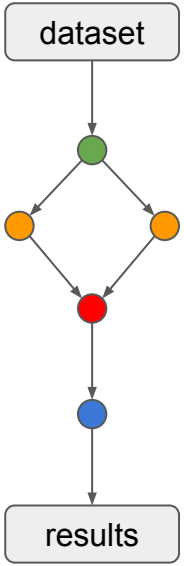
Snakemake

Конструктор биоинформатических пайплайнов

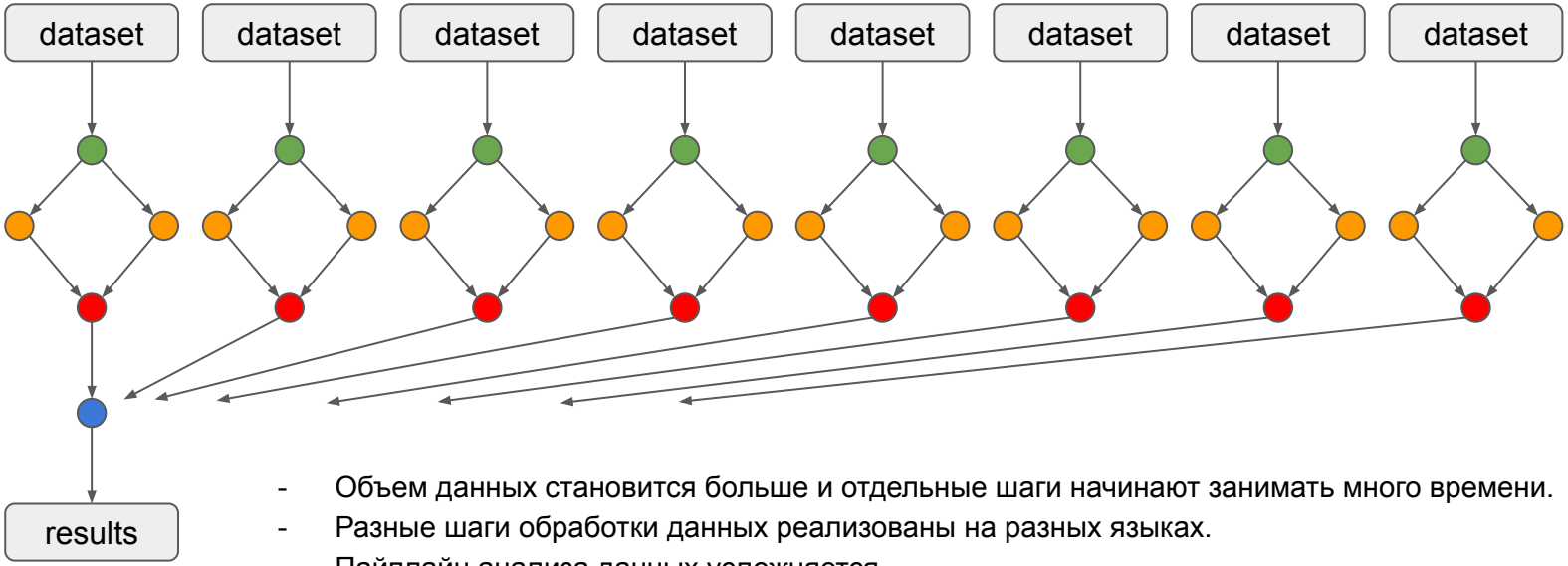
Материал подготовил:
Андрей Томаровский

Биоинформатический пайплайн



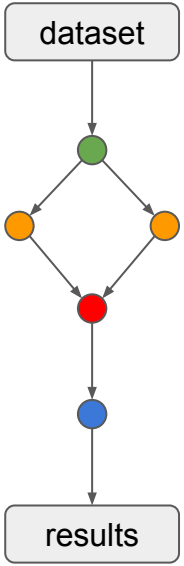

```
#!/usr/bin/bash
ASSEMBLY=”...”
READS=”...”
Command 1 ..
Command 2 ..
Command 3 ..
Command 4 ..
Command 5 ..
```

Биоинформатический пайплайн



- Объем данных становится больше и отдельные шаги начинают занимать много времени.
- Разные шаги обработки данных реализованы на разных языках.
- Пайплайн анализа данных усложняется.

Что можно сделать?




```
#!/usr/bin/bash
ASSEMBLY="..."
READS="..."
Command 1 ..
Command 2 ..
Command 3 ..
Command 4 ..
Command 5 ..
```

→

```
#!/usr/bin/bash
ASSEMBLY="..."
READS="..."
Command 1 ..
```

```
#!/usr/bin/bash
BAM="..."
Command 2 ..
```

```
#!/usr/bin/bash
BAM_BAI="..."
Command 3 ..
```

```
#!/usr/bin/bash
VCF="..."
Command 4 ..
```

```
#!/usr/bin/bash
STATS="..."
Command 5 ..
```

Решение - использовать Snakemake

BIOINFORMATICS APPLICATION NOTE
Vol. 28 no. 19 2012, pages 2520–2522
doi:10.1093/bioinformatics/bts480

Genome analysis
Advance Access publication August 20, 2012

Snakemake—a scalable bioinformatics workflow engine

Johannes Köster[1,2,*] and Sven Rahmann[1]
[1]Genome Informatics, Institute of Human Genetics, University of Duisburg-Essen and [2]Paediatric Oncology, University Childrens Hospital, 45147 Essen, Germany

Associate Editor: Alfonso Valencia

DOI: https://doi.org/10.1093/bioinformatics/bts480
Оф. сайт: https://snakemake.github.io/
Документация: https://snakemake.readthedocs.io/en/stable/
Каталог пайплайнов:
https://snakemake.github.io/snakemake-workflow-catalog/

Полная установка:

```
$ conda install -n base -c conda-forge mamba
$ conda activate base
$ mamba create -c conda-forge -c bioconda -n snakemake snakemake
```

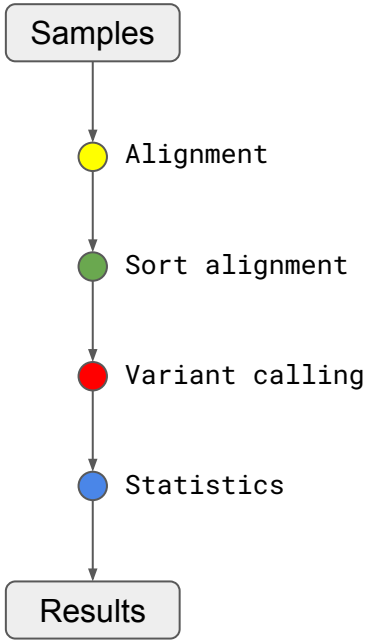
Минимальная установка:

```
$ pip install snakemake
```

Snakemake

Часть 1. Основы

Структура пайплайна




```
SAMPLES = ["A", "B", "C"]

rule all:
    input:
        "results/plots/quals.svg"

rule alignment:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        "results/mapped/{sample}.bam"
    shell:
        "bwa mem {input} | samtools view -b - > {output}"

rule sort_alignment:
    input:
        "results/mapped/{sample}.bam"
    output:
        "results/mapped/{sample}.sorted.bam"
    shell:
        "samtools sort -o {output} {input}"

rule bcftools_call:
    input:
        fa="data/genome.fa",
        bam=expand("results/mapped/{sample}.sorted.bam", sample=SAMPLES),
    output:
        "results/calls/all.vcf"
    shell:
        "bcftools mpileup -f {input.fa} {input.bam} |"
        “bcftools call -mv - > {output}"

rule plot_quals:
    input:
        "results/calls/all.vcf"
    output:
        "results/plots/quals.svg"
    script:
        "scripts/plot-quals.py"
```

Структура правил

```
rule alignment:
    input:
        "data/genome.fa",
        "data/samples/A.fastq"
    output:
        "results/mapped/A.bam"
    shell:
        "bwa mem {input} | samtools view -b - > {output}"
```

Структура правил

```
rule alignment:
    input:
        "data/genome.fa", # [0]
        "data/samples/A.fastq" # [1]
    output:
        "results/mapped/A.bam" # [0]
    shell:
        "bwa mem {input[0]} {input[1]} | samtools view -b - > {output[0]}"
```

Структура правил

```
rule alignment:
    input:
        fa="data/genome.fa",
        fastq="data/samples/A.fastq"
    output:
        bam="results/mapped/A.bam"
    shell:
        "bwa mem {input.fa} {input.fastq} |"
        "samtools view -b - > {output.bam}"
```

Структура правил

```
rule alignment:
    input:
        fa="data/genome.fa",
        fastq="data/samples/{sample}.fastq"
    output:
        bam="results/mapped/{sample}.bam"
    shell:
        "bwa mem {input.fa} {input.fastq} |"
        "samtools view -b - > {output.bam}"
```

Wildcards

```
$ ls data/samples/*.fastq
data/samples/A.fastq
data/samples/B.fastq
data/samples/C.fastq
```

Структура правил

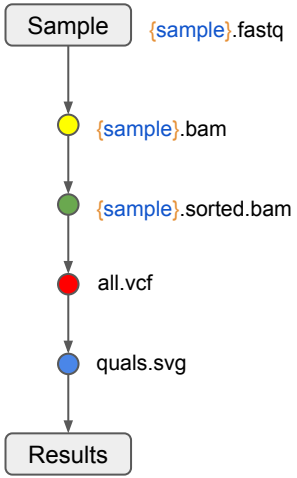
```
rule alignment:
    input:
        fa="data/genome.fa",
        fastq="data/samples/{sample}.fastq"
    output:
        bam="results/mapped/{sample}.bam"
    run:
        with open(output.bam, “w”) as out:
            for l in sorted(open(input.a)):
                out.write(l)
```

Структура правил

```
rule alignment:
    input:
        fa="data/genome.fa",
        fastq="data/samples/{sample}.fastq"
    output:
        bam="results/mapped/{sample}.bam"
    script:
        "scripts/plot-quals.py"
```

```
#!/usr/bin/python3
with open(snakemake.output.bam, “w”) as out:
    for l in sorted(open(snakemake.input[0])):
        out.write(l)
```

Принцип работы Snakemake



Alignment	*.bam
Sort alignment	*.sorted.bam
Variant calling	*.vcf
Statistics	*.svg

```
SAMPLES = ["A", "B", "C"]

rule all:
    input:
        "results/plots/quals.svg"

rule alignment:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        "results/mapped/{sample}.bam"
    shell:
        "bwa mem {input} | samtools view -b - > {output}"

rule sort_alignment:
    input:
        "results/mapped/{sample}.bam"
    output:
        "results/mapped/{sample}.sorted.bam"
    shell:
        "samtools sort -o {output} {input}"

rule bcftools_call:
    input:
        fa="data/genome.fa",
        bam=expand("results/mapped/{sample}.sorted.bam", sample=SAMPLES),
    output:
        "results/calls/all.vcf"
    shell:
        "bcftools mpileup -f {input.fa} {input.bam} |"
        “bcftools call -mv - > {output}"

rule plot_quals:
    input:
        "results/calls/all.vcf"
    output:
        "results/plots/quals.svg"
    script:
        "scripts/plot-quals.py"
```

Принцип работы Snakemake

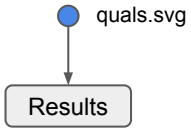
```
SAMPLES = ["A", "B", "C"]

rule all:
    input:
        "results/plots/quals.svg"
```

Results

Alignment	*.bam
Sort alignment	*.sorted.bam
Variant calling	*.vcf
Statistics	*.svg

Принцип работы Snakemake

```
SAMPLES = ["A", "B", "C"]

rule all:
    input:
        "results/plots/quals.svg"
```

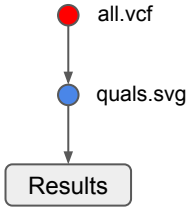


Alignment	*.bam
Sort alignment	*.sorted.bam
Variant calling	*.vcf
Statistics	*.svg

```
rule plot_quals:
    input:
        "results/calls/all.vcf"
    output:
        "results/plots/quals.svg"
    script:
        "scripts/plot-quals.py"
```

Принцип работы Snakemake

```
SAMPLES = ["A", "B", "C"]

rule all:
    input:
        "results/plots/quals.svg"
```

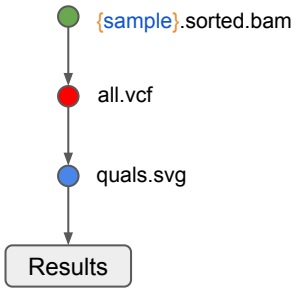



```
rule bcftools_call:
    input:
        fa="data/genome.fa",
        bam=expand("results/mapped/{sample}.sorted.bam", sample=SAMPLES),
    output:
        "results/calls/all.vcf"
    shell:
        "bcftools mpileup -f {input.fa} {input.bam} |"
        “bcftools call -mv - > {output}"

rule plot_quals:
    input:
        "results/calls/all.vcf"
    output:
        "results/plots/quals.svg"
    script:
        "scripts/plot-quals.py"
```

Alignment	*.bam
Sort alignment	*.sorted.bam
Variant calling	*.vcf
Statistics	*.svg

Принцип работы Snakemake

```
SAMPLES = ["A", "B", "C"]

rule all:
    input:
        "results/plots/quals.svg"
```

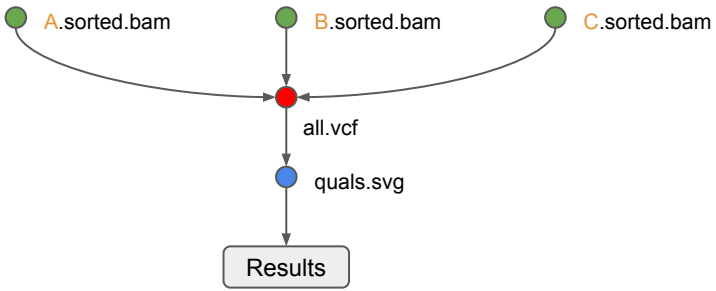



```
rule sort_alignment:
    input:
        "results/mapped/{sample}.bam"
    output:
        "results/mapped/{sample}.sorted.bam"
    shell:
        "samtools sort -o {output} {input}"

rule bcftools_call:
    input:
        fa="data/genome.fa",
        bam=expand("results/mapped/{sample}.sorted.bam", sample=SAMPLES),
    output:
        "results/calls/all.vcf"
    shell:
        "bcftools mpileup -f {input.fa} {input.bam} |"
        "bcftools call -mv - > {output}"

rule plot_quals:
    input:
        "results/calls/all.vcf"
    output:
        "results/plots/quals.svg"
    script:
        "scripts/plot-quals.py"
```

Alignment	*.bam
Sort alignment	*.sorted.bam
Variant calling	*.vcf
Statistics	*.svg

Принцип работы Snakemake

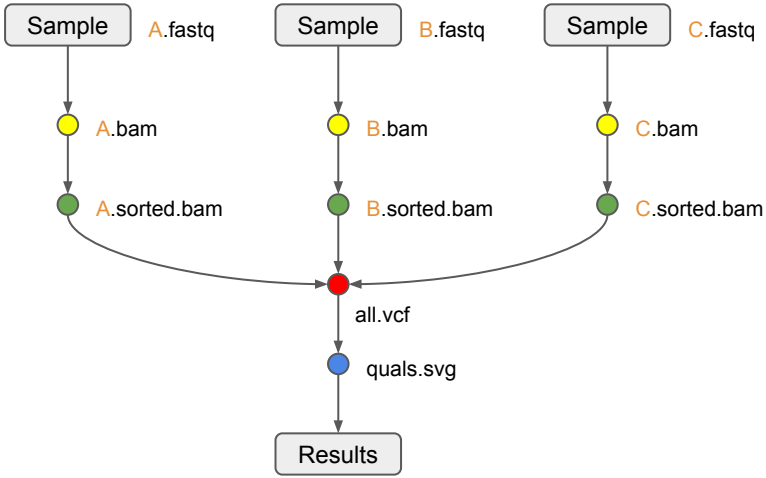
```
SAMPLES = ["A", "B", "C"]

rule all:
    input:
        "results/plots/quals.svg"
```




```
rule sort_alignment:
    input:
        "results/mapped/{sample}.bam"
    output:
        "results/mapped/{sample}.sorted.bam"
    shell:
        "samtools sort -o {output} {input}"

rule bcftools_call:
    input:
        fa="data/genome.fa",
        bam=expand("results/mapped/{sample}.sorted.bam", sample=SAMPLES),
    output:
        "results/calls/all.vcf"
    shell:
        "bcftools mpileup -f {input.fa} {input.bam} |"
        "bcftools call -mv - > {output}"

rule plot_quals:
    input:
        "results/calls/all.vcf"
    output:
        "results/plots/quals.svg"
    script:
        "scripts/plot-quals.py"
```

Alignment	*.bam
Sort alignment	*.sorted.bam
Variant calling	*.vcf
Statistics	*.svg

Принцип работы Snakemake



Alignment	*.bam
Sort alignment	*.sorted.bam
Variant calling	*.vcf
Statistics	*.svg

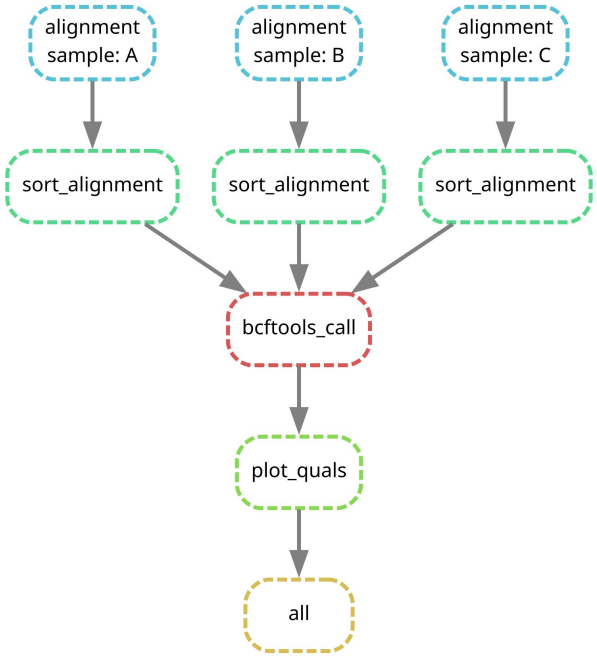
```
SAMPLES = ["A", "B", "C"]

rule all:
    input:
        "results/plots/quals.svg"

rule alignment:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        "results/mapped/{sample}.bam"
    shell:
        "bwa mem {input} | samtools view -b - > {output}"

rule sort_alignment:
    input:
        "results/mapped/{sample}.bam"
    output:
        "results/mapped/{sample}.sorted.bam"
    shell:
        "samtools sort -o {output} {input}"

rule bcftools_call:
    input:
        fa="data/genome.fa",
        bam=expand("results/mapped/{sample}.sorted.bam", sample=SAMPLES),
    output:
        "results/calls/all.vcf"
    shell:
        "bcftools mpileup -f {input.fa} {input.bam} |"
        "bcftools call -mv - > {output}"

rule plot_quals:
    input:
        "results/calls/all.vcf"
    output:
        "results/plots/quals.svg"
    script:
        "scripts/plot-quals.py"
```

Условия выполнения Snakemake

Пайплайн будет выполнятся в том случае, если:

- Конечный выходной файл является целевым и не существует.
- Выходной файл одного из правил нужен другому правилу и не существует.
- Входной файл новее выходного файла.
- Указана опция для принудительного выполнения (`--forceall`). В этом случае, snakemake заново создаст новые файлы, даже если в их обновлении нет необходимости.

Интерфейс командной строки




```
# dry-run (пробный запуск) и печать выполняемых команд:
snakemake -s Snakefile --dry-run --print

# Запуск пайплайна для получения выходного файла A.bam:
snakemake -s Snakefile results/mapped/A.bam

# Запуск пайплайна до тех пор, пока он не достигнет указанных правил или файлов:
snakemake --until bcftools_call

# Запуск с повторным выполнением или созданием указанных правил или файлов:
snakemake --forcerun results/mapping/B.bam

# Запуск правил, выполнение которых было не завершено:
snakemake --rerun-incomplete

# Сохранять выходные файлы правил, выполнение которых завершилось ошибкой:
snakemake --keep-incomplete

# Визуализация графа пайплайна при помощи dot:
snakemake --dag | dot -T svg > dag.svg

# Сформировать отчет по запущенному пайплайну:
snakemake --report report.html
```

Conda

```
rule alignment:
    input:
        fa="data/genome.fa",
        fastq="data/samples/{sample}.fastq"
    output:
        bam="results/mapped/{sample}.bam"
    conda:
        "conda.yaml"
    shell:
        "bwa mem {input.fa} {input.fastq} |"
        "samtools view -b - > {output.bam}"
```

```
channels:
  - bioconda
  - conda-forge
dependencies:
  - bcftools
  - samtools
  - bwa
  - quast
```

```
# автоматическая установка необходимых инструментов:
$ snakemake --use-conda
```

Snakemake

Часть 2. Дополнительные возможности

Параметры

```
rule alignment:
    input:
        fa="data/genome.fa",
        fastq="data/samples/{sample}.fastq"
    output:
        bam="results/mapped/{sample}.bam"
    params:
        header=r"@RG\tID:{sample}\tSM:{sample}\tPL:Illumina"
    shell:
        "bwa mem -R {params.header} -t {threads} {input.fa} {input.fastq} |"
        "samtools view -b - > {output.bam}"
```

Логирование

```
rule alignment:
    input:
        fa="data/genome.fa",
        fastq="data/samples/{sample}.fastq"
    output:
        bam="results/mapped/{sample}.bam"
    params:
        header=r"@RG\tID:{sample}\tSM:{sample}\tPL:Illumina"
    logs:
        stderr="logs/alignment.{sample}.log"
    shell:
        "(bwa mem -R {params.header} -t {threads} {input.fa} {input.fastq} |"
        "samtools view -b - > {output.bam}) 2> {logs.stderr}"
```

Ресурсы

```
rule alignment:
    input:
        fa="data/genome.fa",
        fastq="data/samples/{sample}.fastq"
    output:
        bam="results/mapped/{sample}.bam"
    resources:
        cpus=4,
        mem_mb=8000,
        time="10:00:00"
    threads: 4
    shell:
        "bwa mem -t {threads} {input.fa} {input.fastq} |"
        "samtools view -b - > {output.bam}"
```

```
# параллельное выполнение двух правил “alignment”
$ snakemake --cores 8
```

Ресурсы

```
rule alignment:
    input:
        fa="data/genome.fa",
        fastq="data/samples/{sample}.fastq"
    output:
        bam="results/mapped/{sample}.bam"
    resources:
        cpus=4,
        mem_mb=8000,
        time="10:00:00"
    threads: 4
    shell:
        "bwa mem -t {threads} {input.fa} {input.fastq} |"
        "samtools view -b - > {output.bam}"
```

```
# последовательное выполнение правил “alignment” с -t 2
$ snakemake --cores 2
```

Ресурсы

```
rule alignment:
    input:
        fa="data/genome.fa",
        fastq="data/samples/{sample}.fastq"
    output:
        bam="results/mapped/{sample}.bam"
    resources:
        cpus=4,
        mem_mb=8000,
        time="10:00:00"
    threads: 4
    shell:
        "bwa mem -t {threads} {input.fa} {input.fastq} |"
        "samtools view -b - > {output.bam}"
```

```
# последовательное выполнение правил “alignment” с -t 4 и mem_mb 8000 mb
$ snakemake --cores 10 --resources mem_mb=8000
```

Config-файл

```
SAMPLES = ["A", "B", "C"]

rule sort_alignment:
    input:
        "results/mapped/{sample}.bam"
    output:
        "results/mapped/{sample}.sorted.bam"
    threads: 1
    resources:
        mem_mb=4000,
        time="2:00:00"
    shell:
        "samtools sort -@ {threads} -o {output} {input}"

rule bcftools_call:
    input:
        fa="data/genome.fa",
        bam=expand("results/mapped/{sample}.sorted.bam", sample=SAMPLES),
    output:
        "results/calls/all.vcf"
    shell:
        "bcftools mpileup -f {input.fa} {input.bam} | bcftools call -mv - > {output}"
```

Config-файл

```
configfile: "config.yaml"

rule sort_alignment:
    input:
        "results/mapped/{sample}.bam"
    output:
        "results/mapped/{sample}.sorted.bam"
    threads: config["sort_align_threads"]
    resources:
        mem_mb=config["sort_align_mem_mb"],
        time=config["sort_align_time"]
    shell:
        "samtools sort -@ {threads} -o {output} {input}"

rule bcftools_call:
    input:
        fa="data/genome.fa",
        bam=expand("results/mapped/{sample}.sorted.bam", sample=config["samples"]),
    output:
        "results/calls/all.vcf"
    shell:
        "bcftools mpileup -f {input.fa} {input.bam} | bcftools call -mv - > {output}"
```

```
$ cat config.yaml
# ---- samples ----
samples: ["A", "B", "C"]

# ---- resources ----
sort_alignment_threads: 1
sort_alignment_mem_mb: 4000
sort_alignment_time: "2:00:00"
```

Python: входные функции

```
configfile: "config.yaml"

def header_line(wildcards):
    if config["header_line"]:
        return "-R {}".format(config["header_line"])
    else:
        return ""

rule alignment:
    input:
        fa="data/genome.fa",
        fastq=lambda wildcards: config["samples"][wildcards.sample]
    output:
        bam="results/mapped/{sample}.bam"
    params:
        header=header_line,
        prefix=lambda wildcards, output: output[0][:-4]
    threads: 4
    shell:
        "bwa mem {params.header} -t {threads} {input.fa} {input.fastq} |"
        "samtools view -b - > {output.bam}"
```

Python: Условия

```
if config["alignment_tool"] == "bwa":
    rule alignment:
        input:
            fa="data/genome.fa",
            fastq=lambda wildcards: config["samples"][wildcards.sample]
        output:
            bam="results/mapped/{sample}.bam"
        params:
            prefix=lambda wildcards, output: output[0][:-4]
       logs:
           stderr="logs/alignment.{sample}.log"
        threads: 4
        shell:
            "(bwa mem {params.header} -t {threads} {input.fa} {input.fastq} |"
            "samtools view -b - > {params.prefix}.bam) 2> {logs.stderr}"

elif config["alignment_tool"] == "bowtie":
    ...
```

Python: Условия

```
rule alignment:
    input:
        fa="data/genome.fa",
        fastq=lambda wildcards: config["samples"][wildcards.sample]
    output:
        bam="results/mapped/{sample}.bam"
    params:
        tool=config["alignment_tool"],
        index=lambda wildcards, input: input[0][:-3]   # fasta prefix
    logs:
        stderr="logs/alignment.{sample}.log"
    threads: 4
    run:
    ⟶  if params.tool = "bwa":
            shell("bwa mem {params.header} -t {threads} {input.fa} {input.fastq} |
            samtools view -b - > {params.prefix}.bam")
        elif params.tool = "bowtie":
            shell("bowtie2 -p {threads} -x {params.index} -U {input.fastq} |
            samtools view -b - > {params.prefix}.bam")
```

Временные и защищенные файлы

```
rule alignment:
    input:
        fa="data/genome.fa",
        fastq="data/samples/{sample}.fastq"
    output:
        temp("results/mapped/{sample}.bam")
    shell:
        "bwa mem -t {threads} {input.fa} {input.fastq} |"
        "samtools view -b - > {output}"


rule sort_alignment:
    input:
        bam="results/mapped/{sample}.bam"
    output:
        protected("results/mapped/{sample}.sorted.bam")
    shell:
        "samtools sort -@ {threads} -o {output} {input.bam}"
```

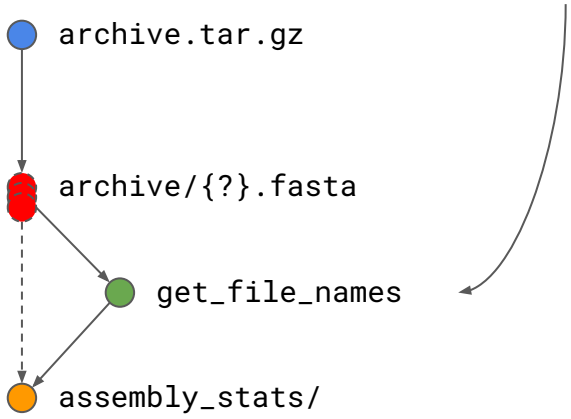
-rw-r--r--

protected(-r--r--r--)

Checkpoint

```
def get_file_names(wildcards):
    checkpoint_output = checkpoints.extract_some_files.get(**wildcards).output[0]
    samples = glob_wildcards(os.path.join(checkpoint_output, "{sample}.fasta")).sample
    return expand("archive/{sample}.fasta", sample=samples)

rule all:
    input:
        "assembly_stats"

checkpoint extract_some_files:
    input:
        "archive.tar.gz"
    output:
        directory("archive")
    shell:
        "tar -xf {input}"

rule quast:
    input:
        get_file_names
    output:
        directory("assembly_stats")
    shell:
       "quast --output-dir {output} {input}"
```

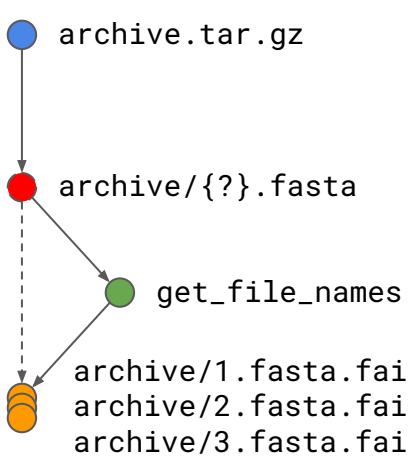
```
['archive/1.fasta',
 'archive/2.fasta',
 'archive/3.fasta']
```



Checkpoint

```
def get_file_names(wildcards):
    checkpoint_output = checkpoints.extract_some_files.get(**wildcards).output[0]
    samples = glob_wildcards(os.path.join(checkpoint_output, "{sample}.fasta")).sample
    return expand("archive/{sample}.fasta.fai", sample=samples)

rule all:
    input:
        get_file_names

checkpoint extract_some_files:
    input:
        "archive.tar.gz"
    output:
        directory("archive")
    shell:
        "tar -xf {input}"

rule rename:
    input:
        "archive/{sample}.fasta"
    output:
        "archive/{sample}.fasta.fai"
    shell:
       "samtools faidx {input}"
```

```
['archive/1.fasta.fai',
 'archive/2.fasta.fai',
 'archive/3.fasta.fai']
```



Использование на кластере


```
$ snakemake --jobs 30 \
            --cluster "sbatch..."
```

```
$ snakemake --profile /path/to/profile.yaml
$ cat /path/to/profile.yaml
# Количество параллельных задач:
jobs:                                                    30

# Использование Conda
use-conda: True

# Ожидать выходные файлы 1 минуту
latency-wait: 60

# Печатать лог в случае ошибки
show-failed-logs: True

# Перезапуск упавших задач (3 попытки)
rerun-incomplete: True
restart-times: 3

# Команда для планировщика задач SLURM:
cluster: "sbatch -t {resources.time} -m {resources.mem_mb}
-c {resources.cpus} -o {log.cluster_log} -e {log.cluster_err}"

# Команда для планировщика задач PBS:
cluster: "qsub -l
walltime={resources.time},mem={resources.mem_mb},nodes=1:ppn={
resources.cpus} -o {log.cluster_log} -e {log.cluster_err}"
```

Бенчмаркинг

```
rule alignment:
    input:
        fa="data/genome.fa",
        fastq="data/samples/{sample}.fastq"
    output:
        bam="results/mapped/{sample}.bam"
    params:
        header=r"@RG\tID:{sample}\tSM:{sample}\tPL:Illumina"
    benchmark:
       "benchmarks/alignment.{sample}.log"
    logs:
        stderr="logs/alignment.{sample}.log"
    resources:
        cpus=4,
        mem_mb=8000,
        time="10:00:00"
    threads: 4
    shell:
        "(bwa mem -R {params.header} -t {threads} {input.fa} {input.fastq} |"
        "samtools view -b - > {output.bam}) 2> {logs.stderr}"
```

Бенчмаркинг

```
$ cat benchmarks/alignment.A.log
s          h:m:s     max_rss   max_vms   max_uss   max_pss   io_in   io_out   mean_load   cpu_time
504.4501   0:08:24   66.95     8080.21   40.27     46.70     13.12   0.29     97.39       490.61
```

Метрика	Тип данных	Описание
s	float (сек)	Время выполнения процесса в секундах
h:m:s	строка	Время выполнения процесса в формате “часы:минуты:секунды”
max_rss	float (MB)	Resident Set Size (RSS) - физическая используемая память ОЗУ
max_vms	float (MB)	Virtual Memory Size (VMS) - используемая память ОЗУ и диска
max_uss	float (MB)	Unique Set Size (USS) - уникальная память, используемая процессом
max_pss	float (MB)	Proportional Set Size (PSS) - объем памяти с учетом общего использования
io_in	float (MB)	Суммарное количество прочитанных MB
io_out	float (MB)	Суммарное количество записанных MB
mean_load	float	Средняя нагрузка процессора
cpu_time	float (сек)	Процессорное время в секундах, затраченное на выполнение задачи

Распространение Snakemake-пайплайнов

```
# GitHub репозиторий:
|- workflow/
|     |- envs/
|     |     |- default.yaml
|     |- rules/
|     |     |- rules_1.smk
|     |     |- rules_2.smk
|     |- scripts/
|     |     |- script_1.py
|     |     |- script_2.R
|- Snakefile
|- config.yaml
|- README.md
```

```
# Архивация пайплайна
$ snakemake --archive myworkflow.tar.gz
```

```
# Клонирование репозитория в рабочую директорию:
$ git clone https://github.com/workflows/variant-calling.git

# Изменение конфиг-файла, если это необходимо:
$ vim config.yaml

# Запуск пайплайна:
$ conda activate snakemake
$ snakemake --dry-run --use-conda
```