目录

1.课程简介

为什么出这个课程?

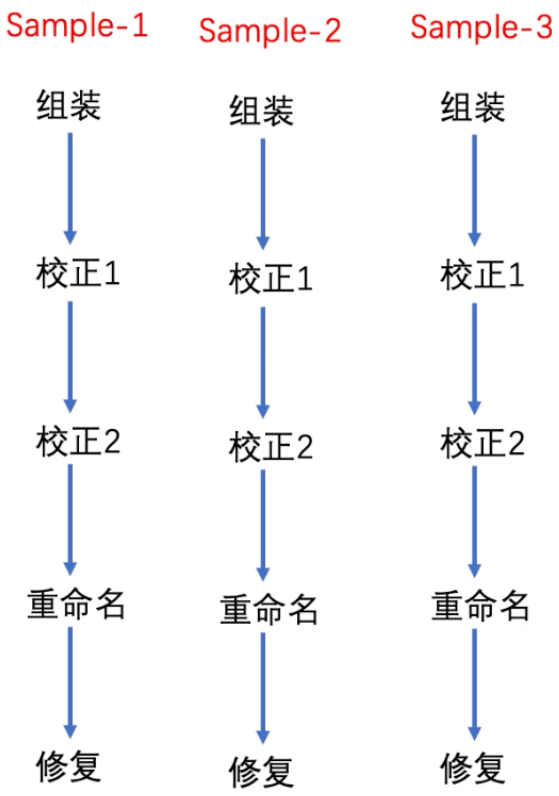
能交给计算机干的事情我们就让给他，我们腾出时间去玩不香吗?

适合人群

有一定的python基础，会基本的linux操作，希望使用Snakemake来提高生信分析效率

Snakemake概况

- 背景



你是否曾经满怀期待地去看程序跑完了没有，想接着跑下一步?

- 以前的解决方式
  - 单任务多样本并行

```
# kneaddata是一套工作流，依赖trimmomatic进行质控和去接头；
# 依赖bowtie2比对宿主，并筛选非宿主序列
# kneaddata -h # 显示帮助
# 现实中是有一大堆样品，你可以逐个修改样品名运行，如果服务器性能允许可以并行加速分析
# parallel --citation # 打will cite以后不再提醒
time parallel -j 3 --xapply \
  'kneaddata -i {1} -i {2} \
  -o temp/qc -v -t 3 --remove-intermediate-output \
  --trimmomatic /conda2/share/trimmomatic-0.36-3/ --trimmomatic-options
"SLIDINGWINDOW:4:20 MINLEN:50" \
  --bowtie2-options "--very-sensitive --dovetail" -db
/db/bowtie2/Homo_sapiens' \
 ::: seq/*_1.fq.gz ::: seq/*_2.fq.gz
# real 16s, user 1m28s
```

```
# Usage of the script:
# for one file:
sbatch 01_filtering.sh readFile1.gz readFile2.gz
# or for all files:
for f in *_R1_001.fastq.gz;
do
sbatch ~/path/01_filtering.sh $f ${f%*R1*}R2_001. !→fastq.gz;
done
```

- 单样本多任务按顺序运行，按顺序将命令堆放在一个shell文件里

```
# 2.4.5 LEfSe差异分析和Cladogram
# 修改样本品为组名
sed '1 s/p[0-9]*//g' result/metaphlan2/taxonomy.tsv | grep -v '#' >
result/metaphlan2/lefse.txt
# 格式转换为lefse内部格式
lefse-format_input.py result/metaphlan2/lefse.txt temp/input.in -c 1 -o
1000000
# 运行lefse
run_lefse.py temp/input.in temp/input.res
# 绘制物种树注释差异
lefse-plot_cladogram.py temp/input.res result/metaphlan2/lefse_cladogram.pdf -
-format pdf --dpi 600
# 绘制所有差异features柱状图
lefse-plot_res.py temp/input.res result/metaphlan2/lefse_res.pdf --format pdf
--dpi 600
# 绘制单个features柱状图(同STAMP中barplot)
grep -v '-' temp/input.res | sort -k3,3n  # 查看显著差异features，按丰度排序
lefse-plot_features.py -f one --feature_name "k__Bacteria.p__Firmicutes" --
format pdf \
  temp/input.in temp/input.res result/metaphlan2/Firmicutes.pdf
# 批量绘制所有差异features柱状图
lefse-plot_features.py -f diff --archive none --format pdf \
  temp/input.in temp/input.res result/metaphlan2/features
```

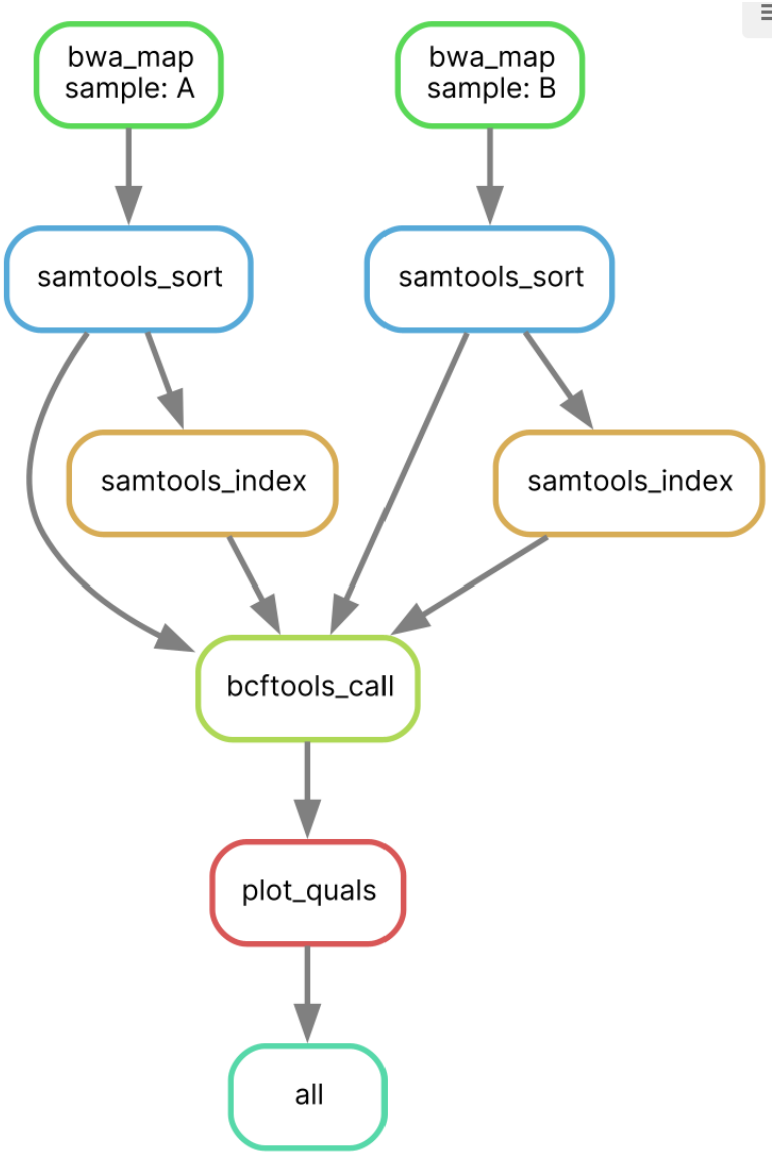
- 多任务多样本并行，自己写python脚本

```
#以我之前写的assembly.py为列，实现二代基因组的组装
```

- 上述方法的缺点：1）都没有统一的编写规则，遇到新的需求就得重新写；2）可读性不强、代码冗余；3）容错能力低；4）运行环境分配繁琐；5）不能断点续跑

- Snakemake能够解决什么问题?
  - 按照指定的顺序执行任务(命令)，通常是以前一个任务的输出结果作为下一个命令的输入
  - 批量并行地执行任务流
  - 智能分配线程和内存
  - 错误发现与纠正：检查命令是否运行成功、输出文件是否完整等
  - 独立分配运行环境

- 断点续跑
- 完美结合python、R等常用生信分析语言



你将学到什么?

- Snakemake的工作原理
- 在实际案例中学习Snakemake流程文件(`.smk`)的编写
- 一些实用的linux技巧、zshell命令、python语法
- 使用conda (mamba) 搭建案例分析环境
- 使用PyCharm、VS Code实现本地编写远程运行
- 流程运行结束用邮件、微信通知

资料下载

- 链接
- 添加微信进交流群，请备注来自CCtalk的哪个课程

2.搭建分析环境

2.1 配置一个舒适的zshell终端

- 安装zshell

```
sudo apt install zsh
```

- 更改用户的shell程序

```
chsh -s /bin/zsh
```

- 安装oh-my-zsh：官网

```
sh -c "$(wget
https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh -O -)"
```

- 更改主题

```
# 修改配置文件
vim ~/.zshrc
# 修改主题名称
ZSH_THEME="ys"
```

- 添加自动提示插件

```
# 把下载好的插件移动到插件目录
mv zsh-autosuggestions/ ~/.oh-my-zsh/custom/plugins/
# 修改插件配置
vim ~/.zshrc
plugins=(git last-working-dir zsh-autosuggestions)
```

- 实用zshell命令

```
# 批量重命名

autoload -U zmv
zmv 'G([123]A).fq.gz' 'G-7D-$1.fq.gz'
zmv 'G(*).(*).gz' 'G-$1.$2kk.gz'

# 递归搜索

## 查找所有普通文件
print -l **/*(.)
## 查找所有后缀是gz结尾的文件
print -l **/*.gz
## 列出最近一天修改过内容的文件
print -l *(.m-1)
```

```
## 列出最近一个月没有读取过的文件
print -l *(.aM+1)
## 列出当前目录下小于 2k 的文件
print -l *(.Lk-2)

# 字符串截取

aa="1234"
## 取第一个
echo $aa[1]
## 取最后一个
echo $aa[-1]
## 取中间两个
echo $aa[2,3]

# 字符串拼接

% str1=abc
% str2=def

% str2+=$str1
% echo $str2
defabc

# 字符串替换

% str=abcabc

## 只替换找到的第一个
% echo ${str/bc/ef}
aefabc

# for循环
for i (*){
echo $i
}
# while循环
while {read i} {
echo $i
} < Flavo_accession.txt
```

2.2 安装环境管理工具

- miniconda官方网站：[miniconda]

```
# 下载安装脚本到本地
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
# 添加执行权限
chmod +x Miniconda3-latest-Linux-x86_64.sh
# 运行脚本
./Miniconda3-latest-Linux-x86_64.sh
```

- Mambaforge的github地址：Mambaforge

👉增强版的conda，速度更快，交互信息更优雅，推荐安装这个！✅

```
# 下载安装脚本到本地
wget https://github.com/conda-forge/miniforge/releases/latest/download/Mambaforge-pypy3-Linux-x86_64.sh
# 添加执行权限
chmod +x Mambaforge-pypy3-Linux-x86_64.sh
# 运行脚本
./Mambaforge-pypy3-Linux-x86_64.sh
```

2.3 创建新环境

```
# 创建一个新的目录
mkdir snakemake-tutorial
cd snakemake-tutorial
# 下载资料
curl -L https://api.github.com/repos/snakemake/snakemake-tutorial-data/tarball -o snakemake-tutorial-data.tar.gz
# 选择性解压，只解压该压缩包里的data和environment.yaml
tar --wildcards -xf snakemake-tutorial-data.tar.gz --strip 1 "*/data" "*/environment.yaml"
# 以environment.yaml里面的配置信息新建一个名字为snakemake-tutorial的环境
mamba env create --name snakemake-tutorial --file environment.yaml
# 激活该环境
mamba activate snakemake-tutorial
```

- 安装r来支持运行R脚本，以及vcfR包用于读取vcf文件

```
mamba install -c r r r-vcfR
```

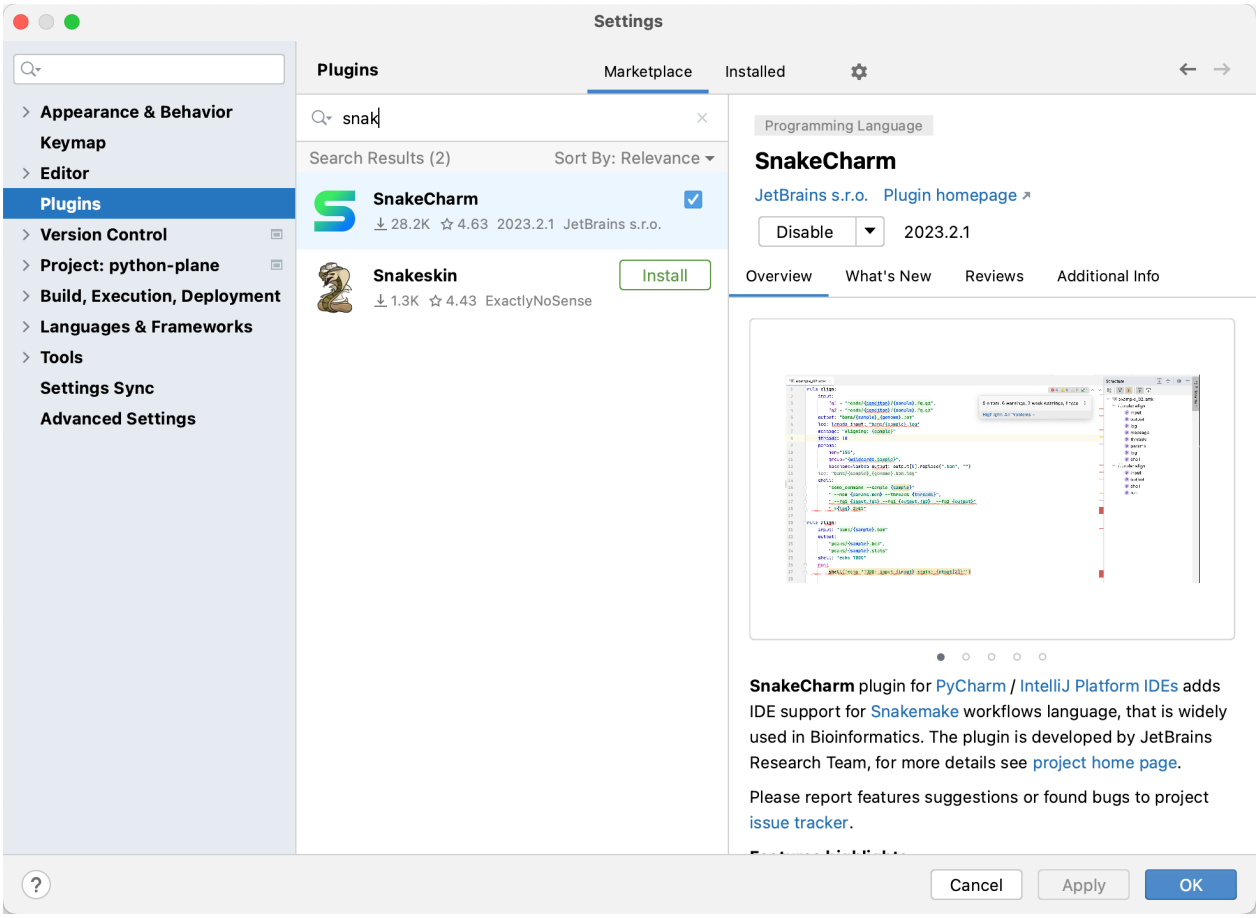
2.4 集成开发环境配置

2.4.1 PyCharm

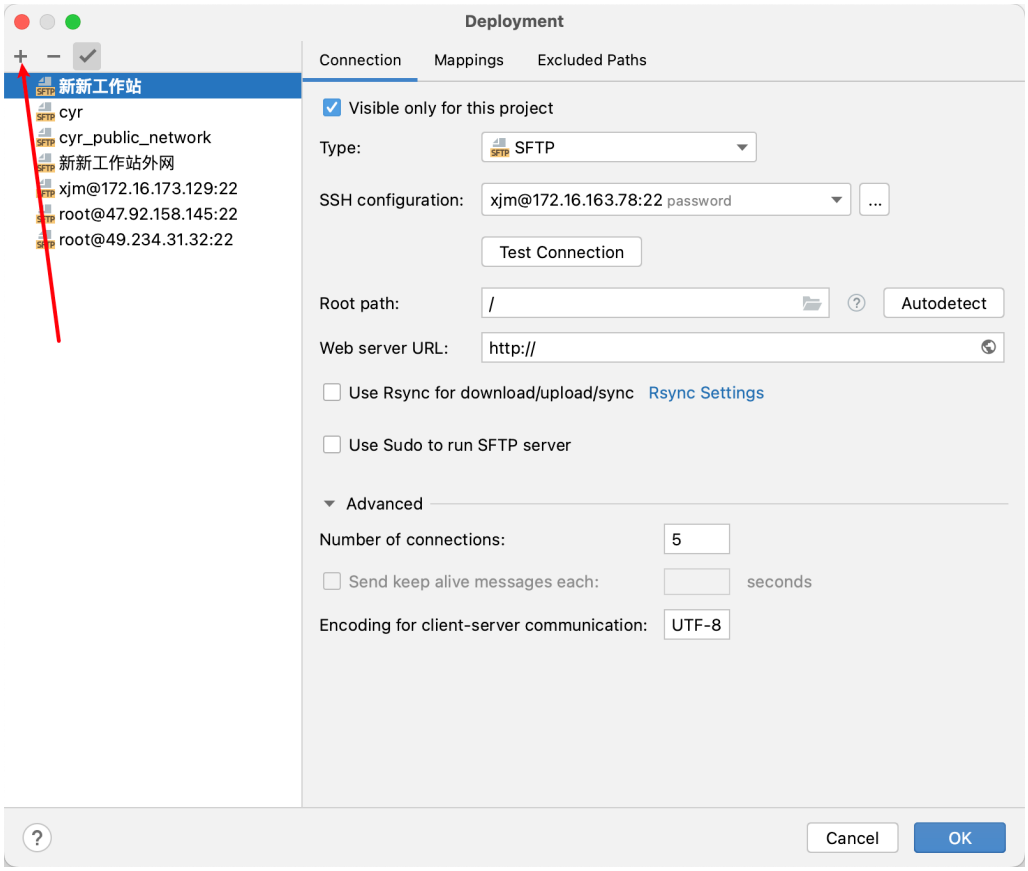
软件下载链接

- PyCharm 官方网站：链接
- Gateway 官方网站：链接
- 申请教育免费链接：链接

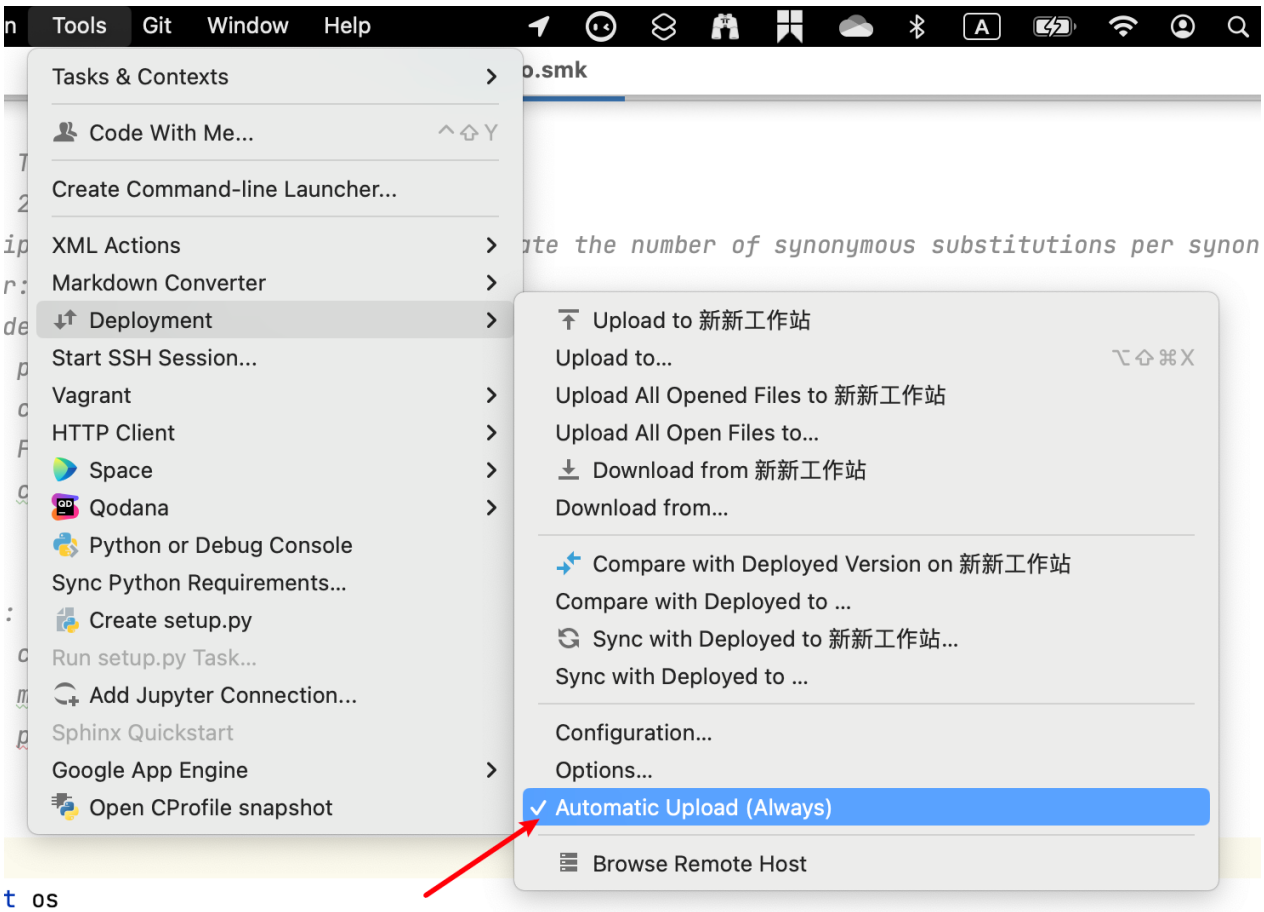
安装snakeCharm插件



配置本地目录和服务器目录的映射



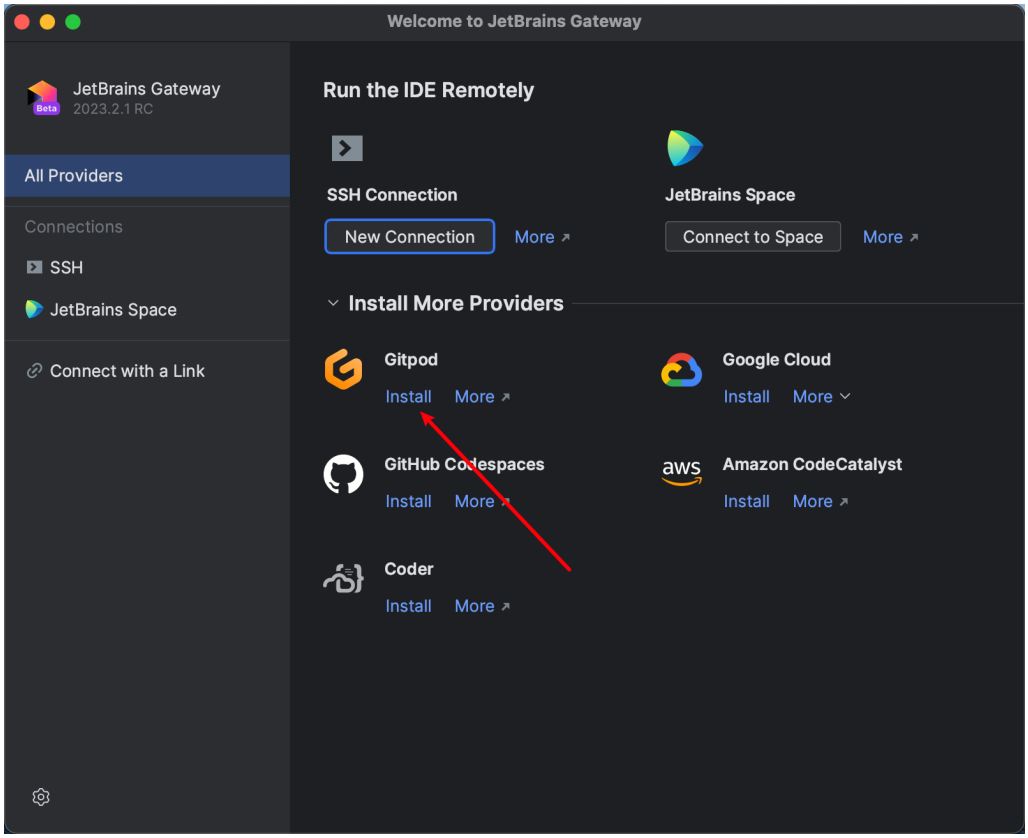
开启自动同步



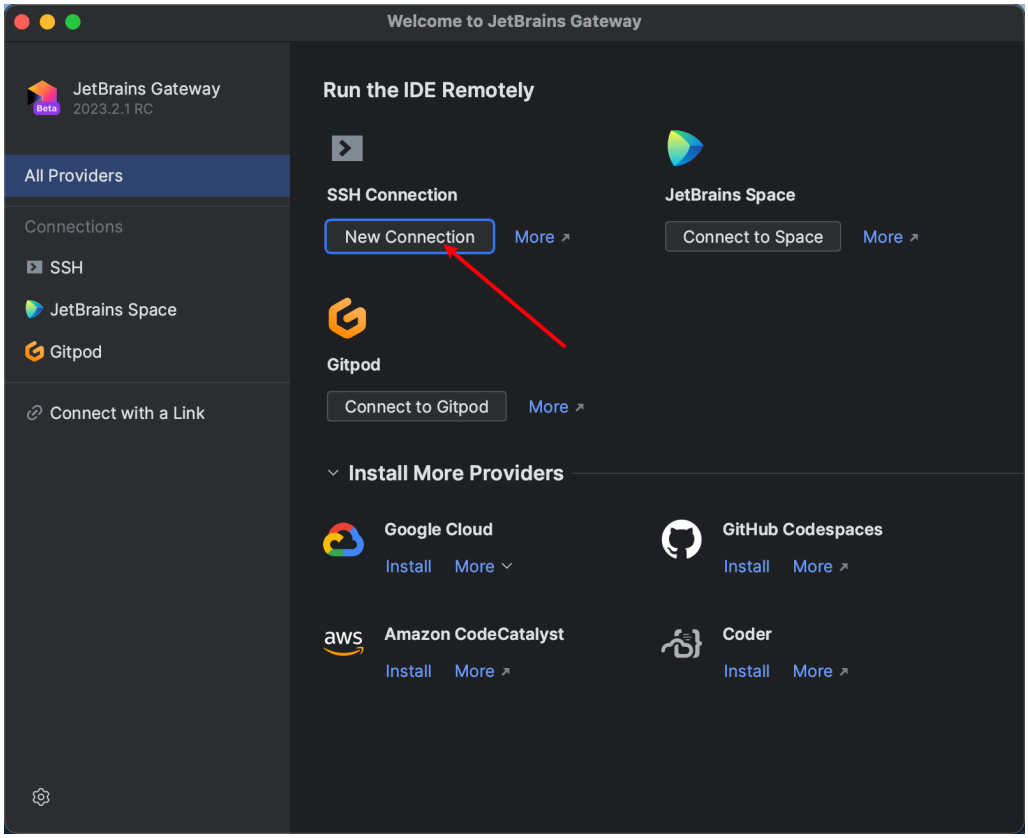

```
t os
os import path
```

也可以使用Gateway直接进行远程开发

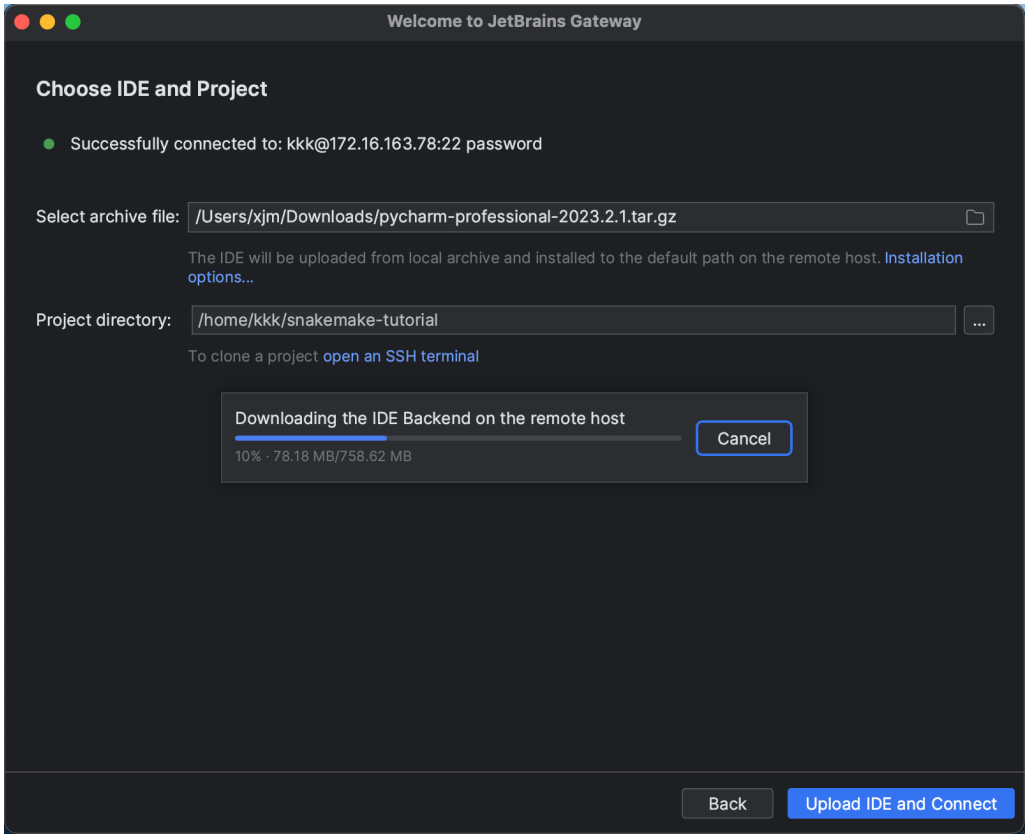
- 安装Gitpod插件



- 使用Gitpod免费环境：snakemake-tutorial GitPod workspace
- 使用自己的服务器



- 选择IDE和开发目录，服务器下载速度慢的话可以上传本地下载好的IDE

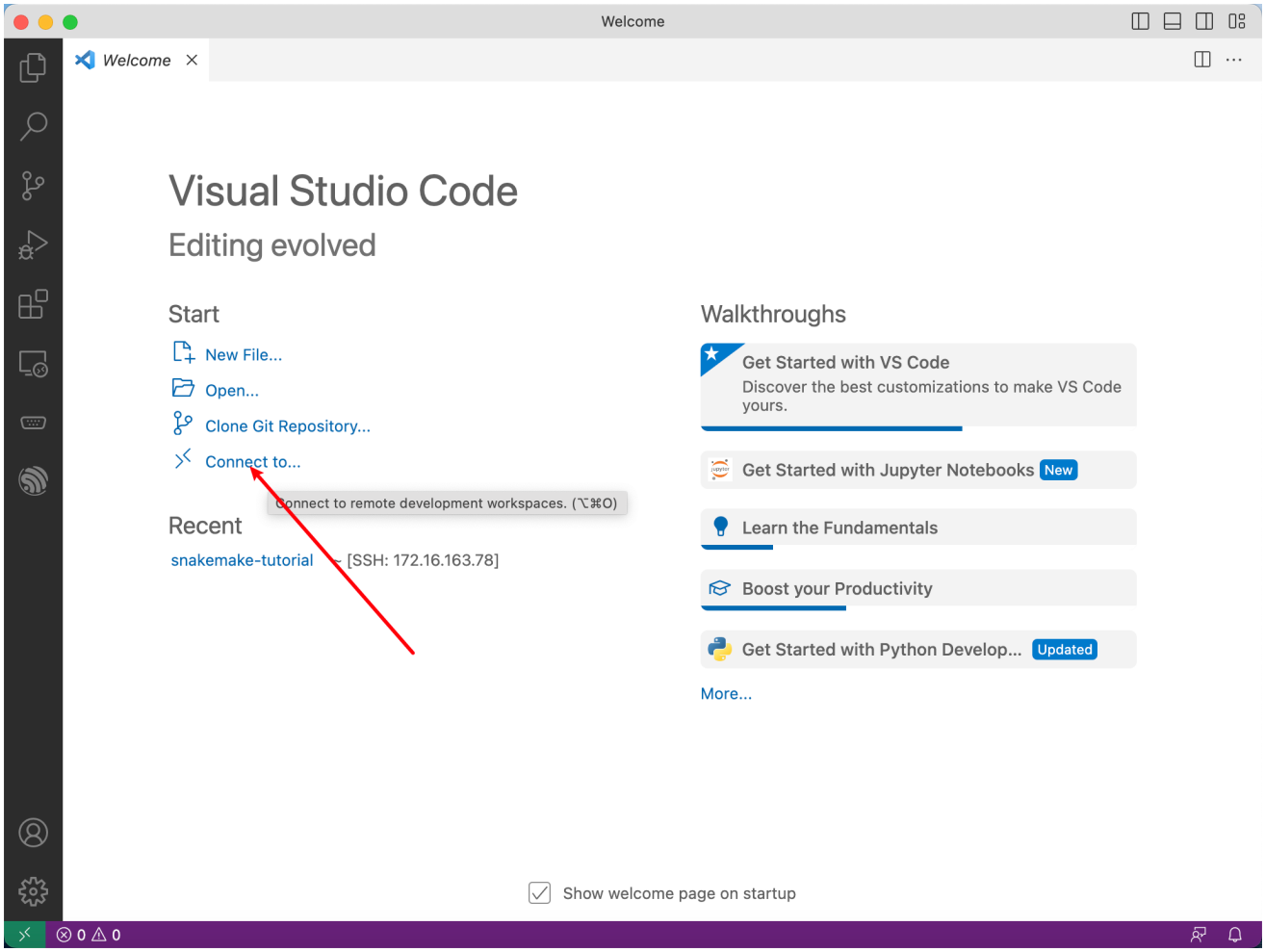


2.4.2 VS Code

> ## 软件下载

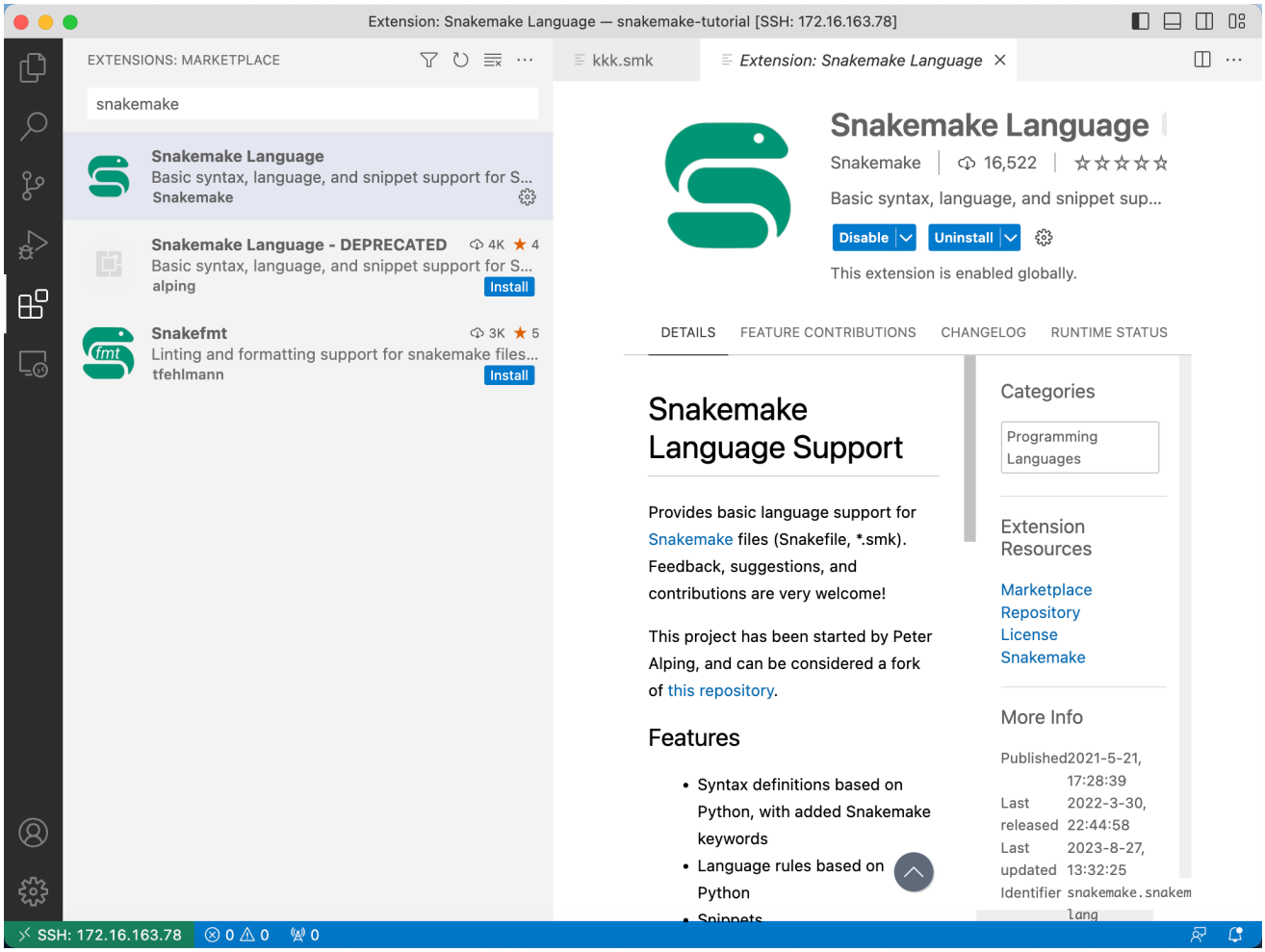
- VS Code 官方网站：链接

> ## 连接远程服务器

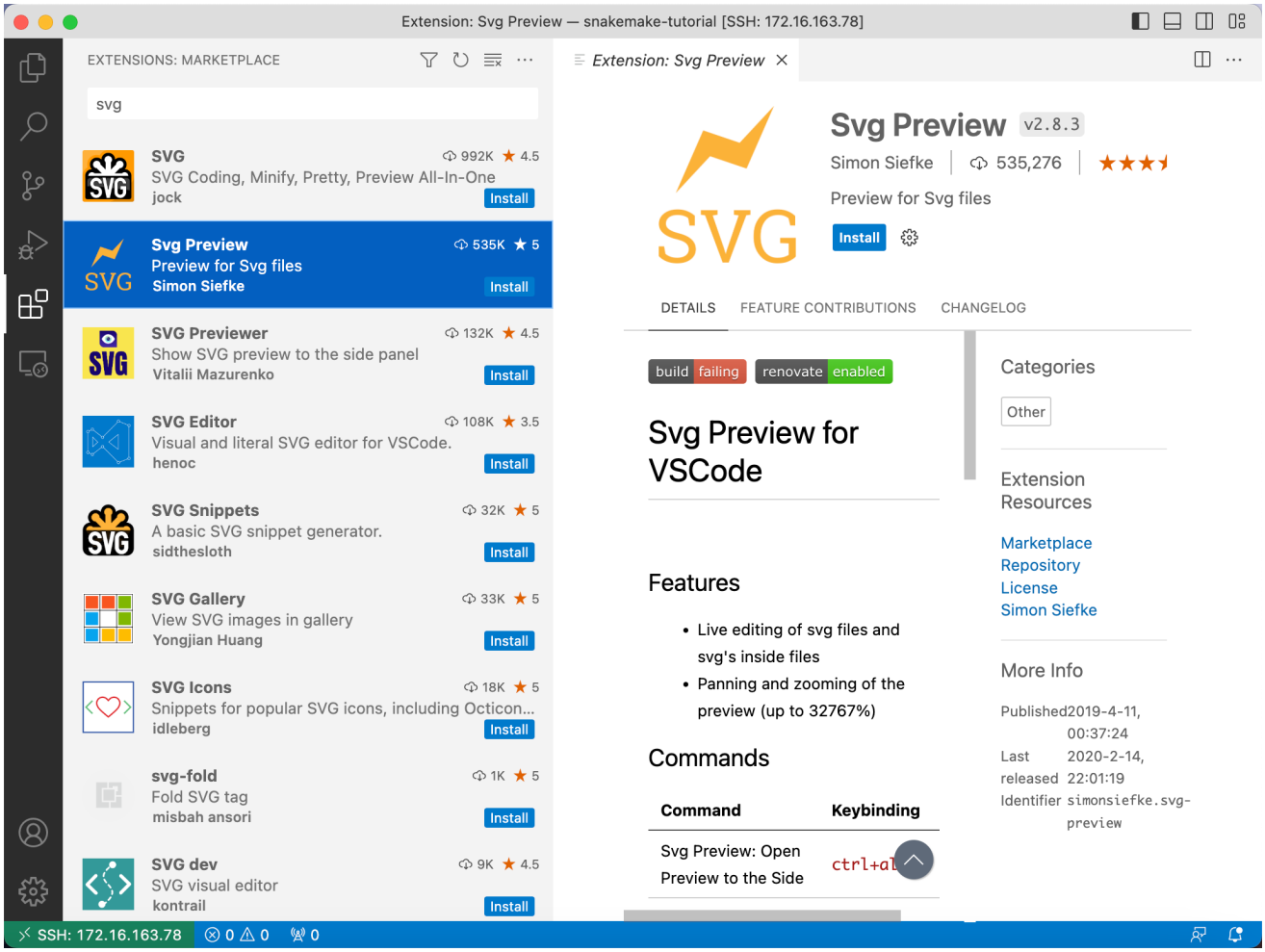


安装插件

- snakemake语法高亮

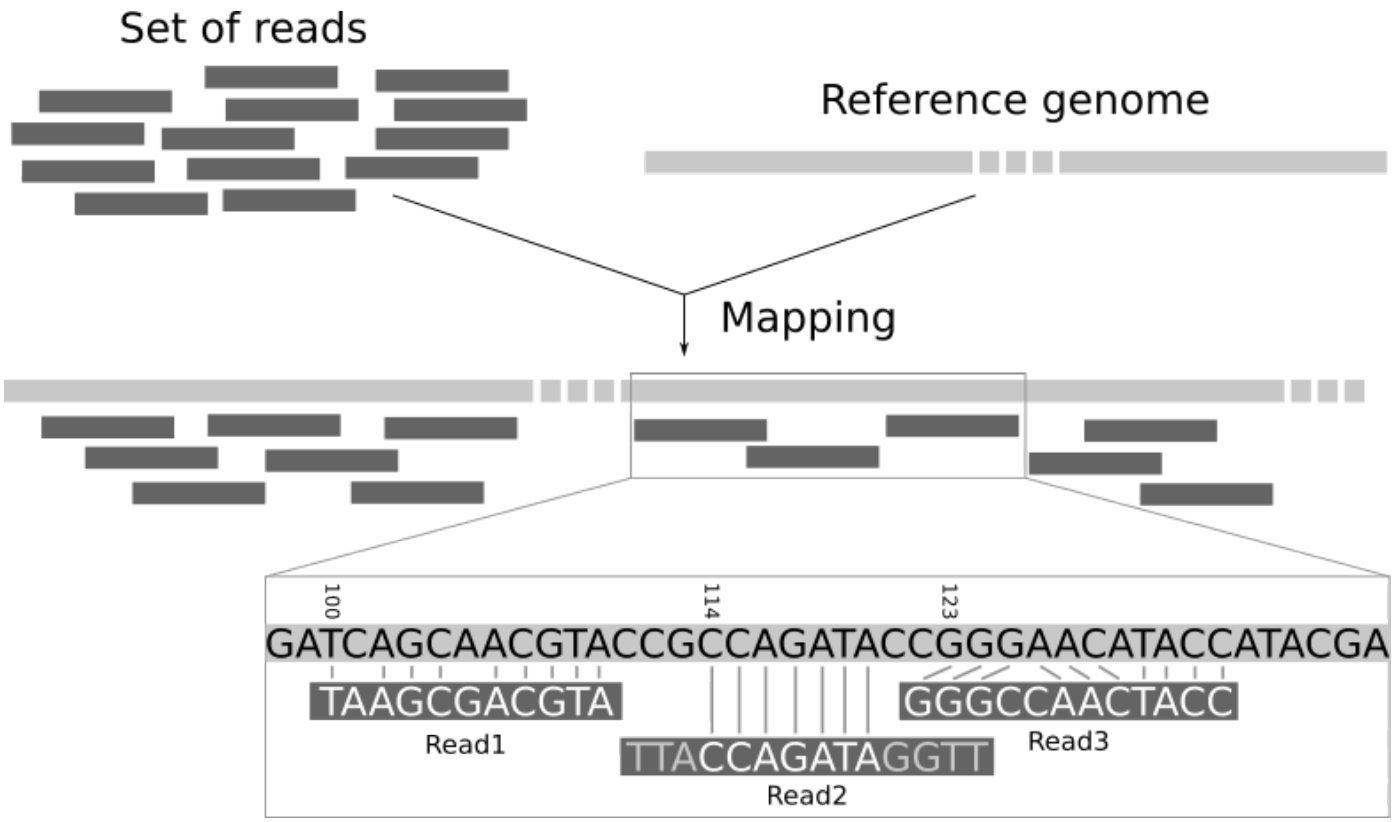

Extension: Snakemake Language — snakemake-tutorial [SSH: 172.16.163.78]
EXTENSIONS: MARKETPLACE
snakemake
Snakemake Language
Basic syntax, language, and snippet support for S...
Snakemake
Snakemake Language - DEPRECATED
4K ★ 4
Basic syntax, language, and snippet support for S...
alping
Install
Snakefmt
3K ★ 5
Linting and formatting support for snakemake files...
tfehlmann
Install
kkk.smk
Extension: Snakemake Language
Snakemake Language
Snakemake | 16,522 | ☆☆☆☆☆
Basic syntax, language, and snippet sup...
Disable
Uninstall
This extension is enabled globally.
DETAILS
FEATURE CONTRIBUTIONS
CHANGELOG
RUNTIME STATUS
Snakemake Language Support
Provides basic language support for Snakemake files (Snakefile, *.smk). Feedback, suggestions, and contributions are very welcome!
This project has been started by Peter Alping, and can be considered a fork of this repository.
Features
Syntax definitions based on Python, with added Snakemake keywords
Language rules based on Python
Snippets
Categories
Programming Languages
Extension Resources
Marketplace
Repository
License
Snakemake
More Info
Published 2021-5-21, 17:28:39
Last released 2022-3-30, 22:44:58
Last updated 2023-8-27, 13:32:25
Identifier snakemake.snakemake-lang
SSH: 172.16.163.78
0 0
0

- SVG查看器



3. snakemake的语法规则

3.1 语法规则初探——以SNP计算为例



3.1.1 基础规则

单个样本

- 新建一个名为`mapping.smk`的snakemake文件

养成在文件头部添加说明注释的习惯

```
"""
name: Calculate SNP
date: 2023-08-27
description: Snakemake pipeline to calculate SNP.
author: Jianmin Xie
dependencies:
    - snakemake-minimal >=7.3
    - jinja2
    - matplotlib
    - graphviz
    - bcftools =1.15
    - samtools =1.15
    - bwa =0.7.17
    - pysam =0.19
    - pygments


rules:
    - bwa_map

"""
```

- 原始命令

```
bwa mem data/genome.fa data/samples/A.fastq|samtools view -Sb - > mapped_reads/A.bam
```

```
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/A.fastq"
    output:
        "mapped_reads/A.bam"
    shell:
        "bwa mem {input} | samtools view -Sb - > {output}"

# samtool 参数
# -S 自动检测输入格式
# -b 输出bam格式
```

- 试运行

不指定流程文件的话会自动寻找当前目录有没有：`Snakefile`, `snakefile`, `workflow/Snakefile`, `workflow/snakefile`

```
# -s snakemake流程文件
# -n 试运行，只输出运行计划
# -p 输出详细运行哪些命令
snakemake -s mapping.smk -np mapped_reads/A.bam
```

- 真运行

Snakemake会自动创建缺失的目录，然后再执行任务。

```
# --cores, -c 使用cpu核数，真正运行必须指定这个参数，不写后面的个数将使用全部cpu
snakemake -s mapping.smk --cores 1 -p mapped_reads/A.bam
snakemake -s mapping.smk -c 1 -p mapped_reads/A.bam
```

Snakemake会重新运行如下情况的作业：

1）当输入文件中的某一个文件比输出文件中的某一个文件更新

2）或者输入文件中的某一个文件将会被另一个作业更新时

3）流程代码被修改

通配符匹配多个样本

- 新建一个名为`mapping-1.smk`的snakemake文件

```
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        "mapped_reads/{sample}.bam"
    shell:
        "bwa mem {input} | samtools view -Sb - > {output}"
```

- 试运行

```
# snakemake会根据rule里面output的通配符位置和你的目标文件路径进行匹配
snakemake -s mapping-1.smk -np mapped_reads/B.bam
# 指定多个目标文件
snakemake -s mapping-1.smk -np mapped_reads/A.bam mapped_reads/B.bam
# or
snakemake -s mapping-1.smk -np mapped_reads/{A,B}.bam
```

可以看到snakemake并没有输出`mapped_reads/A.bam`的信息，因为我们上一步已经生成了。我们可以更新一下输入文件的修改日期：

```
# 更新修改日期
touch data/samples/A.fastq

snakemake -s mapping-1.smk -np mapped_reads/{A,B}.bam
```

追加下一个rule对bam文件进行排序

- 在`mapping-1.smk`下面追加如下代码

```
rule samtools_sort:
    input:
        "mapped_reads/{sample}.bam"
    output:
        "sorted_reads/{sample}.bam"
    shell:
        "samtools sort -T sorted_reads/{wildcards.sample} "
        "-O bam {input} > {output}"
```

Snakemake允许通过`wildcards`对象访问shell命令中的通配符，该对象具有每个通配符的值作为属性。

`shell`声明符下面允许使用多行的引号，Python会自动拼接起来。

- 试运行

```
snakemake -s mapping-1.smk -np sorted_reads/{A,B}.bam
```

追加下一个rule对bam文件构建索引

- 在`mapping-1.smk`下面追加如下代码

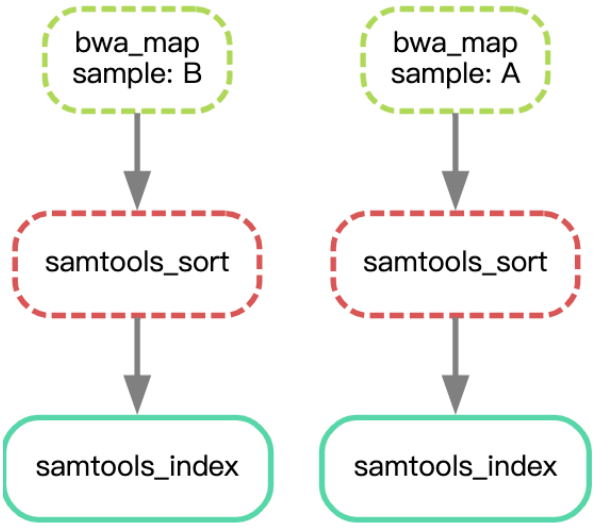
```
rule samtools_index:
    input:
        "sorted_reads/{sample}.bam"
    output:
        "sorted_reads/{sample}.bam.bai"
    shell:
        "samtools index {input}"
```

- 试运行

```
snakemake -s mapping-1.smk -np sorted_reads/{A,B}.bam.bai
```

- 流程可视化

```
# --dag 不要执行任何操作，打印用dot语言表示的作业有向无环图。
snakemake -s mapping-1.smk --dag sorted_reads/{A,B}.bam.bai | dot -Tsvg > dag.svg
```



3.1.2 通配符生成多样本列表及流程可视化

Snakemake提供了一个收集输入文件的函数：

```
expand("sorted_reads/{sample}.bam", sample=SAMPLES)
```

根据给定的模式`"sorted_reads/{sample}.bam"`，获取了一个文件列表，其中已经使用给定样本列表`SAMPLES`中的值进行了格式化，即

```
["sorted_reads/A.bam", "sorted_reads/B.bam"]
```

该函数在模式中包含多个通配符时特别有用。例如，

```
expand("sorted_reads/{sample}.{replicate}.bam", sample=SAMPLES, replicate=[0, 1])
```

将`SAMPLES`中与列表`[0, 1]`的所有元素相互组合，得到的结果是：

```
["sorted_reads/A.0.bam", "sorted_reads/A.1.bam", "sorted_reads/B.0.bam",
"sorted_reads/B.1.bam"]
```

追加下一个rule来计算所有样本多SNP信息

- 在`mapping-1.smk`下面追加如下代码

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

rule bcftools_call:
    input:
        fa="data/genome.fa",
        bam=expand("sorted_reads/{sample}.bam", sample=SAMPLES),
        bai=expand("sorted_reads/{sample}.bam.bai", sample=SAMPLES) # 虽然下面的命令参数没有用到，但是要指定这个目标输入上面的samtools_index才会执行
    output:
        "calls/all.vcf"
    shell:
        "bcftools mpileup -f {input.fa} {input.bam} | "
        "bcftools call -mv - > {output}"
```

有时，使用多个输入或输出文件时，在Shell命令中单独引用它们很方便。这可以通过为输入或输出文件指定名称来完成，例如：`fa=...`

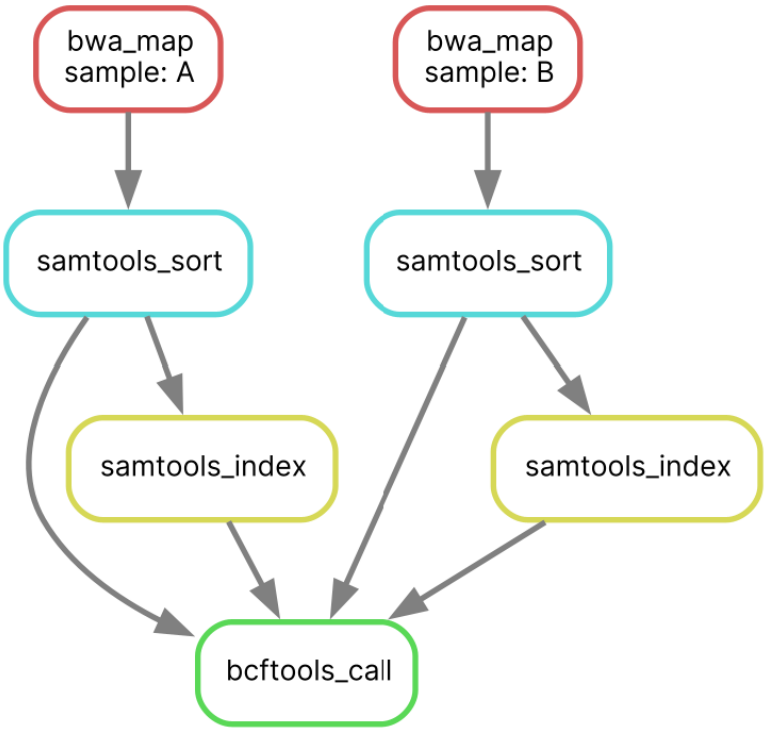
- 试运行

vcf格式介绍：链接1、链接2、How is the GT field in a VCF file defined?

```
snakemake -s mapping-1.smk -np calls/all.vcf
```

- 流程可视化

```
snakemake -s mapping-1.smk --dag calls/all.vcf | dot -Tsvg > dag.svg
```



3.1.3 运行外部python脚本

通常，工作流不仅包含调用各种工具，还包含自定义代码，例如计算汇总统计信息或创建图表。虽然Snakemake允许您在规则中直接编写Python代码，但通常将这样的逻辑移动到单独的脚本中是合理的。为此，Snakemake提供了脚本指令。将以下规则添加到`mapping-1.smk`中：

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

- 创建该python脚本`plot-quals.py`

```
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from pysam import VariantFile

quals = [record.qual for record in VariantFile(snakemake.input[0])]
plt.hist(quals)

plt.savefig(snakemake.output[0])
```

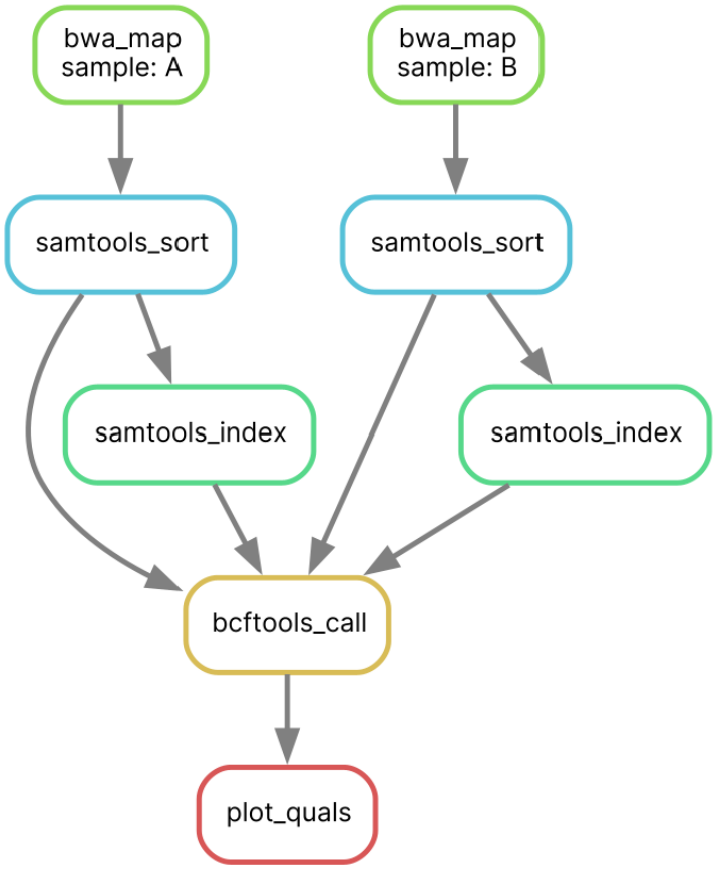
在脚本中，`rule`的所有属性，例如`input`、`output`、通配符等，都作为全局`snakemake`对象的属性可用。

- 试运行

```
snakemake -s mapping-1.smk -np plots/quals.svg
```

- 流程可视化

```
snakemake -s mapping-1.smk --dag plots/quals.svg | dot -Tsvg > dag.svg
```



以命令行传参数的方式运行外部脚本

缺点是参数太多时比较繁琐

```
rule plot_quals:
    input:
        "calls/all.vcf"
    output:
        "plots/quals.svg"
    shell:
        "python scripts/plot-quals-param.py -i {input} -o {output}"
```

- 创建该python脚本`plot-quals-param.py`

```
import optparse

import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from pysam import VariantFile
def plot(options):
    quals = [record.qual for record in VariantFile(options.input)]
    plt.hist(quals)
    plt.savefig(options.output)

if __name__ == '__main__':
```

```
    parser = optparse.OptionParser()  ## 写入上面定义的帮助信息
    parser.add_option('-i', '--input', dest='input', type='string', help='input path')
    parser.add_option('-o', '--output', dest='output', type='string', help='output dir')
    options, args = parser.parse_args()
    plot(options)
```

3.1.4 添加目标文件的总rule及可视化

如果在命令行中没有指定目标文件，Snakemake将定义Snakefile的第一个规则为目标。因此，最好的做法是在工作流的顶部拥有一个all规则，该规则具有所有通常所需的目标文件作为输入文件。

- 在`mapping-1.smk`顶部添加如下代码

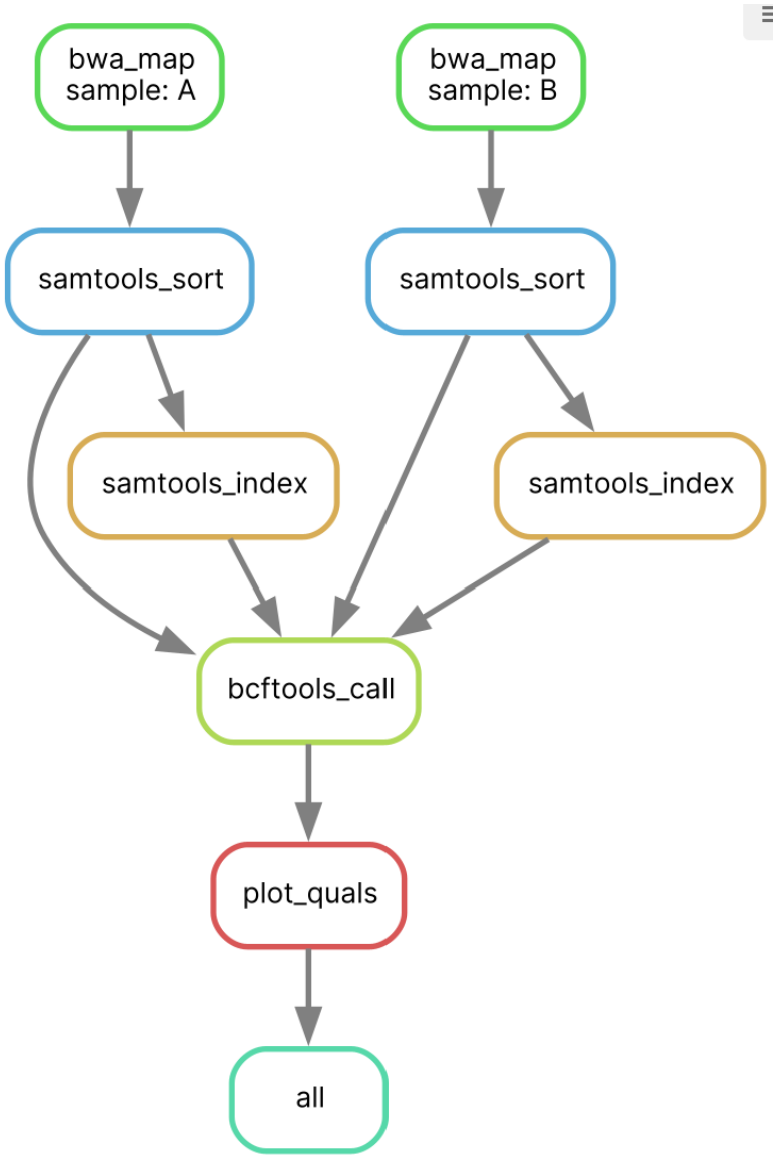
```
rule all:
    input:
        "plots/quals.svg"
```

- 试运行

```
# 不需要再指定目标文件了
snakemake -s mapping-1.smk -np
```

- 可视化

```
# 不需要再指定目标文件了
snakemake -s mapping-1.smk --dag | dot -Tsvg > dag.svg
```

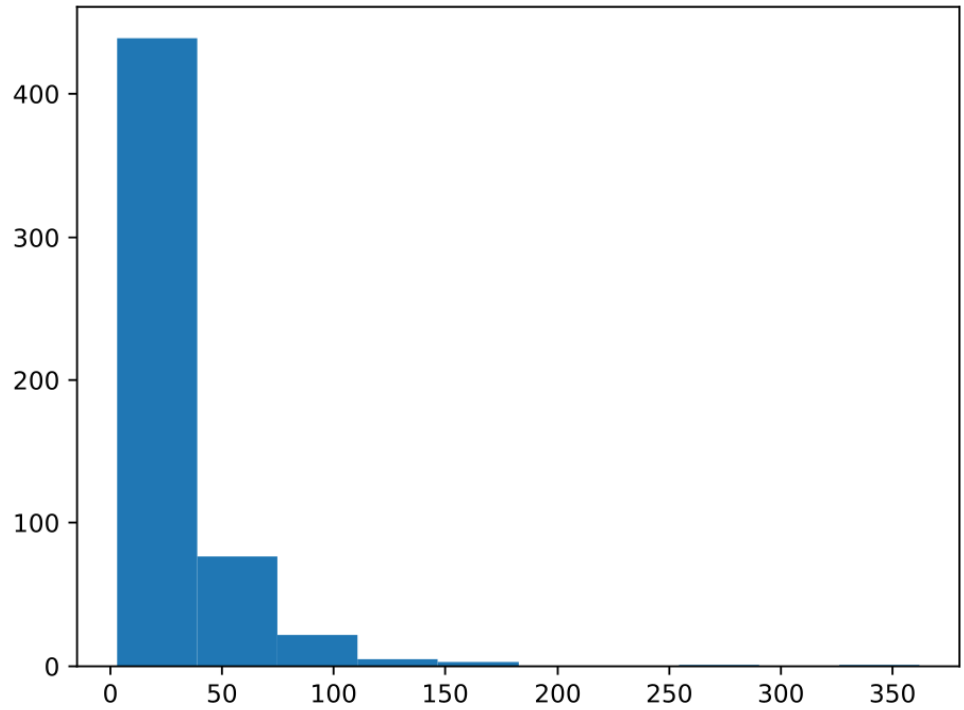


- 真运行

```
snakemake -s mapping-1.smk --cores 2 -p
```

- 每个突变位置的计算质量分频率分布直方图`quals.svg`

QUAL列给出了对ALT中的整体质量得分，即判断该位点是否为变异位点。



3.2 进阶功能

3.2.1 指定线程

- 新建一个名为`mapping-2.smk`的snakemake文件

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

rule all:
    input:
        expand("mapped_reads/{sample}.bam", sample=SAMPLES)

rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        "mapped_reads/{sample}.bam"
    threads: 8
    shell:
        "bwa mem -t {threads} {input} | samtools view -Sb - > {output}"
```

- 试运行

```
# 虽然指定了8线程，但是只会分配2线程
snakemake -s mapping-2.smk --cores 2 -np
# 线程足够，会分配8线程并行运行两个任务，有剩余的线程会分配给下游任务
snakemake -s mapping-2.smk --cores 20 -np
```

如果使用`--cores`而没有指定具体的数字，则会使用所有可用的核心。

使用 `--forceall` 参数，可以强制执行工作流的完整重新执行。将此参数与不同的 `--cores` 值结合使用，并检查调度程序如何选择要并行运行的作业。

3.2.2 使用外部配置文件

优点是不同数据不需要更改snakemake流程文件的源码，缺点是需要编写不同的配置文件

- 创建一个`config.yaml`文件

yaml语法介绍：链接

- `config/config.yaml`

```yaml
samples:
    A: data/samples/A.fastq
    B: data/samples/B.fastq
```

- 修改`mapping-2.smk`的代码为：

```
configfile: "config/config.yaml"

rule all:
    input:
        expand("mapped_reads/{sample}.bam", sample=config["samples"].keys())

def get_bwa_map_input_fastqs(wildcards):
    return config["samples"][wildcards.sample]

rule bwa_map:
    input:
        "data/genome.fa",
        get_bwa_map_input_fastqs
    output:
        "mapped_reads/{sample}.bam"
    threads: 8
    shell:
        "bwa mem -t {threads} {input} | samtools view -Sb - > {output}"
```

3.2.3 使用os.listdir()来读取样本

👉最推荐的读取样本方式！✅

- `config/config.yaml`

```yaml
samples_path: data/samples
suffix: fastq
```

```
configfile: "config/config.yaml"
import os
from os import path

samples_path = config["samples_path"]
# SAMPLES = [i.replace(".fastq","") for i in os.listdir(samples_path)]
SAMPLES = [".".join(i.split(".")[0:-1]) for i in os.listdir(samples_path)]

def get_bwa_map_input_fastqs(wildcards):
    fq_path = path.join(samples_path,f"{wildcards.sample}.{config['suffix']}")
    return fq_path

rule all:
    input:
        expand("mapped_reads/{sample}.bam", sample=SAMPLES)

rule bwa_map:
    input:
        "data/genome.fa",
        get_bwa_map_input_fastqs
    output:
        "mapped_reads/{sample}.bam"
    threads: 8
    shell:
        "bwa mem -t {threads} {input} | samtools view -Sb - > {output}"


```

3.2.4 rule 函数

- rule里面的`input`、`output`、`params`等都可以传入一个函数或者匿名函数，函数参数名是固定写法!

```
def get_bwa_map_input_fastqs(wildcards):
    return config["samples"][wildcards.sample]

rule bwa_map:
    input:
        "data/genome.fa",
        get_bwa_map_input_fastqs
    output:
        "mapped_reads/{sample}.bam"
    threads: 8
    shell:
        "bwa mem -t {threads} {input} | samtools view -Sb - > {output}"

# 等同于下面
rule bwa_map:
```

```
    input:
        "data/genome.fa",
        lambda wildcards: config["samples"][wildcards.sample]
    output:
        "mapped_reads/{sample}.bam"
    threads: 8
    shell:
        "bwa mem -t {threads} {input} | samtools view -Sb - > {output}"
```

3.2.5 rule 参数

有时，shell命令不仅由输入和输出文件以及一些静态参数组成。特别是，根据作业的通配符值，可能需要设置其他参数。为此，Snakemake允许使用params指令为规则定义任意参数。

- 修改 read group header line

```
import os
SAMPLES = [i.replace(".fastq","") for i in os.listdir("data/samples")]

rule all:
    input:
        expand("mapped_reads/{sample}.bam", sample=SAMPLES)

rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        "mapped_reads/{sample}.bam"
    params:
        rg=r"@RG\tID:{sample}\tSM:{sample}"
    threads: 8
    shell:
        "bwa mem -R '{params.rg}' -t {threads} {input} | samtools view -Sb - > {output}"
```

会在`bwa mem -R '{params.rg}' -t {threads} {input}`这一步生成的SAM文件头部增加信息

```
@SQ     SN:I    LN:230218
@RG     ID:A    SM:A
@PG     ID:bwa  PN:bwa  VN:0.7.17-r1188 CL:bwa mem -R @RG\tID:A\tSM:A -t 8 data/genome.fa data/samples/A.fastq
```

函数支持多参数，input 和 output只支持一个wildcards参数

- `config/config.yaml`

```
samples_path: data/samples
suffix: fastq
```

```
configfile: "config/config.yaml"
import os
from os import path

samples_path = config["samples_path"]
SAMPLES = [".".join(i.split(".")[0:-1]) for i in os.listdir(samples_path)]

def get_bwa_map_input_fastqs(wildcards):
    fq_path = path.join(samples_path,f"{wildcards.sample}.{config['suffix']}")
    return fq_path


def test(wildcards, input, output, threads, resources):
    print(input, output, threads, resources)
    return fr"@RG\tID:{wildcards.sample}\tSM:{wildcards.sample}"

rule all:
    input:
        expand("mapped_reads/{sample}.bam", sample=SAMPLES)
rule bwa_map:
    input:
        "data/genome.fa",
        get_bwa_map_input_fastqs
    output:
        "mapped_reads/{sample}.bam"
    params:
        rg = test
    threads: 8
    shell:
        "bwa mem -R '{params.rg}' -t {threads} {input} | samtools view -Sb - > {output}"
```

```
# 输出如下
data/genome.fa data/samples/A.fastq mapped_reads/A.bam 8 8 1 /tmp
data/genome.fa data/samples/B.fastq mapped_reads/B.bam 8 8 1 /tmp
```

3.2.6 日志

```
import os
SAMPLES = [i.replace(".fastq","") for i in os.listdir("data/samples")]

rule all:
```

```
    input:
        expand("mapped_reads/{sample}.bam", sample=SAMPLES)

rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        "mapped_reads/{sample}.bam"
    params:
        rg=r"@RG\tID:{sample}\tSM:{sample}"
    log:
        "logs/bwa_mem/{sample}.log"
    threads: 8
    shell:
        "(bwa mem -R '{params.rg}' -t {threads} {input} | "
        "samtools view -Sb - > {output}) 2> {log}"
```

`2>`标准错误输出重定向

3.2.7 标记临时文件和保护文件

在我们的工作流程中，我们为每个样本创建两个BAM文件，即规则`bwa_map`和`samtools_sort`的输出。当我们分析真实数据时，底层数据通常非常庞大。因此，生成的BAM文件需要大量的磁盘空间，并且它们的创建需要一些时间。为了节省磁盘空间，您可以将输出文件标记为临时文件。一旦所有需要它作为输入的消耗性作业都已执行，Snakemake将删除标记的文件。

通过read mapping和排序创建BAM文件是一项计算成本高昂的工作，因此保护最终的BAM文件免受意外删除或修改是有必要的。

```
import os
SAMPLES = [i.replace(".fastq","") for i in os.listdir("data/samples")]

rule all:
    input:
        expand("sorted_reads/{sample}.bam", sample=SAMPLES)
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        temp("mapped_reads/{sample}.bam")
    params:
        rg=r"@RG\tID:{sample}\tSM:{sample}"
    log:
        "logs/bwa_mem/{sample}.log"
    threads: 8
    shell:
        "(bwa mem -R '{params.rg}' -t {threads} {input} | "
```

```
        "samtools view -Sb - > {output}) 2> {log}"

rule samtools_sort:
    input:
        "mapped_reads/{sample}.bam"
    output:
        protected("sorted_reads/{sample}.bam")
    shell:
        "samtools sort -T sorted_reads/{wildcards.sample} "
        "-O bam {input} > {output}"
```

```
$ ll sorted_reads/
# 没有写权限
total 4.2M
-r--r--r-- 1 kkk kkk 2.1M 8月  27 22:11 B.bam
-r--r--r-- 1 kkk kkk 2.1M 8月  27 22:11 C.bam

```

3.2.8 使用全局变量rules

```
import os
SAMPLES = [i.replace(".fastq","") for i in os.listdir("data/samples")]

rule all:
    input:
        expand("sorted_reads/{sample}.bam", sample=SAMPLES)
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        temp("mapped_reads/{sample}.bam")
    params:
        rg=r"@RG\tID:{sample}\tSM:{sample}"
    log:
        "logs/bwa_mem/{sample}.log"
    threads: 8
    shell:
        "(bwa mem -R '{params.rg}' -t {threads} {input} | "
        "samtools view -Sb - > {output}) 2> {log}"

rule samtools_sort:
    input:
        rules.bwa_map.output
    output:
        protected("sorted_reads/{sample}.bam")
    shell:
        "samtools sort -T sorted_reads/{wildcards.sample} "
```

```
        "-O bam {input} > {output}"
```

3.2.9 直接运行python代码

```
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from pysam import VariantFile

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


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


rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        "mapped_reads/{sample}.bam"
    shell:
        "bwa mem {input}|samtools view -Sb - > {output}"

rule samtools_sort:
    input:
        "mapped_reads/{sample}.bam"
    output:
        "sorted_reads/{sample}.bam"
    shell:
        "samtools sort -T sorted_reads/{wildcards.sample} "
        "-O bam {input} > {output}"

rule samtools_index:
    input:
        "sorted_reads/{sample}.bam"
    output:
        "sorted_reads/{sample}.bam.bai"
    shell:
        "samtools index {input}"

rule bcftools_call:
    input:
        fa="data/genome.fa",
        bam=expand("sorted_reads/{sample}.bam", sample=SAMPLES),
        bai=expand("sorted_reads/{sample}.bam.bai", sample=SAMPLES) # 虽然下面的命令参数没有用到，但是要指定这个目标输入上面的samtools_index才会执行
```

```
    output:
        "calls/all.vcf"
    shell:
        "bcftools mpileup -f {input.fa} {input.bam} | "
        "bcftools call -mv - > {output}"

rule plot_quals:
    input:
        "calls/all.vcf",
    output:
        "plots/quals.svg"
    params:
        kkk=90
    run:
        quals = [record.qual for record in VariantFile(input[0])]
        plt.hist(quals)
        plt.savefig(output[0])
```

- 遍历输出shell()输出结果的行

```


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


rule all:
    input:
        "calls/quals.txt"


rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        "mapped_reads/{sample}.bam"
    shell:
        "bwa mem {input}|samtools view -Sb - > {output}"

rule samtools_sort:
    input:
        "mapped_reads/{sample}.bam"
    output:
        "sorted_reads/{sample}.bam"
    shell:
        "samtools sort -T sorted_reads/{wildcards.sample} "
        "-O bam {input} > {output}"

rule samtools_index:
```

```
    input:
        "sorted_reads/{sample}.bam"
    output:
        "sorted_reads/{sample}.bam.bai"
    shell:
        "samtools index {input}"

rule bcftools_call:
    input:
        fa="data/genome.fa",
        bam=expand("sorted_reads/{sample}.bam", sample=SAMPLES),
        bai=expand("sorted_reads/{sample}.bam.bai", sample=SAMPLES) # 虽然下面的命令参数没有用到，但是要指定这个目标输入上面的samtools_index才会执行
    output:
        "calls/all.vcf"
    shell:
        "bcftools mpileup -f {input.fa} {input.bam} | "
        "bcftools call -mv - > {output}"

rule plot_quals:
    input:
        "calls/all.vcf",
    output:
        "calls/quals.txt"

    run:
        quals_list = []
        for line in shell(f'grep -v "#" {input[0]}', iterable=True):
            split_res = line.split("\t")
            print(split_res)
            qual = split_res[5]
            quals_list.append(qual)
        with open(output[0], "w") as f:
            f.write("\n".join(quals_list))


```

3.2.10 监听流程运行的起始、成功或者失败

```
import os
SAMPLES = [i.replace(".fastq","") for i in os.listdir("data/samples")]

onstart:
    print(">>>>>The pipeline has been initiated!")

onsuccess:
    print(">>>>>Your pipeline has been completed successfully!")

```

```
onerror:
    print(">>>>>Your process has encountered an error!")


rule all:
    input:
        expand("sorted_reads/{sample}.bam", sample=SAMPLES)
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        "mapped_reads/{sample}.bam"
    params:
        rg=r"@RG\tID:{sample}\tSM:{sample}"
    log:
        "logs/bwa_mem/{sample}.log"
    threads: 8
    shell:
        "(bwa mem -R '{params.rg}' -t {threads} {input} | "
        "samtools view -Sb - > {output}) 2> {log}"

rule samtools_sort:
    input:
        rules.bwa_map.output
    output:
        "sorted_reads/{sample}.bam"
    shell:
        "samtools sort -T sorted_reads/{wildcards.sample} "
        "-O bam {input} > {output}"


```

3.3 其他进阶特性

3.3.1 任务运行状态参数测量

- 官方文档：链接

`benchmark`指令需要一个字符串参数，用于指定存储基准测试结果的文件路径。与输出文件类似，路径可以包含通配符（必须与输出文件中的通配符相同）。当从规则分配的作业执行时，Snakemake会测量时钟时间和内存使用量（以MiB为单位），并以制表符分隔的格式将其存储在文件中。可以重复多次基准测试，以了解测量结果的可变性。可以通过在基准测试文件中添加注释来实现，例如使用`repeat("benchmarks/{sample}.bwa.benchmark.txt", 3)`指示Snakemake运行该作业三次。重复测量结果会作为制表符分隔的基准测试文件中的连续行出现。

```
import os
```

```
SAMPLES = [i.replace(".fastq","") for i in os.listdir("data/samples")]

rule all:
    input:
        expand("sorted_reads/{sample}.bam", sample=SAMPLES)
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        temp("mapped_reads/{sample}.bam")
    params:
        rg=r"@RG\tID:{sample}\tSM:{sample}"
    log:
        "logs/bwa_mem/{sample}.log"
    benchmark:
        "benchmarks/{sample}.bwa.benchmark.txt"
    threads: 8
    shell:
        "(bwa mem -R '{params.rg}' -t {threads} {input} | "
        "samtools view -Sb - > {output}) 2> {log}"

rule samtools_sort:
    input:
        rules.bwa_map.output
    output:
        "sorted_reads/{sample}.bam"
    shell:
        "samtools sort -T sorted_reads/{wildcards.sample} "
        "-O bam {input} > {output}"
```

- 测量结果

```
s    h:m:s    max_rss max_vms max_uss max_pss io_in    io_out   mean_load    cpu_time
0.3070   0:00:00 17.91    693.95   9.62     10.66    0.00     0.00     0.00     0.39
```

3.3.2 生成运行报告

在您的工作流程完成后，报告中包含的所有其他信息（例如运行时统计信息）都是在创建过程中自动收集的。这些统计信息是从存储在您的工作目录内的 .snakemake 目录中的元数据中获取的。

这个命令需要连接网络

- 查看全部流程的报告

```
snakemake -s mapping-2.smk --report report.html
```

- 查看某一规则或者某一目标文件的报告

```
snakemake -s thread.smk sorted_reads/A.bam --report report.html
```

3.3.3 模块化编程——include

可以在不同流程中引用同一个规则文件，另一个Snakefile及其所有规则都会被包含到当前文件中：

- `config/config_module.yaml`

```
samples:
    A: data/samples/A.fastq
    B: data/samples/B.fastq

ref_genome: data/genome.fa
```

- `bwa_map_module.smk`

```
rule bwa_map:
    input:
        config["ref_genome"],
        lambda wildcards: config["samples"][wildcards.sample]
    output:
        temp("mapped_reads/{sample}.bam")
    params:
        rg=r"@RG\tID:{sample}\tSM:{sample}"
    log:
        "logs/bwa_mem/{sample}.log"
    threads: 8
    shell:
        "(bwa mem -R '{params.rg}' -t {threads} {input} | "
        "samtools view -Sb - > {output}) 2> {log}"
```

- `main-1.smk`

```
configfile: "config/config_module.yaml"
include: "bwa_map_module.smk"
rule all:
    input:
        expand("sorted_reads/{sample}.bam", sample=config["samples"].keys())

rule samtools_sort:
    input:
        rules.bwa_map.output
    output:
        "sorted_reads/{sample}.bam"
    shell:
        "samtools sort -T sorted_reads/{wildcards.sample} "
```

```
        "-O bam {input} > {output}"
```

- `main-2.smk`

```
configfile: "config/config_module.yaml"
include: "bwa_map_module.smk"
rule all:
    input:
        expand("sorted_reads/{sample}.bam", sample=config["samples"].keys())

rule samtools_sort:
    input:
        rules.bwa_map.output
    output:
        "sorted_reads/{sample}.sam"
    shell:
        "samtools view -S {input} > {output}"
```

3.3.4 模块化编程——module

可以导入部分rule，并且传人自定义的配置信息，例如基因组组装、转录组分析都需要对测序序列进行质控，可以把质控规则单独存到一个smk文件为一个模块

- 官方文档：链接
- `config/config_module.yaml`

```
samples:
    A: data/samples/A.fastq
    B: data/samples/B.fastq

ref_genome: data/genome.fa
```

- `multi_module.smk`

```
rule bwa_map:
    input:
        config["ref_genome"],
        lambda wildcards: config["samples"][wildcards.sample]
    output:
        temp("mapped_reads/{sample}.bam")
    params:
        rg=r"@RG\tID:{sample}\tSM:{sample}"
    log:
        "logs/bwa_mem/{sample}.log"
    threads: 8
    shell:
        "(bwa mem -R '{params.rg}' -t {threads} {input} | "
        "samtools view -Sb - > {output}) 2> {log}"
```

```
15
16
17 rule samtools_index:
18     input:
19         "sorted_reads/{sample}.bam"
20     output:
21         "sorted_reads/{sample}.bam.bai"
22     shell:
23         "samtools index {input}"
```

- `main-1.smk`

`default_target: True` 设置为默认目标文件规则

```
1  configfile: "config/config_module.yaml"
2  main_config = {
3      "samples":{
4          "A":"data/samples/A.fastq",
5          "B":"data/samples/B.fastq"
6      },
7      "ref_genome":"data/genome.fa"
8
9  }
10 module bwa:
11     snakefile: "multi_module.smk"
12     config: main_config
13
14
15 use rule * from bwa as my_*
16
17 rule all:
18     input:
19         expand("sorted_reads/{sample}.bam.bai", sample=config["samples"].keys())
20     default_target: True
21
22
23 rule samtools_sort:
24     input:
25         rules.my_bwa_map.output
26     output:
27         "sorted_reads/{sample}.bam"
28     shell:
29         "samtools sort -T sorted_reads/{wildcards.sample} "
30         "-O bam {input} > {output}"
31
32
33
```

- `main-2.smk`

`default_target: True` 设置为默认目标文件规则

```
configfile: "config/config_module.yaml"
main_config = {
    "samples":{
        "A":"data/samples/A.fastq",
        "B":"data/samples/B.fastq"
    },
    "ref_genome":"data/genome.fa"

}
module bwa:
    snakefile: "multi_module.smk"
    config: main_config


use rule bwa_map from bwa as my_bwa_map

rule all:
    input:
        expand("sorted_reads/{sample}.bam", sample=config["samples"].keys())
    default_target: True


rule samtools_sort:
    input:
        rules.my_bwa_map.output
    output:
        "sorted_reads/{sample}.bam"
    shell:
        "samtools sort -T sorted_reads/{wildcards.sample} "
        "-O bam {input} > {output}"

```

- 模块的继承——继承后面会讲

不必使用所有规则，可以导入特定规则。甚至可以在使用它们之前修改特定规则，通过最终的`with:`后跟一个列出要覆盖的项目的块。此修改可以在一般导入之后执行，并将覆盖任何未修改的相同规则的导入。

```
from snakemake.utils import min_version
min_version("6.0")

module other_workflow:
    # here, plain paths, URLs and the special markers for code hosting providers (see below) are possible.
    snakefile: "other_workflow/Snakefile"
    config: config["other-workflow"]

use rule * from other_workflow as other_*

use rule some_task from other_workflow as other_some_task with:
    output:
        "results/some-result.txt"
```

3.3.5 使用独立运行环境并自动部署软件依赖

为了获得完全可重复的数据分析，仅仅能够执行每个步骤并记录所有使用的参数是不够的。使用的软件工具和库也必须被记录。在本教程中，您已经看到了如何使用Conda为整个工作流程指定一个隔离的软件环境。通过Snakemake，您可以更进一步，为每个规则指定Conda环境。这样，您甚至可以使用冲突的软件版本（例如将Python 2与Python 3结合使用）。

- `envs/samtools.yaml`

```
channels:
  - bioconda
  - conda-forge
dependencies:
  - samtools =1.9
```

- `individual_env.smk`

```
import os
SAMPLES = [i.replace(".fastq","") for i in os.listdir("data/samples")]

rule all:
    input:
        expand("sorted_reads/{sample}.bam", sample=SAMPLES)
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        temp("mapped_reads/{sample}.bam")
    params:
        rg=r"@RG\tID:{sample}\tSM:{sample}"
```

```
    log:
        "logs/bwa_mem/{sample}.log"
    benchmark:
        "benchmarks/{sample}.bwa.benchmark.txt"
    threads: 8
    shell:
        "(bwa mem -R '{params.rg}' -t {threads} {input} | "
        "samtools view -Sb - > {output}) 2> {log}"

rule samtools_sort:
    input:
        rules.bwa_map.output
    output:
        "sorted_reads/{sample}.bam"
    conda:
        "envs/samtools.yaml"
    shell:
        "samtools sort -T sorted_reads/{wildcards.sample} "
        "-O bam {input} > {output}"
```

- 运行

它将在作业执行之前自动创建所需的环境并激活它们。最佳实践是在环境定义中至少指定软件包的主要和次要版本。以这种方式为每个规则指定环境有两个优点。首先，工作流定义还记录了所有使用的软件版本。其次，可以在纯净系统上重新执行工作流（无需管理员权限），而无需安装除Snakemake和Miniconda之外的任何先决条件。

安装环境后运行

```
snakemake -s individual_env.smk --cores 20 --use-conda -p
```

只安装环境不运行

```
snakemake -s individual_env.smk --cores 20 --use-conda --conda-create-envs-only
```

- 环境保存位置：`.snakemake/conda/e8e8320a738e03d4611acae104c742bc_`

```
$ snakemake -s individual_env.smk -np -c 20  --list-conda-envs
Building DAG of jobs...
environment container   location
envs/samtools.yaml      .snakemake/conda/e8e8320a738e03d4611acae104c742bc_
```

- 使用已有的conda环境

有时候，从一个已经存在的命名的conda环境中引用规则可能很方便，而不是从头开始定义一个新的环境。缺点是这可能会影响可重复性，因为工作流程依赖于此环境在任何执行工作流程的新系统上以完全相同的方式存在。在这种情况下，你必须手动处理这个问题。因此，上面描述的使用环境定义文件的方法是高度推荐和首选的。

```
import os
SAMPLES = [i.replace(".fastq","") for i in os.listdir("data/samples")]

rule all:
    input:
        expand("sorted_reads/{sample}.bam", sample=SAMPLES)
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        temp("mapped_reads/{sample}.bam")
    params:
        rg=r"@RG\tID:{sample}\tSM:{sample}"
    log:
        "logs/bwa_mem/{sample}.log"
    benchmark:
        "benchmarks/{sample}.bwa.benchmark.txt"
    threads: 8
    shell:
        "(bwa mem -R '{params.rg}' -t {threads} {input} | "
        "samtools view -Sb - > {output}) 2> {log}"

rule samtools_sort:
    input:
        rules.bwa_map.output
    output:
        "sorted_reads/{sample}.bam"
    conda:
        "samtools_env"
    shell:
        "samtools sort -T sorted_reads/{wildcards.sample} -O bam {input} >
{output}"
```

3.3.6 通配符限制

用到了python的正则表达式

- 过滤样本，只运行B和C样本

```
import os
SAMPLES = [i.replace(".fastq","") for i in os.listdir("data/samples")]

```

```
rule all:
    input:
        expand("sorted_reads/{sample}.bam", sample=SAMPLES)
rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        temp("mapped_reads/{sample}.bam")
    params:
        rg=r"@RG\tID:{sample}\tSM:{sample}"
    log:
        "logs/bwa_mem/{sample}.log"
    benchmark:
        "benchmarks/{sample}.bwa.benchmark.txt"
    threads: 8
    shell:
        "(bwa mem -R '{params.rg}' -t {threads} {input} | "
        "samtools view -Sb - > {output}) 2> {log}"

rule samtools_sort:
    input:
        rules.bwa_map.output
    output:
        "sorted_reads/{sample,[BC]}.bam"
    shell:
        "samtools sort -T sorted_reads/{wildcards.sample} -O bam {input} >
{output}"
```

- 解决歧义

它可以帮助指导基于正则表达式的匹配，以便通配符分配到文件名的正确部分。考虑输出文件`{sample}.{group}.txt`，并假设目标文件是`A.1.normal.txt`。不清楚`dataset="A.1"`和`group="normal"`还是`dataset="A"`和`group="1.normal"`是正确的分配。在这里，通过`{sample, [A-Z]+}.{group}.txt`来约束数据集通配符解决了这个问题。

- 写在规则关键字`wildcard_constraints`里

```
rule complex_conversion:
    input:
        "{dataset}/inputfile"
    output:
        "{dataset}/file.{group}.txt"
    wildcard_constraints:
        dataset="\d+"
    shell:
        "somecommand --group {wildcards.group}  < {input}  > {output}"
```

你还可以定义适用于所有规则的全局通配符约束条件：

```
wildcard_constraints:
    dataset="\d+"

rule a:
    ...

rule b:
    ...
```

3.3.7 打印代码完善建议

- 文档：链接

```
$ snakemake -s mapping-2.smk --lint
Lints for rule bwa_map (line 14, /home/kkk/snakemake-tutorial/mapping-2.smk):
    * Specify a conda environment or container for each rule.:
      This way, the used software for each specific step is documented, and the workflow can be executed on any machine without prerequisites.
      Also see:

https://snakemake.readthedocs.io/en/latest/snakefiles/deployment.html#integrated-package-management

https://snakemake.readthedocs.io/en/latest/snakefiles/deployment.html#running-jobs-in-containers

Lints for rule samtools_sort (line 57, /home/kkk/snakemake-tutorial/mapping-2.smk):
    * No log directive defined:
      Without a log directive, all output will be printed to the terminal. In distributed environments, this means that errors are harder to discover. In local environments, output of concurrent jobs will be mixed and become unreadable.
      Also see:
      https://snakemake.readthedocs.io/en/stable/snakefiles/rules.html#log-files
    * Specify a conda environment or container for each rule.:
      This way, the used software for each specific step is documented, and the workflow can be executed on any machine without prerequisites.
      Also see:

https://snakemake.readthedocs.io/en/latest/snakefiles/deployment.html#integrated-package-management

https://snakemake.readthedocs.io/en/latest/snakefiles/deployment.html#running-jobs-in-containers

```

3.3.8 在服务器集群上运行

- 官方文档：Cluster execution

3.4 其他语法规则介绍

3.4.1 花括号歧义

当你的命令行本身包含花括号时，需要写成这样`{{}}`

- `text/test.txt`

```
1
2
3
4
5
6
```

- `read_text.smk`

```
shell.executable('/bin/zsh')
SAMPLES = [1,2,3,4,5,6]
rule all:
    input:
        expand("text/{sample}.txt",sample=SAMPLES)

rule read_text:
    input:
        "text/test.txt"
    output:
        "text/{sample}.txt"
    shell:
        "while {{read i}} {{echo $i > {output}}} < {input}"
```

```
rule pre_gff:
    input:
        prokka_output_dir + "/{sample}/{sample}.gff"
    output:
        gff_output_dir + "/{sample}.gff"
    shell:
        "grep '\CDS\s'  {input} | awk -F '[\t;=]' 'BEGIN{{OFS=\"\t\"}} {{print $1,$10,$4,$5}}' > {output}"

```

3.4.2 更改默认命令行执行shell

默认情况下，Shell命令将使用`bash` shell调用（除非工作流通过`shell.executable(...)`指定了不同的默认shell）。

- 使用zshell

```
shell.executable('/bin/zsh') # 放在文件顶部
rule move_genome:
    input:
        rgp=reference_genome_PATH,
        et=WORK_PATH + "/{sample}/exclude_taxa.txt" #把属于本clade的基因组排除
    output:
        directory(WORK_PATH + "/{sample}/genome")
    shell:
        "mkdir {output};"
        "ln -fs {input.rgp}/**/*.(fasta|fna) {output};"
        "while {{read i}} {{ unlink {output}/${{i}}.fasta }} < {input.et}"
```

3.4.3 expand 函数的更多用法

- 传入一个模板列表

```
DATASETS = ["ds1", "ds2"]
FORMATS = ["txt", "csv"]
expand(["{dataset}/a.{ext}", "{dataset}/b.{ext}"], dataset=DATASETS, ext=FORMATS)
```

生成如下

```
["ds1/a.txt", "ds1/b.txt", "ds2/a.txt", "ds2/b.txt", "ds1/a.csv", "ds1/b.csv",
"ds2/a.csv", "ds2/b.csv"]
```

默认情况下，`expand` 使用了 Python 的 itertools 函数 `product`，它会生成提供的通配符值的所有组合。然而，通过插入第二个位置参数，可以将其替换为任何组合函数，例如 `zip`：

```
DATASETS = ["ds1", "ds2"]
FORMATS = ["txt", "csv"]
expand(["{dataset}/a.{ext}", "{dataset}/b.{ext}"], zip, dataset=DATASETS,
ext=FORMATS)
```

生成如下

```
["ds1/a.txt", "ds1/b.txt", "ds2/a.csv", "ds2/b.csv"]
```

您还可以在 `expand` 中掩盖通配符表达式，使其保持不变，例如：

```
expand("{{dataset}}/a.{ext}", ext=FORMATS)
```

will create strings with all values for ext but starting with the wildcard `"{dataset}"`.

3.4.4 multiext 函数

`multiext`提供了`expand`的简化版本，允许我们定义一组仅通过其扩展名不同的输出或输入文件：

```
rule plot:
    input:
        ...
    output:
        multiext("some/plot", ".pdf", ".svg", ".png")
    shell:
        ...
```

效果与使用`expand("some/plot{ext}", ext=[".pdf", ".svg", ".png"])`写作是相同的，但使用了更简单的语法。此外，使用`multiext`定义输出是使用具有多个输出文件的规则进行工作流程缓存的唯一方法。

3.4.5 合理的使用线程

```
rule NAME:
    input: "path/to/inputfile", "path/to/other/inputfile"
    output: "path/to/outputfile", "path/to/another/outputfile"
    threads: workflow.cores * 0.75
    shell: "somecommand --threads {threads} {input} {output}"
```

给定核心的数量在Snakefile中作为工作流对象的属性全局可用：`workflow.cores`。可以执行任何算术操作以从中派生线程数。例如，在上面的示例中，我们为规则保留了给定核心的75%。Snakemake始终向下舍入计算出的值（同时强制执行最小1个线程）。

3.4.6 使用message关键字输出信息

执行snakemake时，控制台会为每个正在运行的规则提供简短的摘要。可以通过为规则指定消息来覆盖此摘要。

```
rule NAME:
    input: "path/to/inputfile", "path/to/other/inputfile"
    output: "path/to/outputfile", "path/to/another/outputfile"
    threads: 8
    message: "Executing somecommand with {threads} threads on the following files
{input}."
    shell: "somecommand --threads {threads} {input} {output}"
```

请注意，通过变量`wildcards`（例如`{wildcards.sample}`），也可以访问通配符，这与使用 shell 命令的方式相同。为了避免与其他变量名称冲突，必须在通配符周围设置命名空间。

3.4.7 运行其他外部脚本(R脚本)

- 官方文档：链接
- 安装读取vcf的R包，在环境搭建的时候已经有介绍

```
mamba install -c r r-vcfR
```

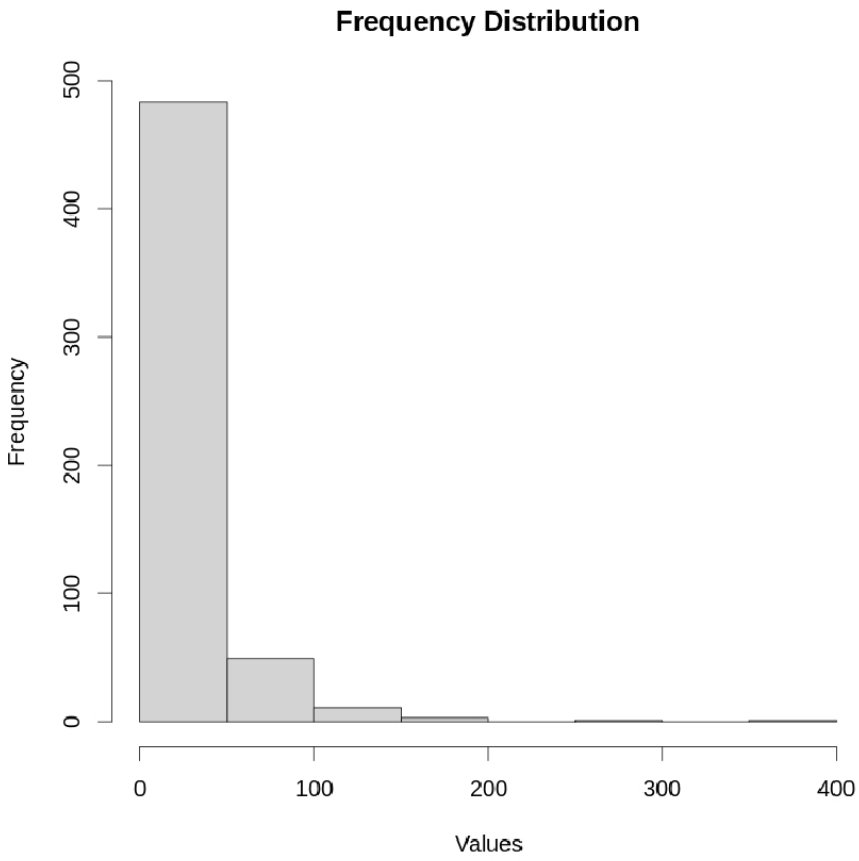
- 创建R文件`scripts/plot-quals.R`

```
library(vcfR)
do_plot <- function(vcf_path, out_path) {
# 读取VCF文件
vcf <- read.vcfR(vcf_path)
# 获取VCF文件中的变异信息
df_vcf <- as.data.frame(vcf@fix)
jpeg(filename = out_path, width = 800, height = 800, quality=100, pointsize=20)
hist(as.numeric(df_vcf$QUAL), main="Frequency Distribution", xlab="Values")
dev.off()
}

do_plot(snakemake@input[[1]], snakemake@output[[1]])
```

- 运行

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

rule all:
    input:
        "plots/quals.jpg"


rule bwa_map:
    input:
        "data/genome.fa",
        "data/samples/{sample}.fastq"
    output:
        "mapped_reads/{sample}.bam"
    shell:
        "bwa mem {input} | samtools view -Sb - > {output}"


rule samtools_sort:
    input:
        "mapped_reads/{sample}.bam"
    output:
        "sorted_reads/{sample}.bam"
    threads: 3
```

```
    message: "Executing somecommand with {threads} threads on the following files {input}."
    shell:
        "samtools sort -T sorted_reads/{wildcards.sample} "
        "-O bam {input} > {output}"


rule samtools_index:
    input:
        "sorted_reads/{sample}.bam"
    output:
        "sorted_reads/{sample}.bam.bai"
    shell:
        "samtools index {input}"


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


rule plot_quals:
    input:
        "calls/all.vcf"
    output:
        "plots/quals.jpg"
    script:
        "scripts/plot-quals.R"
```



- 使用独立的R环境

```
rule plot_quals:
    input:
        "calls/all.vcf"
    output:
        "plots/quals.jpg"
    conda:
        "envs/vcfR.yaml"
    script:
        "scripts/plot-quals.R"
```

`envs/vcfR.yaml`

```
channels:
 - r
dependencies:
 - r=4.2
 - r-vcfr=1.14.0
```

运行

```
snakemake -s mapping-2.smk --use-conda  --cores 20 -p
```

3.4.8 使用目录作为输出

```
rule spades:
    input:
        r1 = rules.fastp.output.r1,
        r2 = rules.fastp.output.r2,
    output:
        directory(SPADES_OUT_PATH + "/{sample}")
    threads: 16
    shell:
        "spades.py --isolate -1 {input.r1} -2 {input.r2} -t {threads} -m 48 -o
{output} &> /dev/null"
```

3.4.9 忽略输入文件的更新

为确定是否需要重新创建输出文件，Snakemake检查同一作业的任何输入文件的文件修改日期（即时间戳）是否比输出文件的时间戳更新。可以通过将输入文件标记为“古老的”来覆盖此行为。忽略这些文件的时间戳，并始终假定其比任何输出文件都要旧：

```
rule NAME:
    input:
        ancient("path/to/inputfile")
    output:
        "path/to/outputfile"
    shell:
        "somecommand {input} {output}"
```

这意味着一旦文件 `path/to/outputfile` 被生成，即使将来修改了输入文件，它也不会被触发重新创建。请注意，任何强制重新创建文件的标志也适用于被标记为“古老”的文件。

比如说我们改动了输入文件但是做的改动对下游任务没有产生影响，那么就可以利用这个减少计算量

3.4.10 确保输出文件的属性--非空或校验和合规性

在成功生成输出文件后，可以注释特定的附加标准以确保它们符合要求。例如，这可以用于检查输出文件是否非空，或将它们与给定的sha256校验和进行比较。如果使用此功能，Snakemake将在考虑作业成功之前检查此类带注释的文件。可以按以下方式检查非空性：

```
rule NAME:
    output:
        ensure("test.txt", non_empty=True)
    shell:
        "somecommand {output}"
```

上面，输出文件 `test.txt` 被标记为非空。如果命令 `somecommand` 生成一个空输出，作业将以错误形式失败，并列出意外的空文件。

sha256校验和可以按以下方式进行比较：

```
my_checksum = "u98a9cjsd98saud090923ßkpoasköf9ß32"

rule NAME:
    output:
        ensure("test.txt", sha256=my_checksum)
    shell:
        "somecommand {output}"
```

在下载文件的时候可以用

```
my_checksum = "u98a9cjsd98saud090923ßkpoasköf9ß32"
rule download:
    params:
        taxon="{sample}"
    output:
        ensure("Miniconda3-latest-Linux-x86_64.sh", sha256=my_checksum)
    shell:
        "wget -O Miniconda3-latest-Linux-x86_64.sh
https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh"
```

除了提供纯字符串的校验和外，还可以提供指向函数的指针（类似于输入函数）。该函数必须接受一个参数，该参数将是从应用规则生成的通配符对象，用于创建一些请求的输出文件：

```
def get_checksum(wildcards):
    # e.g., look up the checksum with the value of the wildcard sample
    # in some dictionary
    return my_checksums[wildcards.sample]

rule NAME:
    output:
        ensure("test/{sample}.txt", sha256=get_checksum)
    shell:
        "somecommand {output}"
```

请注意，您还可以使用lambda表达式而不是完整的函数定义。

通常，将`ensure`注释与重新运行结合使用是一个好主意，例如在检测到无效校验和或空文件时进行重试。

- 例子：双端测序fastq数据指控软件fastp

这个软件的特点是输出的fastq文件是一直在写入的，即使失败也会存在并且拥有一定文件大小，但是只有成功才会生成json和html文件，所以我们只要保证输出的html或者json文件不为空就说明运行成功。

```
rule fastp:
    input:
        r1 = reads_dir + "/{sample}/{sample}_1.fq.gz",
        r2 = reads_dir + "/{sample}/{sample}_2.fq.gz",
    output:
```

```
        r1 = FASTP_OUT_PATH + "/{sample}/{sample}_paired_1.fq.gz",
        r2 = FASTP_OUT_PATH + "/{sample}/{sample}_paired_2.fq.gz",
        json = ensure(FASTP_OUT_PATH + "/{sample}/{sample}_fastp.json", non_empty=True), # If the command somecommand happens to generate an empty output, the job will fail with an error listing the unexpected empty file.
        html = ensure(FASTP_OUT_PATH + "/{sample}/{sample}_fastp.html", non_empty=True), # Often, it is a good idea to combine ensure annotations with retry definitions, e.g. for retrying upon invalid checksums or empty files.
    log:
        FASTP_OUT_PATH + "/{sample}/{sample}_fastp.log"
    threads: 8
    shell:
        "fastp -i {input.r1} -I {input.r2} -o {output.r1} -O {output.r2} "
        "-j {output.json} -h {output.html} -w {threads} "
        "-q 20 " # the quality value that a base is qualified
        "-u 10 " # how many percents of bases are allowed to be unqualified (0~100).
        "-n 3 " # number of N allowed
        "-l 50 " # reads shorter than length_required will be discarded, particularly those with adapter.
        "-5 " # enable trimming in 5' ends.
        "-3 " # enable trimming in 3' ends.
        "&> {log}"
```

3.4.11 定义可能失败规则的重试次数

有时候，规则可能偶尔会失败。例如，当规则下载一些在线资源时，可能会发生这种情况。对于这种情况，可以通过“重试”指令为该特定规则的每个作业定义一定数量的自动重试次数：

这种适合需要用到网络的情况，我们平常的分析软件一般出错都是致命性的，得排查出错误才行，不是说你重跑就能成功的

```
rule a:
    output:
        "test.txt"
    retries: 3
    shell:
        "curl https://some.unreliable.server/test.txt > {output}"
```

```
my_checksum = "u98a9cjsd98saud090923ßkpoasköf9ß32"
rule a:
    output:
        ensure("test.txt", sha256=my_checksum)
    retries: 3
    shell:
        "curl https://some.unreliable.server/test.txt > {output}"
```

通常，将重试功能与确保注释结合使用是一个好主意，例如在无效校验和或空文件上重试。

请注意，也可以全局定义重试（通过`--retries`命令行选项，参见All Options）。规则的本地定义会覆盖全局定义。

重要的是，`retries`指令用于定义平台无关的行为（例如为上述下载命令添加鲁棒性）。对于处理不可靠的集群或云系统，应使用`--retries`命令行选项。

3.4.12 unpack()函数

在某些情况下，您可能希望使输入函数返回命名的输入文件。这可以通过让它们返回具有名称作为字典键和文件名作为字典值的`dict()`对象，并使用`unpack()`关键字来实现。

```
rule all:
    input:
        "someoutput.A.txt"

def myfunc(wildcards):
    return {'foo': f'{wildcards.sample}.txt'}

rule test:
    input:
        unpack(myfunc)
    output:
        "someoutput.{sample}.txt"
    shell:
        "cat {input.foo} > {output}"
```

请注意，`unpack()`仅适用于返回`dict`的输入函数。虽然它也适用于`list`，但请记住，字符串列表（包括嵌套列表）会自动展平。

还要注意，如果您在输入列表中没有传递一个*函数*，而是直接*调用函数*，则不应使用`unpack()`。在这种情况下，您可以简单地使用 Python 的双星号（`**`）运算符来展开参数。

请注意，由于 Snakefile 在执行时被转换为 Python，因此使用星号和双星号展开 Python 运算符时应遵循相同的规则。使用`unpack()`时不适用这些限制。

```
def myfunc1():
    return ['foo.txt']

def myfunc2():
    return {'foo': 'nowildcards.txt'}

rule:
    input:
        *myfunc1(),
        **myfunc2(),
    output:
        "..."
    shell:
```

```
        "..."
```

3.4.13 规则继承

从Snakemake 6.0及更高版本开始，可以从先前定义的规则继承，或者换句话说，在修改后重用现有规则。这通过`use rule`语句实现，该语句还允许声明使用来自外部模块的规则（请参见模块）。考虑以下示例：

```
rule all:
    input:
        "test.out",
        "test2.out"

rule a:
    output:
        "test.out"
    shell:
        "echo test > {output}"

use rule a as b with:
    output:
        "test2.out"
```

如上所示，我们首先声明一个规则a，然后我们将规则a作为规则b重复使用，只更改输出文件并保持其他所有内容不变。实际上，人们经常会更改更多。类似于外部模块的`use rule`，可以修改规则的任何属性（`input`，`output`，`log`，`params`，`benchmark`，`threads`，`resources`等），除了实际执行步骤（`shell`，`notebook`，`script`，`cwl`或`run`）。所有未修改的属性都从父规则继承。

- 示例：基因组多次校正，可以减少代码量

```
rule minimap2:
    input:
        genome=rules.flye.output,
        r1 = rules.fastp.output.r1,
        r2 = rules.fastp.output.r2,
    output:
        minimap2_OUT_PATH + "/{sample}/{sample}_aln.bam"
    threads: 10
    shell:
        "minimap2 -ax sr {input.genome} {input.r1} {input.r2} | samtools sort -@ {threads} -O bam -o {output};"
        "samtools index -@ {threads} {output};"

rule pilon:
    input:
        genome = rules.flye.output,
        bam = rules.minimap2.output
    output:
        pilon_OUT_PATH + "/{sample}/{sample}_pilon.fasta"
```

```
    params:
        prefix = "{sample}_pilon",
        outdir= pilon_OUT_PATH + "/{sample}",
        pilonJar = pilon_jar
    shell:
        "java -Xmx16G -jar {params.pilonJar} --genome {input.genome} --frags
{input.bam} --output {params.prefix} --fix all --mindepth 10 --outdir
{params.outdir} --changes --verbose"


use rule minimap2 as minimap2_2 with:
    input:
        genome = rules.pilon.output,
        r1= rules.fastp.output.r1,
        r2 = rules.fastp.output.r2,
    output:
        minimap2_OUT_PATH + "/{sample}/{sample}_aln_2.bam"
```

3.4.14 判断环境变量是否存在

有时候，将配置信息放入文本文件中是不可取的。例如，对于访问令牌或密码等机密信息来说是如此。在这种情况下，环境变量是首选的方法。Snakemake允许通过添加以下语句来确保环境变量的存在：

```
envvars:
    "SOME_VARIABLE",
    "SOME_OTHER_VARIABLE"
```

执行时，如果变量`SOME_VARIABLE`和`SOME_OTHER_VARIABLE`未定义，Snakemake会带有合理的错误信息失败。否则，它将负责将它们传递给群集和云环境。然而，需要注意的是，这并**不**意味着Snakemake使它们在作业shell命令中可用。相反，出于数据可靠性和可重现性的原因，您需要通过params指令将它们显式地传递给作业，例如：

```
envvars:
    "SOME_VARIABLE"

rule do_something:
    output:
         "test.txt"
    params:
        x=os.environ["SOME_VARIABLE"]
    shell:
        "echo {params.x} > {output}"
```

- 例子

```
import os
envvars:
    "SHELL"
shell.executable(os.environ["SHELL"]) # 使用用户默认的shell
```

3.4.15 配置工作目录

所有在Snakefile中的路径都是相对于执行snakemake命令的目录解释的。可以通过在Snakefile中指定工作目录来覆盖此行为：

```
workdir: "logs"

rule all:
    input:
        "test.out"

rule a:
    output:
        "test.out"
    shell:
        "echo test > {output}"
```

通常，最好只通过命令行设置工作目录，因为上述指令会限制 Snakemake 工作流的可移植性。

```
# --directory, -d
```

3.4.16 推荐的目录结构

```
├── .gitignore
├── README.md
├── LICENSE.md
├── workflow
│   ├── rules
│   │   ├── module1.smk
│   │   └── module2.smk
│   ├── envs
│   │   ├── tool1.yaml
│   │   └── tool2.yaml
│   ├── scripts
│   │   ├── script1.py
│   │   └── script2.R
│   ├── notebooks
│   │   ├── notebook1.py.ipynb
│   │   └── notebook2.r.ipynb
│   ├── report
│   │   ├── plot1.rst
```

```
19 |   |   └── plot2.rst
20 |   └── Snakefile
21 ├── config
22 |   ├── config.yaml
23 |   └── some-sheet.tsv
24 ├── results
25 └── resources
```

4. 流程运行结束用邮件、微信通知

- 移动到环境变量目录

```
sudo mv notifyMe /usr/local/bin/
sudo mv notifyMe_by_wechat /usr/local/bin/
```

- 通过邮件

```
# -c 命令行内容
# -e 你的邮箱
# -s 状态码： 0 成功，1 失败
# -t 标题
notifyMe -c "snakemake -s mapping-2.smk -np -c 2" -e 19jmxie@stu.edu.cn -s 0 -t "Snakemake运行"
```

```
onsuccess:
    print("Your pipeline has been completed successfully!")
    shell('notifyMe -c "snakemake -s main.smk --cores 20 -p" -e "19jmxie@stu.edu.cn" -s 0 -t "snakemake SNP"')

onerror:
    print("Your process has encountered an error!")
    shell('notifyMe -c "snakemake -s main.smk --cores 20 -p" -e "19jmxie@stu.edu.cn" -s 1 -t "snakemake SNP"')
```

- 通过`小驰Coding`微信小程序服务通知

登录，复制二维码链接到浏览器打开并用微信扫一扫

```
$ notifyMe_by_wechat login
>>>>>>>>复制二维码链接到浏览器打开并用微信扫一扫
https://chiplot.online/static/mp_QRCode_img/5c9fb2417a0c31269c1ead09fb7b8884.png
剩余时间： 01:56
```

订阅通知，每次推送都得重新订阅

推送消息

```
# -t 标题
# -s 状态码：  0 成功，1 失败
notifyMe_by_wechat notify -t snakemake -s 0
```

```
onsuccess:
    print("Your pipeline has been completed successfully!")
    shell('notifyMe_by_wechat notify -t Snakemake -s 0')

onerror:
    print("Your process has encountered an error!")
    shell('notifyMe_by_wechat notify -t snakemake -s 1')
```

5. snakemake常用命令行参数介绍

--retries

全局重试次数

```
snakemake -s my.smk -c --retries 4
```

--directory或者-d

设置工作目录

```
snakemake -s my.smk -c -d path/to/
```

--set-threads

Snakemake工作流通常定义了某些规则使用的线程数。有时，覆盖工作流定义中给定的默认值是有意义的。可以通过使用`--set-threads`参数来实现，例如：

```
# 改一个规则
snakemake -s my.smk --cores 4 --set-threads myrule=2
# 改多个规则
snakemake -s thread.smk --cores  --set-threads bwa_map=2 samtools_sort=8 -np
```

--resources, --res

定义额外的资源，类似于-cores（见上文），以约束调度。资源被定义为名称和整数值。例如，`-resources mem_mb = 1000`。规则可以通过定义资源关键字来使用资源，例如resources：mem_mb = 600。如果现在有两个规则需要600个资源'mem_mb'，则调度程序将不会并行运行它们。

- 官方文档：链接

```
snakemake -s my.smk  —resources mem_mb=1000
```

--set-resources

覆盖规则的资源使用情况。这允许对工作流资源进行微调。特别是，可以通过定义规则的特定分区或覆盖临时目录来定位特定的集群节点。因此，VALUE必须是一个正整数或字符串，RULE必须是规则的名称，RESOURCE必须是资源的名称。

```
snakemake -s my.smk —resources myrule:mem_mb=1000
```

--batch

如果你的工作流包含很多作业，Snakemake 可能需要一些时间来推断依赖关系（作业 DAG）和实际需要运行的作业。涉及到的主要瓶颈是文件系统，必须查询文件的存在和修改日期。为了解决这个问题，Snakemake 允许批量运行大型工作流。这样，较少的文件需要一次评估，因此可以更快地推断作业 DAG。通过运行

```
$ snakemake -s thread.smk -p -c 20 --batch all=1/3
```

你指示只计算规则`myrule`的三个批次中的第一个。要生成第二批，请运行

```
$ snakemake -s thread.smk -p -c 20 --batch all=2/3
```

最后，当运行下面命令时

```
$ snakemake -s thread.smk -p -c 20 --batch all=3/3
```

Snakemake会完成规则`myrule`，因为它的所有输入文件都已生成，并完成了工作流。显然，选择执行批处理的规则是一个有很多输入文件和上游作业的规则，例如工作流程中的中央聚合步骤。我们建议所有工作流开发人员告知潜在用户最适合的批处理规则。

--profile

工作流专用配置文件旨在用于定义特定工作流的默认选项，例如为工作流使用的某些自定义资源（例如`api_calls`）提供约束条件，或者在不修改工作流代码本身的情况下覆盖单个规则的线程和资源定义。相比之下，全局配置文件旨在用于定义特定环境的默认选项，例如默认的群集提交命令或要并行运行的作业数量。

例如，命令

```
$ snakemake --profile myprofile
```

在每个用户和全局配置目录中都应该有一个名为`myprofile`的文件夹（在Linux上，这将是`$HOME/.config/snakemake`和`/etc/xdg/snakemake`，您可以通过`snakemake --help`找到您的系统的答案）。或者，可以给出一个绝对或相对路径来指定配置文件夹。当未指定`--profile`参数时要使用的默认配置文件也可以通过环境变量`SNAKEMAKE_PROFILE`设置，例如通过在您的`~/.bashrc`或系统范围的shell默认值中指定`export SNAKEMAKE_PROFILE=myprofile`，这意味着可以省略`--profile`标志。为了在不指定替代配置文件的情况下取消定义此环境变量所定义的配置文件以进行单个运行，可以提供特殊值`none`，即`--profile none`。

配置文件夹应包含一个名为`config.yaml`的文件，该文件定义Snakemake命令行参数的默认值。该配置文件可用于为Snakemake命令行界面的每个选项设置默认值。为此，选项`--someoption`在配置文件中变为`someoption:`。配置文件夹还可以包含辅助文件，例如作业脚本或任何类型的包装器。有关示例，请参见https://github.com/snakemake-profiles/doc。如果选项接受多个参数，则必须在配置文件中以YAML列表的形式给出。如果选项期望结构化参数（例如`--set-threads RULE=VALUE`或`--set-resources RULE:RESOURCE=VALUE`），则可以以预期的形式以字符串形式给出。

```
set-threads: myrule=5
set-resources: myrule:mem=500MB
```

或者更好的选择是作为YAML映射，例如：

```
set-threads:
    myrule: 5
set-resources:
    myrule:
        mem: 500MB
```

通过配置文件设置资源或线程当然更适合工作流程配置文件而不是全局配置文件（因为这些设置可能是特定于工作流程的）。

在https://github.com/snakemake-profiles/doc下，您可以找到公开可用的全局配置文件（例如用于集群系统）。当工作流程的Snakefile位于`workflow/Snakefile`时，配置文件应放置在`workflow/profiles/default/config.yaml`。

--workflow-profile

路径（相对于当前目录）到工作流特定配置文件夹，用于配置Snakemake以使用特定于此工作流的参数（如资源）。如果不使用此标志，则Snakemake将默认使用“profiles / default”（按此顺序相对于当前目录和相对于Snakefile搜索）。要跳过任何工作流特定配置文件，请提供特殊值“none”。在工作流配置文件中进行的设置将覆盖在一般配置文件中进行的设置（请参见“-profile”）。配置文件夹必须包含一个名为“config.yaml”的文件。此文件可以用于以YAML格式设置命令行选项的默认值。例如，“-cluster qsub”变为“cluster：qsub”在YAML文件中。建议使用工作流配置文件设置或覆盖特定于工作流的资源，例如特定规则的线程数量或所需的内存量。请注意，在这种情况下，参数可以作为嵌套的YAML映射在配置文件中给出，例如“set-threads：myrule：4”而不是“set-threads：myrule = 4”。

--dag

为了可视化工作流程，可以使用选项`--dag`。这将创建DAG的表示形式，使用的是graphviz dot语言，需要由graphviz工具`dot`进行后处理。例如，要可视化将要执行的DAG，可以发出以下命令：

```
$ snakemake --dag | dot | display
```

要将此保存到文件中，您可以指定所需的格式：

```
$ snakemake --dag | dot -Tpdf > dag.pdf
```

为了可视化整个DAG，而不考虑最终文件的存在与否，可以使用`forceall`选项：

```
$ snakemake --forceall --dag | dot -Tpdf > dag.pdf
```

当然，通过为`dot`提供更多的命令行参数，可以修改其视觉外观。

--config, -C

在工作流配置对象中设置或覆盖值。工作流配置对象可以作为变量 config 在工作流内部访问。可以通过提供一个 JSON 文件来设置默认值（请参阅文档）。

```
snakemake --config foo=bar name=red
```

--configfile, --configfiles

指定或覆盖工作流程的配置文件（请参阅文档）。以JSON或YAML格式指定的值可在工作流程内的全局配置字典中使用。多个文件按给定顺序彼此覆盖。因此，先前配置文件中缺少的键将由后续配置文件扩展。请注意，此顺序还包括在工作流程定义中定义的配置文件（将首先出现）。

```
snakemake -s thread.smk  -np --configfile config/config_module.yaml
```

--keep-going, -k

如果一个工作失败了，继续进行独立的工作。默认是一个任务失败了，全部停止。

--rerun-triggers

可能的选择：mtime、params、input、software-env、code

定义触发作业重新运行的条件。默认情况下，使用所有触发器，这保证了结果与工作流代码和配置的一致性。如果您更喜欢传统的只考虑文件修改日期的方式，请使用“--rerun-trigger mtime”。

默认值：['mtime'，'params'，'input'，'software-env'，'code']

当你优化了流程代码，但又不会影响输出结果时可以更改这个触发条件，避免重复计算

--force, -f

强制执行已选择的目标或第一个规则，无论是否已创建输出。

默认值：False

只能强制运行那些规则里面不含未知通配符的rule

```
snakemake -s mapping-2.smk -p -f plot_quals -c 20
```

--forceall, -F

强制执行所选（或第一个）规则以及所有依赖于它的规则，而不管已经创建的输出。

默认值：False

只能强制运行那些规则里面不含未知通配符的rule，并且他前面所依赖的rule都会被运行，就是从第一个rule执行到你指定的这个rule

```
snakemake -s mapping-2.smk -np -F bcftools_call -c 20
```

不指定规则将全部重新运行

```
snakemake -s mapping-2.smk -np -F -c 20
```

--forcerun, -R

强制重新执行或创建给定的规则或文件。如果您更改了规则并希望更新工作流中所有输出，请使用此选项。

可以指定任何一个rule，会从这个rule往后面的rule运行到结束，跟`--forceall`正好相反

```
snakemake -s mapping-2.smk -np -c 20 -R samtools_index
```

--rerun-incomplete, --ri

重新运行所有输出被识别为不完整的rule。

默认值: False

--list-input-changes

列出Snakefile中定义的输入文件已更改的所有输出文件（例如，在规则定义中添加了新的输入文件或重命名了文件）。

```
snakemake --list-input-changes
snakemake -n --forcerun $(snakemake --list-input-changes)
```

--list, -l

显示给定的Snakefile中可用的规则。

```
$ snakemake -s mapping-2.smk -l
all
bcftools_call
bwa_map
plot_quals
samtools_index
samtools_sort
```

--list-target-rules, --lt

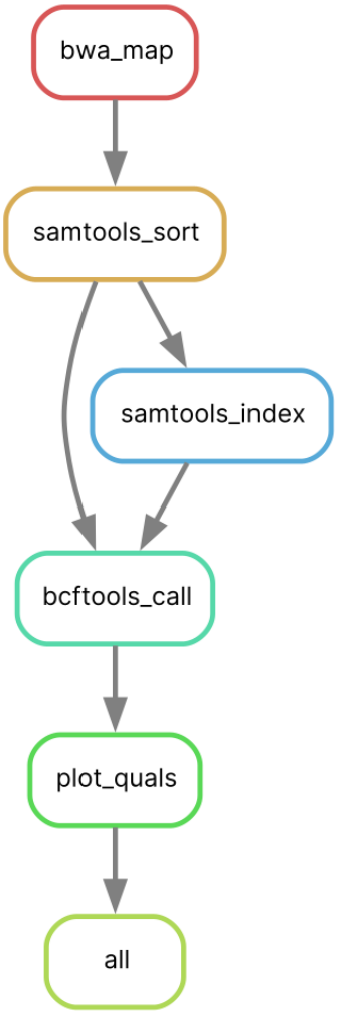
显示给定Snakefile中可用的目标规则。

```
$ snakemake -s mapping-2.smk --lt samtools_index
all
bcftools_call
plot_quals
```

--rulegraph

只显示单样本的简要流程图

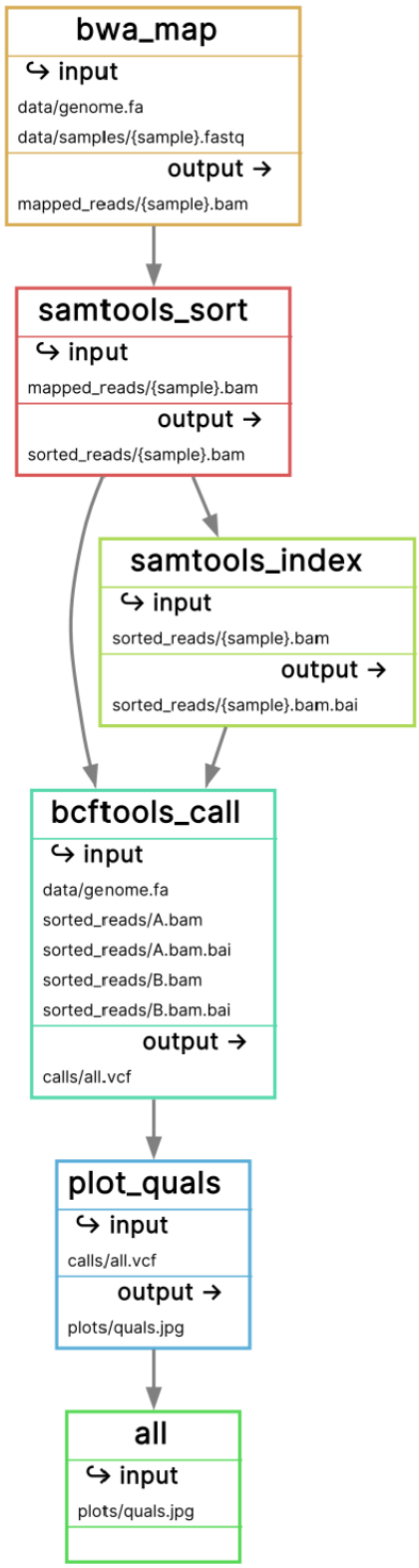
```
snakemake -s mapping-2.smk --rulegraph | dot -Tsvg > rulegraph.svg
```



--filegraph

显示单样本的详细流程图，包括输入输出文件

```
snakemake -s mapping-2.smk --filegraph | dot -Tsvg > filegraph.svg
```



--summary, -S

打印工作流程创建的所有文件的摘要。它具有以下列：文件名、修改时间、规则版本、输入文件、shell命令、状态、计划。因此，规则版本包含文件创建时的版本（请参见规则的版本关键字），状态表示文件是否丢失、其输入文件是否更新或者规则的版本或实现自文件创建以来是否发生变化。输入文件和shell命令列是不言自明的。最后一列表示下一个工作流程执行期间文件是否将被更新或创建。

```
$ snakemake -s mapping-2.smk -S
Building DAG of jobs...
output_file date    rule    version log-file(s) status  plan
plots/quals.jpg Tue Aug 29 09:42:24 2023    plot_quals  -       ok  no update
calls/all.vcf   Tue Aug 29 09:42:23 2023    bcftools_call   -       ok  no update
sorted_reads/A.bam  Tue Aug 29 09:41:35 2023    samtools_sort   -       ok  no
update
mapped_reads/A.bam  Tue Aug 29 00:20:28 2023    bwa_map -       ok  no update
sorted_reads/B.bam  Tue Aug 29 09:37:23 2023    samtools_sort   -       ok  no
update
mapped_reads/B.bam  Tue Aug 29 00:20:28 2023    bwa_map -       ok  no update
sorted_reads/A.bam.bai  Tue Aug 29 09:42:33 2023    samtools_index  -       ok  no
update
sorted_reads/B.bam.bai  Tue Aug 29 09:37:23 2023    samtools_index  -       ok  no
update
```

--delete-all-output

删除工作流生成的所有文件。与 -dry-run 一起使用可以列出文件而不实际删除任何文件。请注意，这不会递归到子工作流中。受保护的文件不会被删除。尽管如此，请小心使用！

```
$ snakemake -s mapping-2.smk -np -c 20 --delete-all-output
Building DAG of jobs...
Would delete plots/quals.jpg
Would delete calls/all.vcf
Would delete sorted_reads/A.bam
Would delete mapped_reads/A.bam
Would delete sorted_reads/B.bam
Would delete mapped_reads/B.bam
Would delete sorted_reads/A.bam.bai
Would delete sorted_reads/B.bam.bai
```

--gui

将基于HTML的用户界面提供给给定的网络和端口，例如168.129.10.15:8000。默认情况下，Snakemake仅在本地网络（默认端口：8000）中可用。要使Snakemake侦听所有IP地址，请将特殊主机地址0.0.0.0添加到URL中（0.0.0.0:8000）。如果Snakemake在像Docker这样的虚拟化环境中使用，则这一点非常重要。如果可能，将打开浏览器窗口。

要先安装Flask

```
mamba install Flask
```

```
snakemake -s mapping-2.smk -np -c 20 --gui
```

指定ip和端口

```
snakemake -s mapping-2.smk -np -c 20 --gui 0.0.0.0:8000

# 查看监听状态
ss -tnlp
# 查看本机ip地址
ifconfig
```

在浏览器中打开：http://172.16.163.78:8000/

--quiet, -q

可能的选择：progress, rules, all

不要输出某些信息。如果没有参数使用，则不输出任何进度或规则信息。定义`all`会导致完全不打印任何信息。

--verbose

打印调试输出。

--show-failed-logs

自动显示失败作业的日志。

--list-conda-envs

列出所有的conda环境及其在磁盘上的位置。

```
$ snakemake -s mapping-2.smk -np -c 20  --list-conda-envs
Building DAG of jobs...
environment container   location
envs/vcfR.yaml      .snakemake/conda/8fb308d5cd70a49d4933170c1fe47b43_
```

--conda-prefix

指定独立conda环境的安装目录，在其中创建“conda”和“conda-archive”目录。它们分别用于存储conda环境及其归档。如果未提供，则该值将设置为相对于调用目录的“.snakemake”目录。如果提供，必须同时设置“-use-conda”标志。该值可以作为相对路径给出，将被推断到调用目录，也可以作为绝对路径给出。该值也可以通过环境变量$SNAKEMAKE_CONDA_PREFIX提供。

--conda-create-envs-only

运行`snakemake --use-conda --conda-create-envs-only`将只安装所需的conda环境，而不运行完整的工作流程。

--conda-cleanup-envs

清理未使用的conda环境。

> ## --conda-cleanup-pkgs

可能的选择：tarballs，cache

创建环境后清理conda包。在“tarballs”模式下，将清理所有已下载的包tarballs。在“cache”模式下，还将清理未使用的包缓存。如果省略了模式，将默认仅清理tarballs。

6. 二代基因组组装实战

6.1 安装相关软件及命令参数介绍

6.1.1 安装相关软件

> ## 创建一个新环境

```
# 新建一个名字为SG_assembly的环境
mamba create -n SG_assembly
# 激活该环境
mamba activate SG_assembly
```

> ## fastp

- github地址：fastp
- 通过mamba安装：

```
mamba install -c conda-forge -c bioconda fastp
```

> ## SPAdes

```
# 创建文件夹
mkdir SG_assembly
cd SG_assembly

# 下载
wget http://cab.spbu.ru/files/release3.15.4/SPAdes-3.15.4-Linux.tar.gz
tar -xzf SPAdes-3.15.4-Linux.tar.gz
cd SPAdes-3.15.4-Linux/bin/

# 添加到环境变量，路径取决你放在哪
vim ~/.zshrc
export PATH=$PATH:path/to/SPAdes-3.15.4-Linux/bin/
```

> ## BUSCO

- 官网：busco
- 官方文档：链接

```
# 通过mamba安装
mamba install -c conda-forge -c bioconda busco=5.5.0
```

6.1.2 相关软件命令参数介绍

fastp

```
# -i 前向序列的路径
# -I 后向序列的路径
# -o 质控后的前向序列
# -O 质控后的后向序列
# -j 输出质控报告json
# -h 输出质控报告html
# -w 指定多少线程
# -q 指定质量分的阈值
# -u 低质量碱基比例
# -n 最多可容忍多少个N碱基，超过这个数量的将被舍弃
# -l 最短read长度，低于这个长度的将被舍弃
# -5 对5'端序列修剪掉低质量序列
# -3 对3'端序列修剪掉低质量序列
mkdir fastp_output
fastp -i SG_reads/PT24_64_1.fq.gz -I SG_reads/PT24_64_2.fq.gz -o fastp_output/PT24_64_paired_1.fq.gz -O fastp_output/PT24_64_paired_2.fq.gz -j fastp_output/PT24_64_fastp.json -h fastp_output/PT24_64_fastp.html -w 16 -q 20 -u 10 -n 3 -l 40 -5 -3
# 因为是校正 所以条件苛刻一点，-u 10 表示一条read里面如果超过10%的碱基质量低于20分就会被丢弃
```

SPAdes

```
# --isolate 建议用于高覆盖度的分离物和多细胞数据。
# -1 质控后的前向序列
# -2 质控后的后向序列
# -t 线程
# -m 内存
# -o 结果输出
spades.py --isolate -1 fastp_output/PT24_64_paired_1.fq.gz -2 fastp_output/PT24_64_paired_2.fq.gz -t 48 -m 48 -o spades_output
```

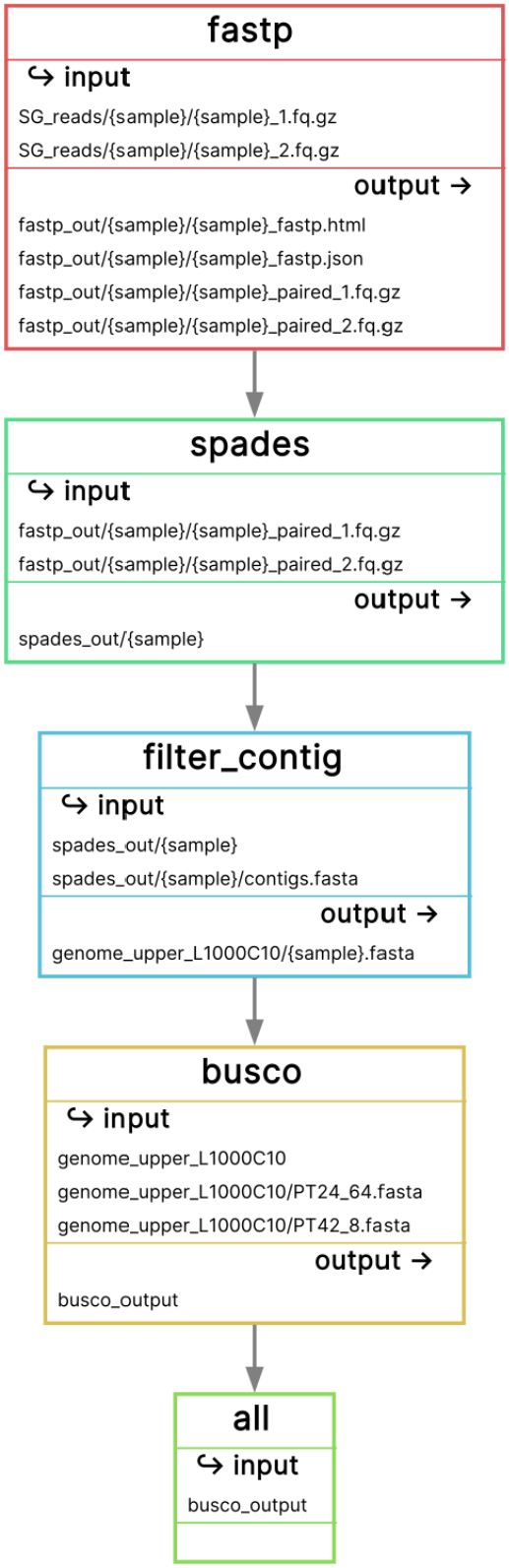
- 如果官网提供的spades运行报错可以换成我自己编译的`3.15.5`版本，这是操作系统不兼容原因

```
== Error ==  system call for: "['/home/kkk/software/SPAdes-3.15.4-Linux/bin/spades-core', '/home/kkk/SG_assembly/spades_out/PT24_64/K21/configs/config.info', '/home/kkk/SG_assembly/spades_out/PT24_64/K21/configs/isolate_mode.info']" finished abnormally, OS return value: -11

```

BUSCO

```
# -i 指定输入的基因组文件或者存放基因组的目录
# -l 数据库名称
# -m 输入的文件类型：genome, proteins, transcriptome
# -o 输出目录
busco -i genome_output  -l bacteria_odb10  -m genome -o busco_output
```

6.2 snakemake流程文件编写



6.2.1 fastp规则

- `workflow/config/config.yaml`

```yaml
samples_path: SG_reads
unqualified: 10
read_length: 40
```

- `workflow/SG_assembly.smk`

```
configfile: "workflow/config/config.yaml"
import os
samples_path = config["samples_path"]
SAMPLES = os.listdir(samples_path)
FASTP_OUT_PATH = "fastp_out"

rule all:
    input:
        expand(FASTP_OUT_PATH + "/{sample}/{sample}_paired_{num}.fq.gz", sample=SAMPLES, num=[1,2]),
        expand(FASTP_OUT_PATH + "/{sample}/{sample}_fastp.{ext}", sample=SAMPLES, ext=['html', 'json']),

rule fastp:
    input:
        r1 = samples_path + "/{sample}/{sample}_1.fq.gz",
        r2 = samples_path + "/{sample}/{sample}_2.fq.gz",
    output:
        r1 = FASTP_OUT_PATH + "/{sample}/{sample}_paired_1.fq.gz",
        r2 = FASTP_OUT_PATH + "/{sample}/{sample}_paired_2.fq.gz",
        json= ensure(temp(FASTP_OUT_PATH + "/{sample}/{sample}_fastp.json"),non_empty=True),
        html = ensure(FASTP_OUT_PATH + "/{sample}/{sample}_fastp.html", non_empty=True), # Often, it is a good idea to combine ensure annotations with retry definitions, e.g. for retrying upon invalid checksums or empty files.
    log:
        "logs/fastp/{sample}.log"
    threads: 8
    params:
        unqualified=config['unqualified'],
        read_length=config['read_length']
    conda:
        "SG_assembly"
    shell:
        "fastp -i {input.r1} -I {input.r2} -o {output.r1} -O {output.r2} "
        "-j {output.json} -h {output.html} -w {threads} "
        "-q 20 " # the quality value that a base is qualified
        "-u {params.unqualified} " # how many percents of bases are allowed to be unqualified (0~100).
        "-n 3 " # number of N allowed
        "-l {params.read_length} " # reads shorter than length_required will be discarded, particularly those with adapter.
        "-5 " # enable trimming in 5' ends.
        "-3 " # enable trimming in 3' ends.
        "&> {log}"
```

- 试运行

```
snakemake -s workflow/SG_assembly.smk -np -c --use-conda
```

6.2.2 spades规则

- `workflow/SG_assembly.smk`

```
configfile: "workflow/config/config.yaml"
import os
samples_path = config["samples_path"]
SAMPLES = os.listdir(samples_path)
FASTP_OUT_PATH = "fastp_out"
SPADES_OUT_PATH = "spades_out"



rule all:
    input:
        # expand(FASTP_OUT_PATH + "/{sample}/{sample}_paired_{num}.fq.gz", sample=SAMPLES, num=[1,2]),
        # expand(FASTP_OUT_PATH + "/{sample}/{sample}_fastp.{ext}", sample=SAMPLES, ext=['html', 'json']),
        expand(SPADES_OUT_PATH + "/{sample}", sample=SAMPLES),

rule fastp:
    input:
        r1 = samples_path + "/{sample}/{sample}_1.fq.gz",
        r2 = samples_path + "/{sample}/{sample}_2.fq.gz",
    output:
        r1 = FASTP_OUT_PATH + "/{sample}/{sample}_paired_1.fq.gz",
        r2 = FASTP_OUT_PATH + "/{sample}/{sample}_paired_2.fq.gz",
        json= ensure(temp(FASTP_OUT_PATH + "/{sample}/{sample}_fastp.json"),non_empty=True),
        html = ensure(FASTP_OUT_PATH + "/{sample}/{sample}_fastp.html", non_empty=True), # Often, it is a good idea to combine ensure annotations with retry definitions, e.g. for retrying upon invalid checksums or empty files.
    log:
        "logs/fastp/{sample}.log"
    threads: 8
    params:
        unqualified=config['unqualified'],
        read_length=config['read_length']
    conda:
        "SG_assembly"
    shell:
        "fastp -i {input.r1} -I {input.r2} -o {output.r1} -O {output.r2} "
        "-j {output.json} -h {output.html} -w {threads} "
        "-q 20 " # the quality value that a base is qualified
```

```
        "-u {params.unqualified} " # how many percents of bases are allowed to be unqualified (0~100).
        "-n 3 " # number of N allowed
        "-l {params.read_length} " # reads shorter than length_required will be discarded, particularly those with adapter.
        "-5 " # enable trimming in 5' ends.
        "-3 " # enable trimming in 3' ends.
        "&> {log}"

rule spades:
    input:
        r1 = rules.fastp.output.r1,
        r2 = rules.fastp.output.r2,
    output:
        directory(SPADES_OUT_PATH + "/{sample}")
    log:
        "logs/spades/{sample}.log"
    benchmark:
        "benchmarks/{sample}.spades.benchmark.txt"
    threads: 48
    resources:
        mem_gb = 48
    shell:
        "spades.py --isolate -1 {input.r1} -2 {input.r2} -t {threads} -m {resources.mem_gb} -o {output} &> {log}"

```

6.2.3 过滤contig规则

- `scripts/filterLengthAndCoverage.py`

```
from os import path
import os
from Bio import SeqIO
import datetime

def filterLength(fastaPath, outputPath, logPath, thLength=1000, thCoverage=10):
    """
    滤掉长度低于某个阈值的contigs
    1000是细菌单个基因的平均长度
    覆盖度低于10，说明这条contig很有可能是污染序列，或者这个地方确实是测序深度不够
    上传基因组到NCBI会帮你检测污染序列，PT34_32被鉴定到2条contigs是污染序列，其覆盖度低于10，其他正常contigs都高于10
    """
    log_f = open(logPath,"w")
    genomeDir = path.dirname(outputPath)
    if not path.exists(genomeDir):
        os.mkdir(genomeDir)

```

```
18
19      if not path.exists(fastaPath):
20          print("No such fasta file")
21          print(fastaPath)
22          os._exit(1)
23      sampleName = ".".join(path.basename(outputPath).split(".")[:-1])
24      records = SeqIO.parse(fastaPath, "fasta")
25      good_records = []
26
27      total_count = 0
28      total_length = 0
29
30      success_count = 0
31      success_length = 0
32      for rec in records:
33          seqLength = len(rec.seq)
34          seqIDSplit = rec.id.split('_')
35          seqCoverage = float(seqIDSplit[-1])
36          total_count += 1
37          total_length += seqLength
38
39          if seqLength >= thLength and seqCoverage >= thCoverage:
40              rec.description = rec.id
41              rec.id = f"{sampleName}_contig_{seqIDSplit[1]}"
42              good_records.append(rec)
43              success_length += seqLength
44              success_count += 1
45          else:
46              current_datetime = datetime.datetime.now()
47              formatted_datetime = current_datetime.strftime("%m/%d/%Y %H:%M:%S")
48              print(f"[{formatted_datetime}] WARNING: {rec.id} fail to pass.",
file=log_f)
49
50      with open(outputPath, "w") as f:
51          SeqIO.write(good_records, f, "fasta")
52
53      out_res = f"""
54      ## Before filter:
55      total number of contigs: {total_count}
56      total length of contigs: {total_length} bp
57
58      ## After filter:
59      total number of contigs: {success_count}
60      total length of contigs: {success_length} bp
61      """
62      print(out_res, file=log_f)
63      log_f.close()
64
65  if __name__ == '__main__':
```

```
    filterLength(snakemake.input.genome, snakemake.output[0], snakemake.log[0], snakemake.params.thLength, snakemake.params.thCoverage)
```

- `workflow/SG_assembly.smk`

```
configfile: "workflow/config/config.yaml"
import os
samples_path = config["samples_path"]
SAMPLES = os.listdir(samples_path)
FASTP_OUT_PATH = "fastp_out"
SPADES_OUT_PATH = "spades_out"
FILTERED_GENOME_OUT_PATH = "genome_upper_L1000C10"


rule all:
    input:
        # expand(FASTP_OUT_PATH + "/{sample}/{sample}_paired_{num}.fq.gz", sample=SAMPLES, num=[1,2]),
        # expand(FASTP_OUT_PATH + "/{sample}/{sample}_fastp.{ext}", sample=SAMPLES, ext=['html', 'json']),
        # expand(SPADES_OUT_PATH + "/{sample}", sample=SAMPLES),
        expand(FILTERED_GENOME_OUT_PATH + "/{sample}.fasta", sample=SAMPLES),

rule fastp:
    input:
        r1 = samples_path + "/{sample}/{sample}_1.fq.gz",
        r2 = samples_path + "/{sample}/{sample}_2.fq.gz",
    output:
        r1 = FASTP_OUT_PATH + "/{sample}/{sample}_paired_1.fq.gz",
        r2 = FASTP_OUT_PATH + "/{sample}/{sample}_paired_2.fq.gz",
        json= ensure(temp(FASTP_OUT_PATH + "/{sample}/{sample}_fastp.json"),non_empty=True),
        html = ensure(FASTP_OUT_PATH + "/{sample}/{sample}_fastp.html", non_empty=True), # Often, it is a good idea to combine ensure annotations with retry definitions, e.g. for retrying upon invalid checksums or empty files.
    log:
        "logs/fastp/{sample}.log"
    threads: 8
    params:
        unqualified=config['unqualified'],
        read_length=config['read_length']
    conda:
        "SG_assembly"
    shell:
        "fastp -i {input.r1} -I {input.r2} -o {output.r1} -O {output.r2} "
        "-j {output.json} -h {output.html} -w {threads} "
        "-q 20 " # the quality value that a base is qualified
        "-u {params.unqualified} " # how many percents of bases are allowed to be unqualified (0~100).
```

```
        "-n 3 " # number of N allowed
        "-l {params.read_length} " # reads shorter than length_required will be discarded, particularly those with adapter.
        "-5 " # enable trimming in 5' ends.
        "-3 " # enable trimming in 3' ends.
        "&> {log}"

rule spades:
    input:
        r1 = rules.fastp.output.r1,
        r2 = rules.fastp.output.r2,
    output:
        directory(SPADES_OUT_PATH + "/{sample}")
    log:
        "logs/spades/{sample}.log"
    benchmark:
        "benchmarks/{sample}.spades.benchmark.txt"
    threads: 48
    resources:
        mem_gb = 48
    shell:
        "spades.py --isolate -1 {input.r1} -2 {input.r2} -t {threads} -m {resources.mem_gb} -o {output} &> {log}"

rule filter_contig:
    input:
        rules.spades.output
    output:
        FILTERED_GENOME_OUT_PATH + "/{sample}.fasta"
    params:
        thLength = 1000,  # 长度阈值
        thCoverage = 10,  # 覆盖度阈值
        genome = SPADES_OUT_PATH + "/{sample}/contigs.fasta"
    log:
        "logs/filter/{sample}.log"
    conda:
        "SG_assembly"
    script:
        "scripts/filterLengthAndCoverage.py"
```

6.2.4 busco规则

- `workflow/SG_assembly.smk`

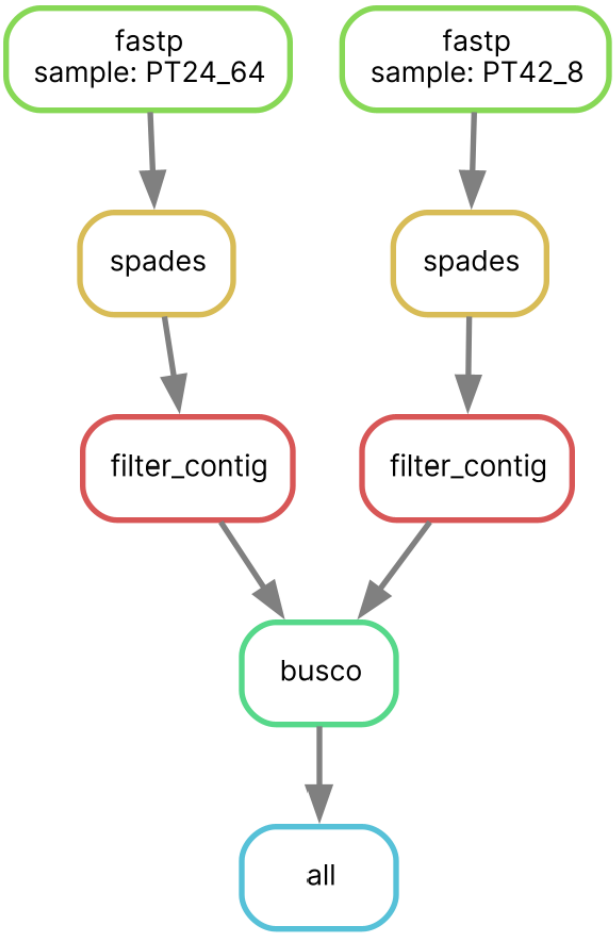
```
import os
configfile: "workflow/config/config.yaml"

sample_path = config["samples_path"]
FASTP_OUT_PATH = "fastp_out"
```

```
SPADES_OUT_PATH = "spades_out"
FILTERED_GENOME_OUT_PATH = "genome_upper_L1000C10"
BUSCO_OUT_PATH = "busco_output"

SAMPLES = os.listdir(sample_path)

onerror:
    print("error!")

onsuccess:
    print("success!")

rule all:
    input:
        # expand(FASTP_OUT_PATH + "/{sample}/{sample}_paired_{num}.fq.gz", sample=SAMPLES, num=[1,2]),
        # expand(FASTP_OUT_PATH + "/{sample}/{sample}_fastp.{ext}", sample=SAMPLES, ext=["json", "html"])
        # expand(SPADES_OUT_PATH + "/{sample}", sample=SAMPLES),
        # expand(FILTERED_GENOME_OUT_PATH + "/{sample}.fasta", sample=SAMPLES),
        BUSCO_OUT_PATH + "/batch_summary.txt"

rule fastp:
    input:
        r1 = sample_path + "/{sample}/{sample}_1.fq.gz",
        r2 = sample_path + "/{sample}/{sample}_2.fq.gz",
    output:
        r1 = FASTP_OUT_PATH + "/{sample}/{sample}_paired_1.fq.gz",
        r2 = FASTP_OUT_PATH + "/{sample}/{sample}_paired_2.fq.gz",
        json = temp(FASTP_OUT_PATH + "/{sample}/{sample}_fastp.json"),
        html = FASTP_OUT_PATH + "/{sample}/{sample}_fastp.html",
    threads: 10
    params:
        unqualified=config['unqualified'],
        read_length=config['read_length']
    log:
        "logs/fastp/{sample}.log"
    conda:
        "SG_assembly"
    shell:
        "fastp -i {input.r1} -I {input.r2} -o {output.r1} -O {output.r2} "
        "-j {output.json} -h {output.html} "
        "-w {threads} -q 20 -u {params.unqualified} -n 3 -l {params.read_length} -5 -3 &> {log}"


rule spades:
    input:
        r1 = rules.fastp.output.r1,
```

```
        r2 = rules.fastp.output.r2,
    threads: 64
    resources:
        mem_gb = 48
    output:
        directory(SPADES_OUT_PATH + "/{sample}")
    benchmark:
        "benchmarks/{sample}.spades.benchmark.txt"
    log:
        "logs/spades/{sample}.log"
    shell:
        "spades.py --isolate -1 {input.r1} -2 {input.r2} -t {threads} -m
{resources.mem_gb} -o {output} &> {log}"


rule filter_contig:
    input:
        rules.spades.output
    output:
        FILTERED_GENOME_OUT_PATH + "/{sample}.fasta"
    params:
        thLength = 1000,  # 长度阈值
        thCoverage = 10,  # 覆盖度阈值
        genome = SPADES_OUT_PATH + "/{sample}/contigs.fasta"
    log:
        "logs/filter/{sample}.log"
    conda:
        "SG_assembly"
    script:
        "scripts/filterLengthAndCoverage.py"


rule busco:
    input:
        expand(FILTERED_GENOME_OUT_PATH + "/{sample}.fasta", sample=SAMPLES)
    output:
        ensure(BUSCO_OUT_PATH + "/batch_summary.txt", non_empty=True)
    params:
        dir = FILTERED_GENOME_OUT_PATH,
        out = BUSCO_OUT_PATH
    conda:
        "SG_assembly"
    log:
        "logs/busco/busco.log"
    shell:
        "busco -i {params.dir}  -l bacteria_odb10  -m genome -o {params.out} -f &>
{log}"
```

- 可视化流程图

```
snakemake -s workflow/SG_assembly.smk --dag | dot -Tsvg > workflow.svg

snakemake -s workflow/SG_assembly.smk --filegraph | dot -Tsvg > filegraph.svg
```



6.2.5 fastp模块化

- `workflow/modules/fastp.smk`

```
FASTP_OUT_PATH = "fastp_out"
samples_path = config["samples_path"]
rule fastp:
    input:
        r1 = samples_path + "/{sample}/{sample}_1.fq.gz",
        r2 = samples_path + "/{sample}/{sample}_2.fq.gz",
    output:
        r1 = FASTP_OUT_PATH + "/{sample}/{sample}_paired_1.fq.gz",
        r2 = FASTP_OUT_PATH + "/{sample}/{sample}_paired_2.fq.gz",
        json= ensure(temp(FASTP_OUT_PATH + "/{sample}/{sample}_fastp.json"),non_empty=True),
        html = ensure(FASTP_OUT_PATH + "/{sample}/{sample}_fastp.html", non_empty=True), # Often, it is a good idea to combine ensure annotations with retry definitions, e.g. for retrying upon invalid checksums or empty files.
```

```
    log:
        "logs/fastp/{sample}.log"
    threads: 8
    params:
        unqualified=config['unqualified'],
        read_length=config['read_length']
    conda:
        "SG_assembly"
    shell:
        "fastp -i {input.r1} -I {input.r2} -o {output.r1} -O {output.r2} "
        "-j {output.json} -h {output.html} -w {threads} "
        "-q 20 " # the quality value that a base is qualified
        "-u {params.unqualified} " # how many percents of bases are allowed to be unqualified (0~100).
        "-n 3 " # number of N allowed
        "-l {params.read_length} " # reads shorter than length_required will be discarded, particularly those with adapter.
        "-5 " # enable trimming in 5' ends.
        "-3 " # enable trimming in 3' ends.
        "&> {log}"
```

- `workflow/SG_assembly_m.smk`

```
import os
configfile: "workflow/config/config.yaml"

sample_path = config["samples_path"]
FASTP_OUT_PATH = "fastp_out"
SPADES_OUT_PATH = "spades_out"
FILTERED_GENOME_OUT_PATH = "genome_upper_L1000C10"
BUSCO_OUT_PATH = "busco_output"

SAMPLES = os.listdir(sample_path)

onerror:
    print("error!")

onsuccess:
    print("success!")


module fastp_m:
    snakefile: "modules/fastp.smk"
    config: config

use rule fastp from fastp_m as my_fastp

```

```
rule all:
    input:
        BUSCO_OUT_PATH + "/batch_summary.txt"
    default_target: True




rule spades:
    input:
        r1 = rules.my_fastp.output.r1,
        r2 = rules.my_fastp.output.r2,
    threads: 64
    resources:
        mem_gb = 48
    output:
        directory(SPADES_OUT_PATH + "/{sample}")
    benchmark:
        "benchmarks/{sample}.spades.benchmark.txt"
    log:
        "logs/spades/{sample}.log"
    shell:
        "spades.py --isolate -1 {input.r1} -2 {input.r2} -t {threads} -m
{resources.mem_gb} -o {output} &> {log}"



rule filter_contig:
    input:
        rules.spades.output
    output:
        FILTERED_GENOME_OUT_PATH + "/{sample}.fasta"
    params:
        thLength = 1000,  # 长度阈值
        thCoverage = 10,  # 覆盖度阈值
        genome = SPADES_OUT_PATH + "/{sample}/contigs.fasta"
    log:
        "logs/filter/{sample}.log"
    conda:
        "SG_assembly"
    script:
        "scripts/filterLengthAndCoverage.py"


rule busco:
    input:
        expand(FILTERED_GENOME_OUT_PATH + "/{sample}.fasta", sample=SAMPLES)
    output:
```

```
        ensure(BUSCO_OUT_PATH + "/batch_summary.txt", non_empty=True)
    params:
        dir = FILTERED_GENOME_OUT_PATH,
        out = BUSCO_OUT_PATH
    conda:
        "SG_assembly"
    log:
        "logs/busco/busco.log"
    shell:
        "busco -i {params.dir}  -l bacteria_odb10  -m genome -o {params.out} -f &> {log}"
```

- 可视化流程

```
snakemake -s workflow/SG_assembly_m.smk  --dag| dot -Tsvg > workflow_m.svg
```