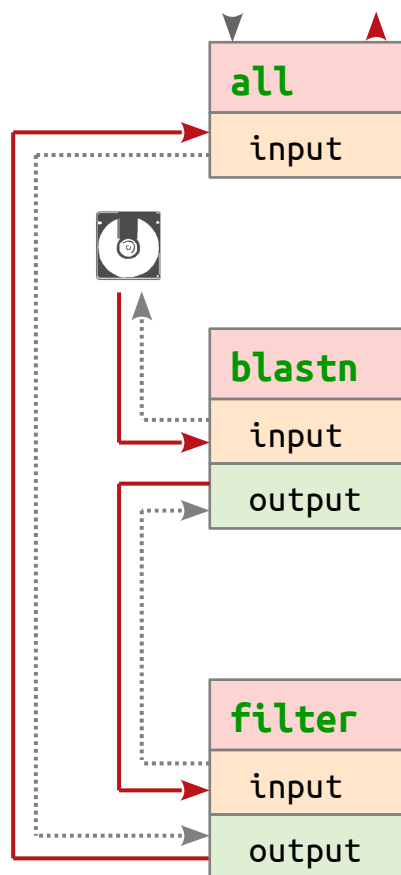


DAG

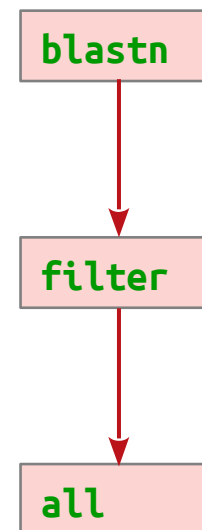


Snakefile: DAG (directed acyclic graph, skierowany graf acykliczny)

Snakefile

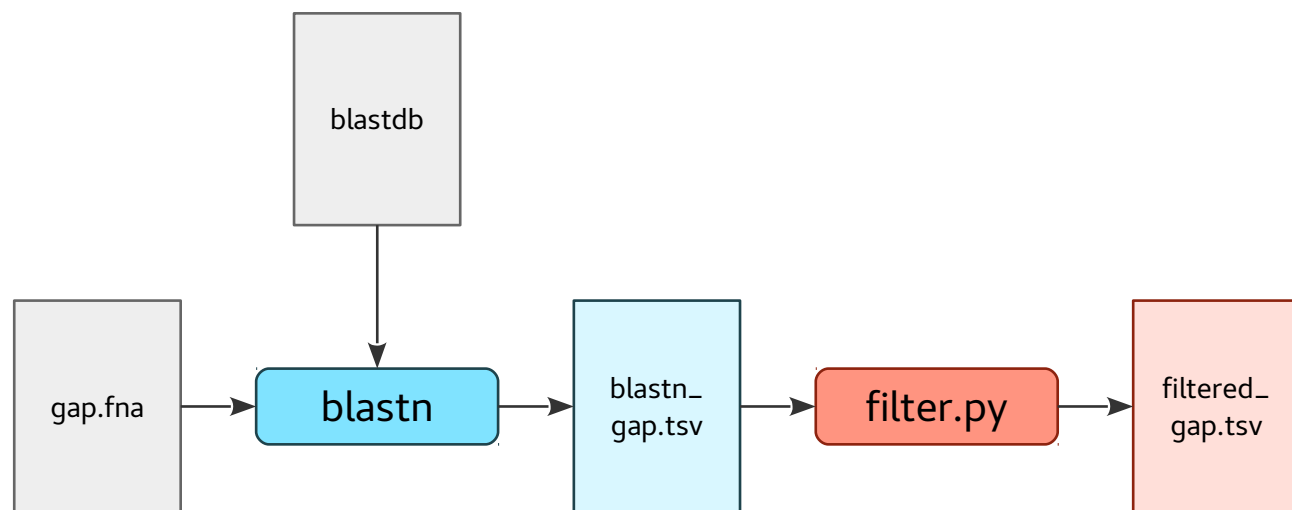


DAG

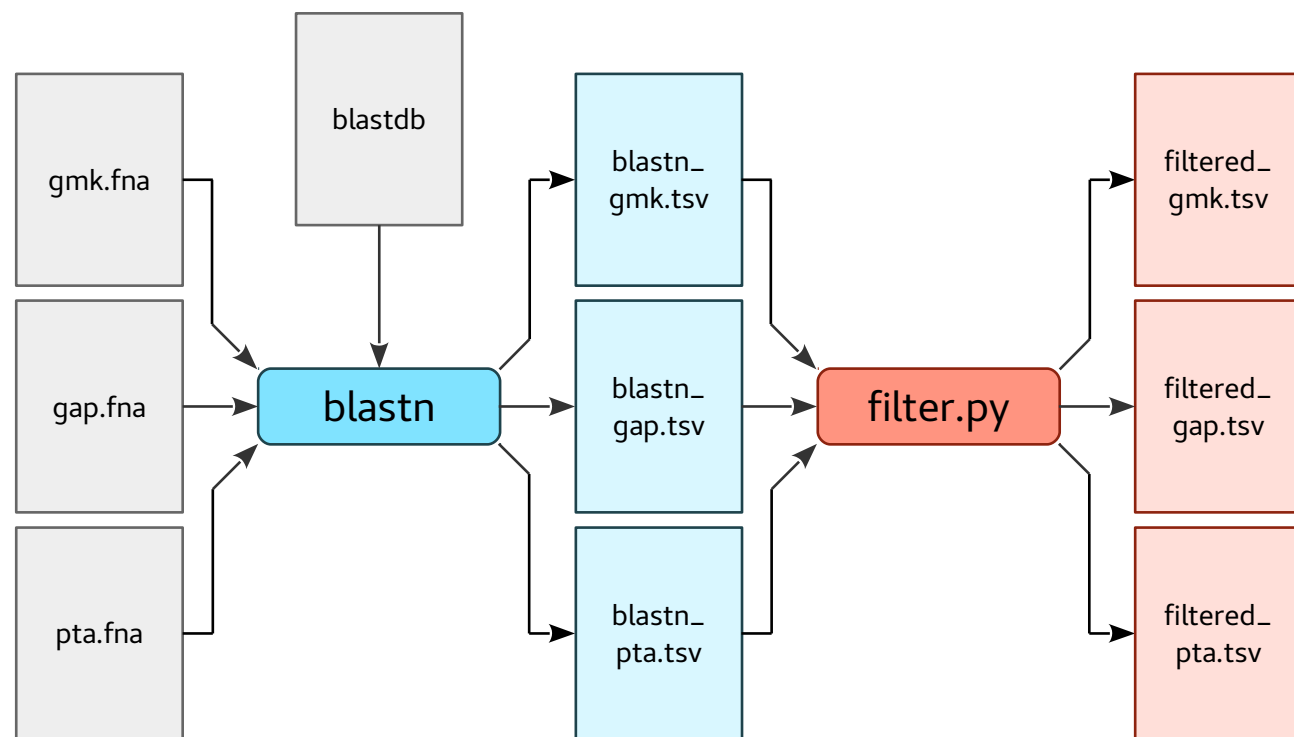


Snakemake: przetwarzanie kolekcji plików

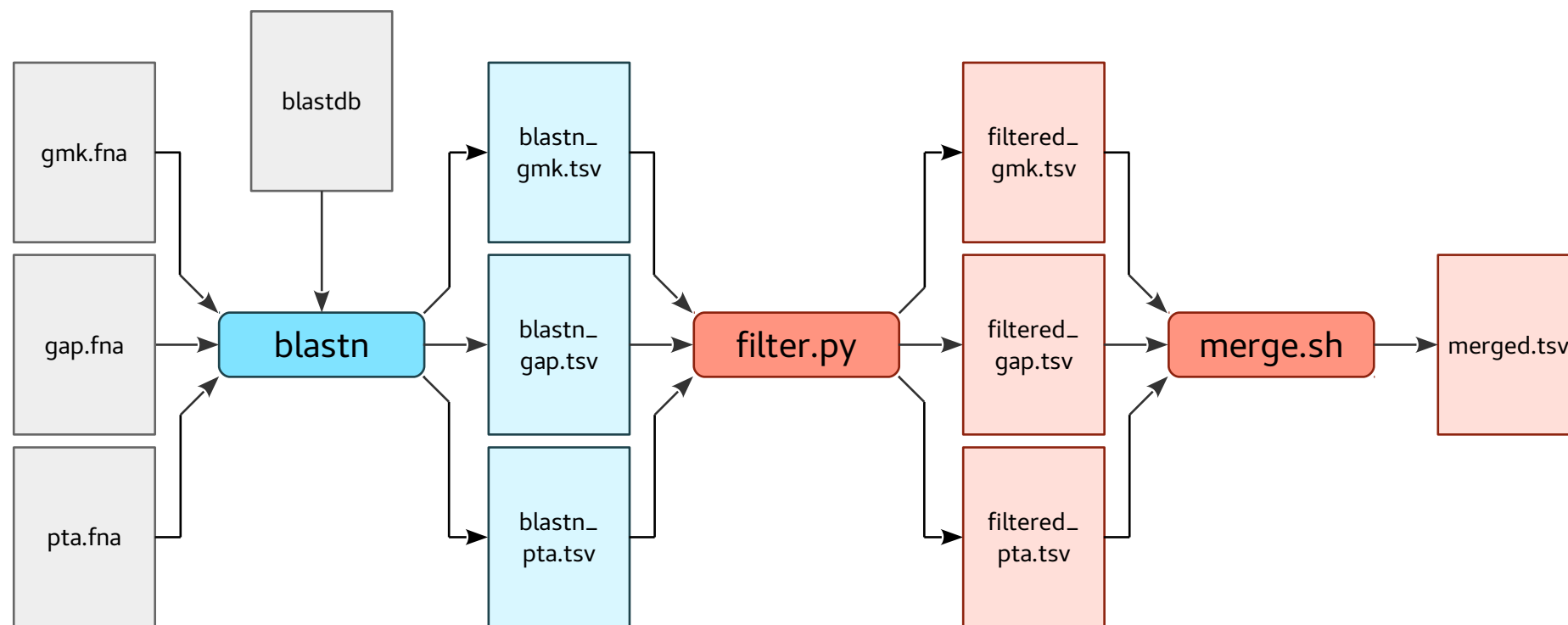
Snakemake: przetwarzanie kolekcji plików



Snakemake: przetwarzanie kolekcji plików



Snakemake: przetwarzanie kolekcji plików



Snakefile: przetwarzanie kolekcji plików

```
seqids, = glob_wildcards('input/{seqid}.fna')

rule all:
    input:
        'output/merged.tsv'

rule blastn:
    params:
        eval = 0.01,
        db    = 'dbs/staph',
        cols  = 'qseqid sseqid qcovs eval'
    input:
        'input/{seqid}.fna'
    output:
        'output/blastn/blastn_{seqid}.tsv'
    log:
        'log/blastn/blastn_{seqid}.log'
    shell:
        '''blastn -db                {params.db} \
            -eval                 {params.eval} \
            -outfmt                "6 {params.cols}" \
            -query                 {input} \
            -out                   {output} \
            > {log} 2>&1

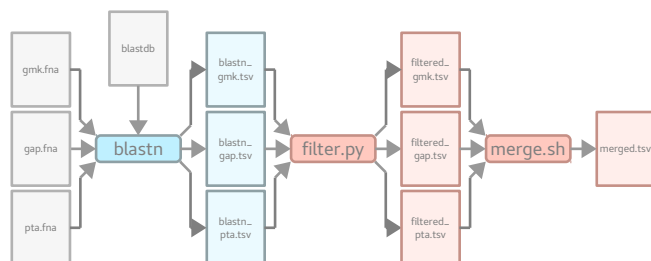
        ...
```

```
rule filter:
    params:
        qcovs = 100,
        cols  = rules.blastn.params.cols
    input:
        rules.blastn.output
    output:
        'output/filter/filtered_{seqid}.tsv'
    log:
        'log/filter/filter_{seqid}.log'
    shell:
        '''scripts/filter.py --qcovs    {params.qcovs} \
                                --cols    "{params.cols}" \
                                --input    {input} \
                                --output   {output} \
                                > {log} 2>&1

        ...

rule merge:
    params:
        cols = rules.blastn.params.cols.replace(' ', '\t'),
        mask  = rules.filter.output[0].replace('{seqid}', '*')
    input:
        expand(rules.filter.output, seqid=seqids)
    output:
        'output/merged.tsv'
    log:
        'log/merge.log'
    shell:
        '''echo "{params.cols}" > {output} 2> {log}
            cat {params.mask} >> {output} 2>> {log}

            ...
```



Snakemake: uruchamianie ciągu analitycznego

Snakemake: uruchamianie ciągu analitycznego

```
# uruchom ciąg analityczny opisany w pliku Snakefile  
# znajdującym się w katalogu roboczym,  
# udostępnił n-rdzeni procesora
```

```
snakemake --cores n
```

```
# uruchom ciąg analityczny podając ścieżkę i nazwę pliku Snakefile
```

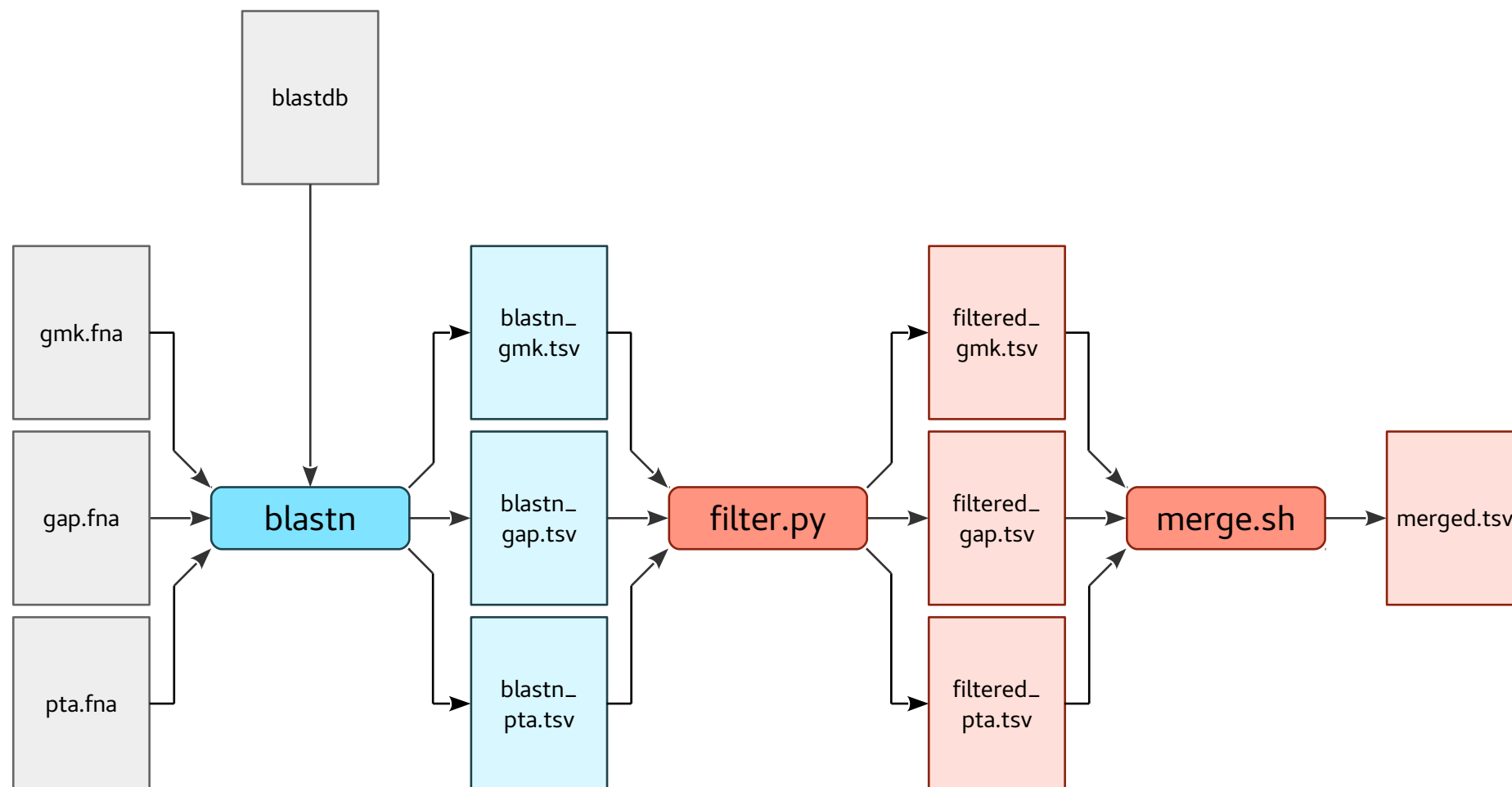
```
snakemake --cores n --snakefile ścieżka_pliku
```

```
# wskaż katalog gdzie Snakemake ma uruchomić ciąg analityczny
```

```
# jeżeli inny niż ten w którym się znajdujesz
```

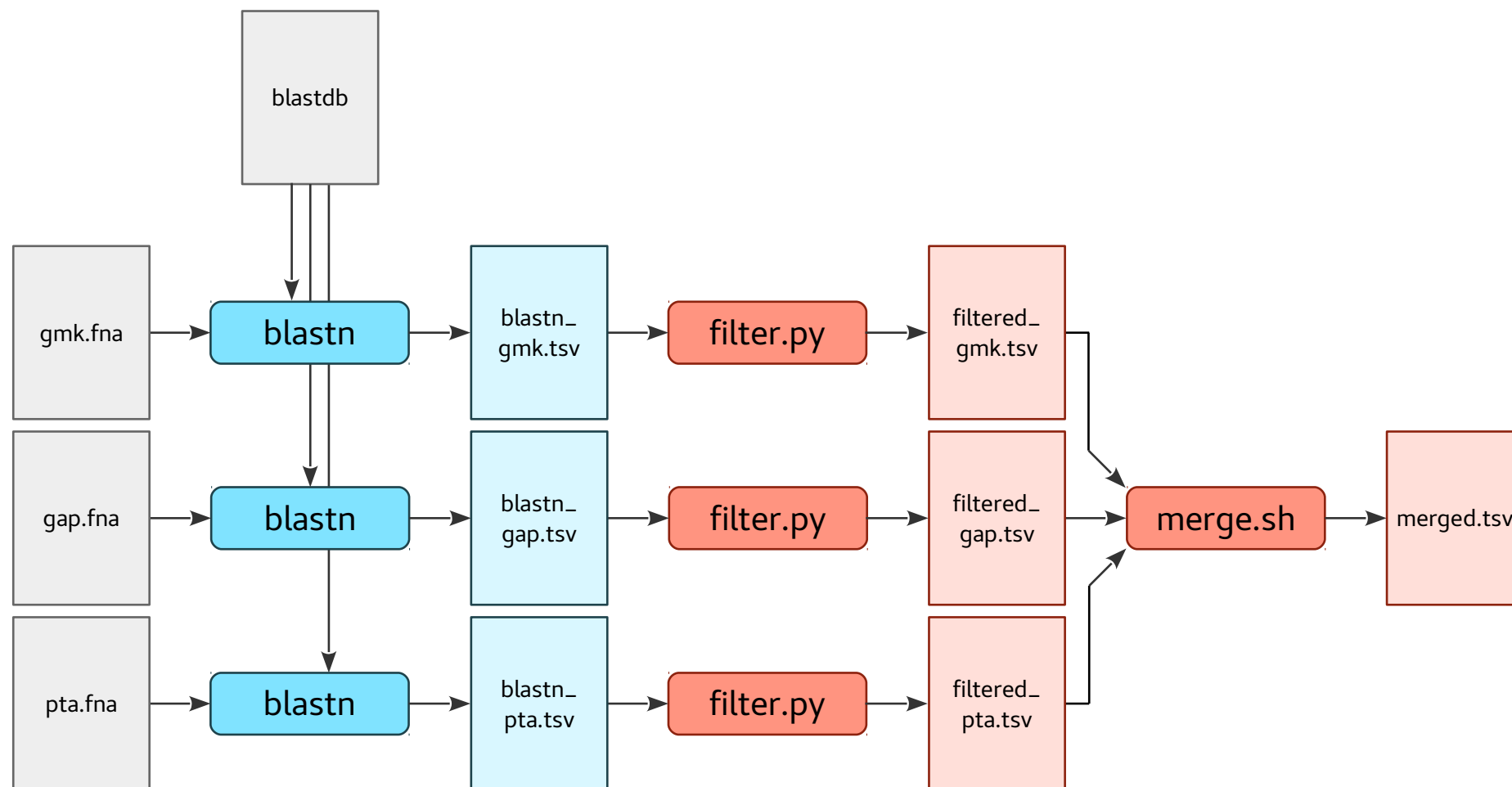
```
snakemake --cores n --directory ścieżka_katalogu
```


Snakemake: przetwarzanie sekwencyjne



```
snakemake --cores 1
```

Snakemake: przetwarzanie równoległe, ilość rdzeni procesora (*cores*)



snakemake --cores 3

Snakemake: uruchamianie ciągu analitycznego

```
# uruchom ciąg analityczny opisany w pliku Snakefile  
# znajdującym się w katalogu roboczym,  
# udostępni n-rdzeni procesora
```

```
snakemake --cores n
```

```
# uruchom ciąg analityczny podając ścieżkę i nazwę pliku Snakefile
```

```
snakemake --cores n --snakefile ścieżka_pliku
```

```
# wskaż katalog gdzie Snakemake ma uruchomić ciąg analityczny
```

```
# jeżeli inny niż ten w którym się znajdujesz
```

```
snakemake --cores n --directory ścieżka_katalogu
```

```
# niezależnie od istniejących plików wynikowych
```

```
# uruchom wszystkie reguły
```

```
snakemake --cores n --forceall
```

```
# niezależnie od istniejących plików wynikowych
```

```
# uruchom konkretną regułę i wszystkie od niej zależne (w dół)
```

```
snakemake --cores n --forcerun rulename
```

Snakemake: inne przydatne komendy

generowanie graficznego schematu ciągu analitycznego

snakemake --dag | dot -Tsvg > dag.svg

uproszczony schemat ciągu analitycznego

snakemake --rulegraph | dot -Tsvg > rulegraph.svg

testowe uruchomienie, bez uruchamiania komend sekcji shell

snakemake --dryrun

podsumowanie zadań do wykonania i ich ilości

snakemake --dryrun --quiet

informacje o wszystkich plikach, które powstały lub

powstaną w trakcie działania ciągu analitycznego

snakemake --summary

Jeżeli nie Snakemake... Nextflow

Snakemake

Dokumentacja:

<https://snakemake.readthedocs.io>

Repozytorium potoków (pipelines):

<https://snakemake.github.io/snakemake-workflow-catalog>

nextflow

Dokumentacja:

<https://www.nextflow.io>

Repozytorium potoków (pipelines):

<https://nf-co.re>

Koniec