(Short?) introduction to Snakemake

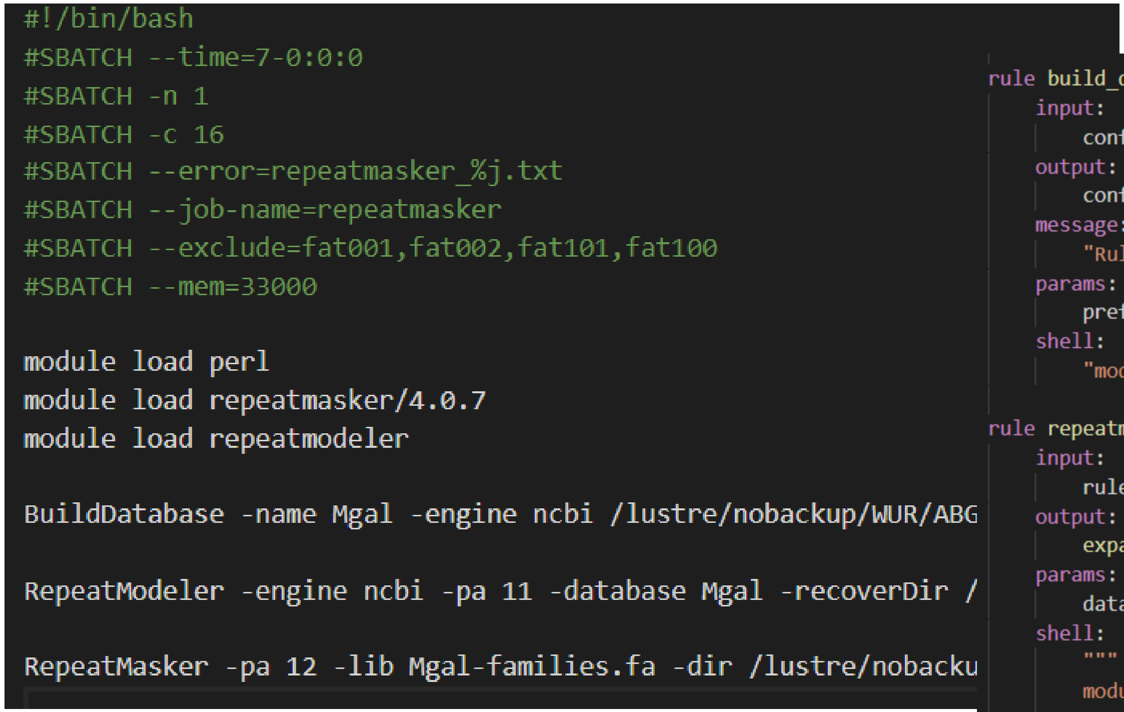
Carolina Barros

08/03/2021

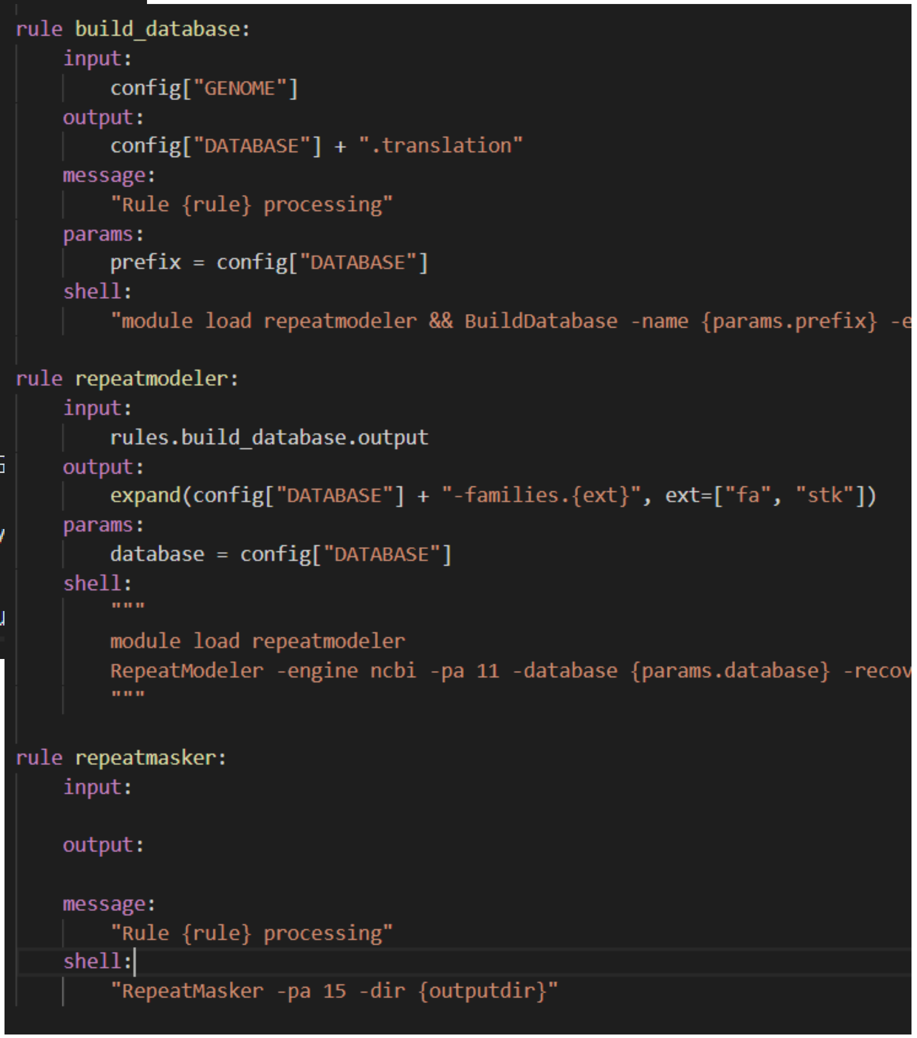
What is Snakemake?

- Workflow management system
- Tool to create reproducible and scalable data analysis
- Based on python
- Can work together with the HPC
- All analysis steps in one file (Snakefile)
  - Each step is one rule
    - Each rule defines how to get output files from input files
      - Each rule has an input, output and command to be run

Snakemake vs bash script



```bash
#!/bin/bash
#SBATCH --time=7-0:0:0
#SBATCH -n 1
#SBATCH -c 16
#SBATCH --error=repeatmasker_%j.txt
#SBATCH --job-name=repeatmasker
#SBATCH --exclude=fat001,fat002,fat101,fat100
#SBATCH --mem=33000

module load perl
module load repeatmasker/4.0.7
module load repeatmodeler

BuildDatabase -name Mgal -engine ncbi /lustre/nobackup/WUR/ABG

RepeatModeler -engine ncbi -pa 11 -database Mgal -recoverDir /

RepeatMasker -pa 12 -lib Mgal-families.fa -dir /lustre/nobacku
```

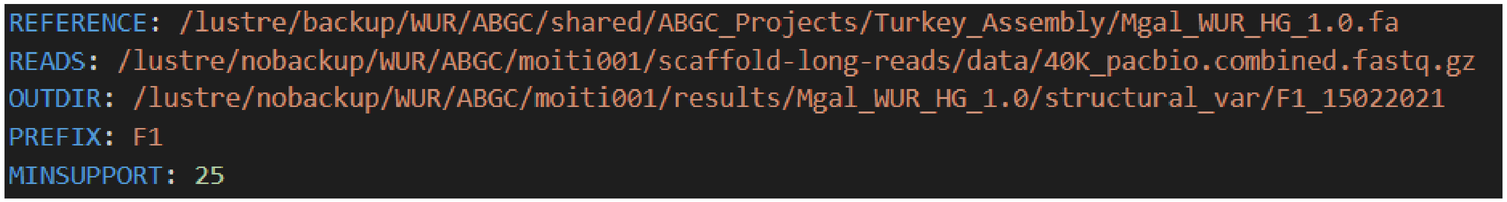


```python
rule build_database:
    input:
        config["GENOME"]
    output:
        config["DATABASE"] + ".translation"
    message:
        "Rule {rule} processing"
    params:
        prefix = config["DATABASE"]
    shell:
        "module load repeatmodeler && BuildDatabase -name {params.prefix} -e

rule repeatmodeler:
    input:
        rules.build_database.output
    output:
        expand(config["DATABASE"] + "-families.{ext}", ext=["fa", "stk"])
    params:
        database = config["DATABASE"]
    shell:
        """
        module load repeatmodeler
        RepeatModeler -engine ncbi -pa 11 -database {params.database} -recov
        """

rule repeatmasker:
    input:

    output:

    message:
        "Rule {rule} processing"
    shell:
        "RepeatMasker -pa 15 -dir {outputdir}"
```

Pipeline files

- Two main files:
  - Config file – contains the files and other parameters to be used
  - Snakefile – contains the rules (programs) it will run

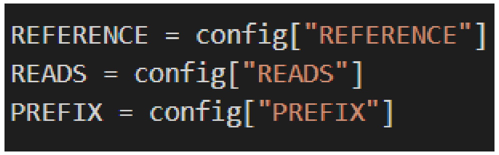
Config:



```
REFERENCE: /lustre/backup/WUR/ABGC/shared/ABGC_Projects/Turkey_Assembly/Mgal_WUR_HG_1.0.fa
READS: /lustre/nobackup/WUR/ABGC/moiti001/scaffold-long-reads/data/40K_pacbio.combined.fastq.gz
OUTDIR: /lustre/nobackup/WUR/ABGC/moiti001/results/Mgal_WUR_HG_1.0/structural_var/F1_15022021
PREFIX: F1
MINSUPPORT: 25
```

You can have as many config files as you want
Allows you to run several instances of the same pipeline at the same time (need to specify different work directories)

In the snakefile:

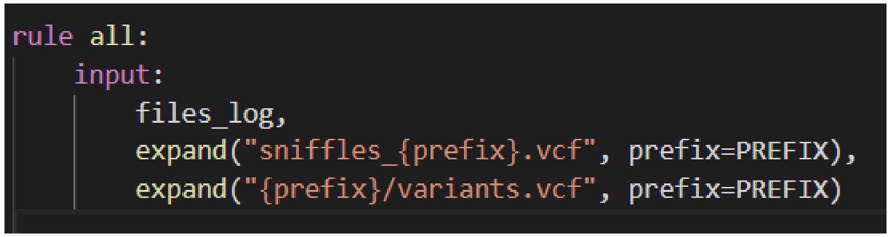


```
REFERENCE = config["REFERENCE"]
READS = config["READS"]
PREFIX = config["PREFIX"]
```

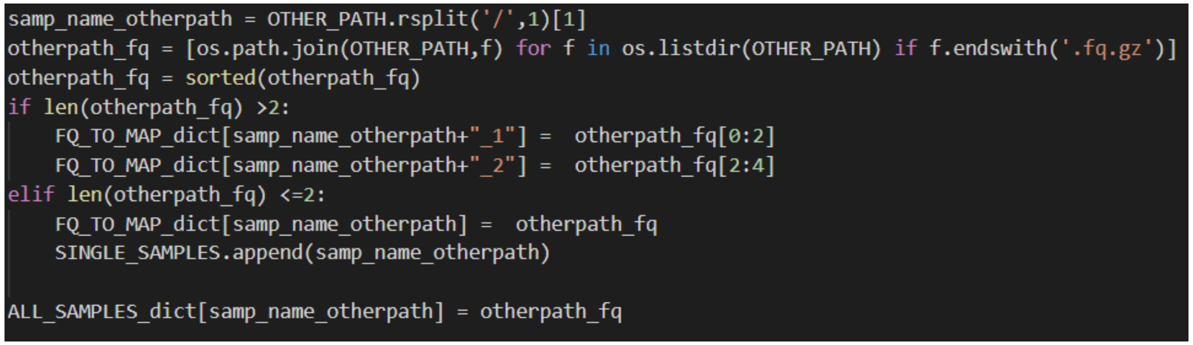
Pipeline files

Snakefile:

- Rule all – first rule that specifies the pipeline's output files
- Other rules
- Python code

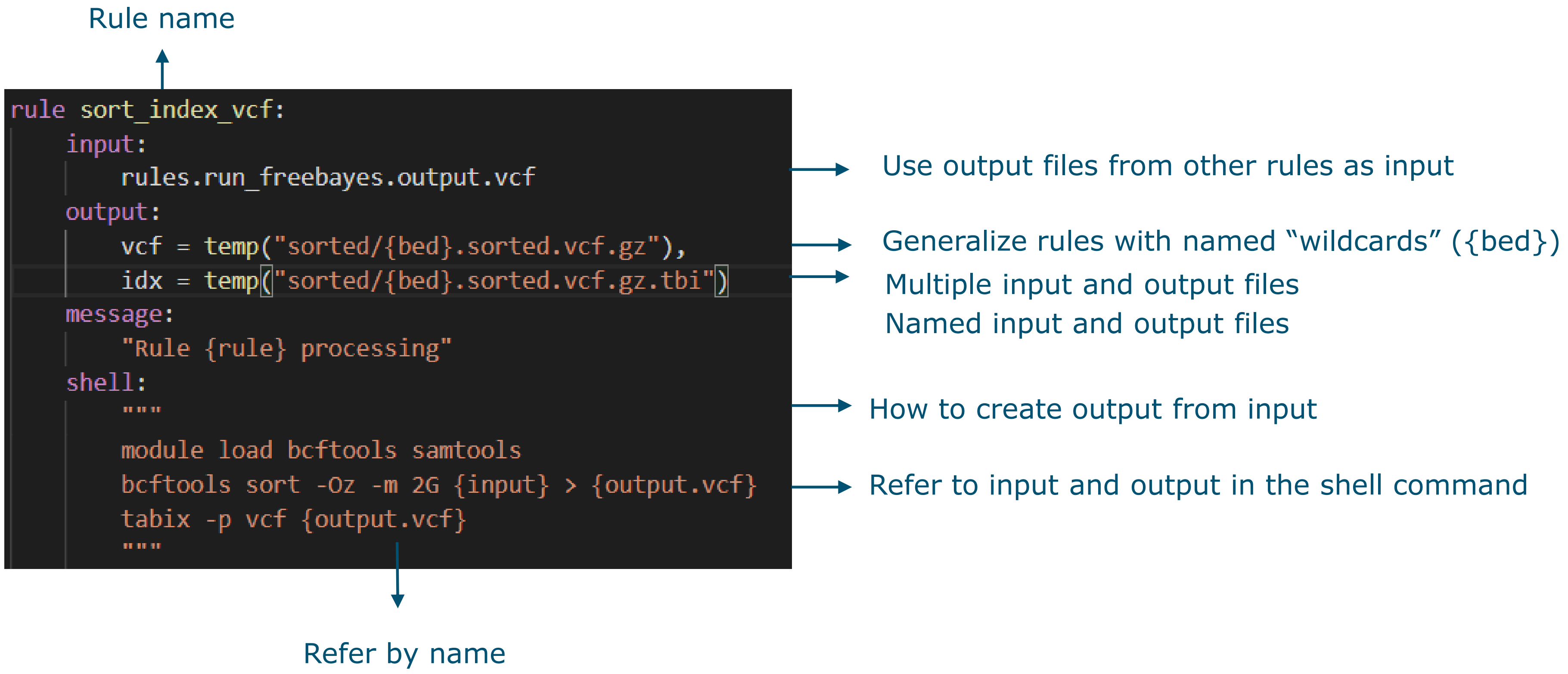


```
rule all:
    input:
        files_log,
        expand("sniffles_{prefix}.vcf", prefix=PREFIX),
        expand("{prefix}/variants.vcf", prefix=PREFIX)
```



```
samp_name_otherpath = OTHER_PATH.rsplit('/',1)[1]
otherpath_fq = [os.path.join(OTHER_PATH,f) for f in os.listdir(OTHER_PATH) if f.endswith('.fq.gz')]
otherpath_fq = sorted(otherpath_fq)
if len(otherpath_fq) >2:
    FQ_TO_MAP_dict[samp_name_otherpath+"_1"] =  otherpath_fq[0:2]
    FQ_TO_MAP_dict[samp_name_otherpath+"_2"] =  otherpath_fq[2:4]
elif len(otherpath_fq) <=2:
    FQ_TO_MAP_dict[samp_name_otherpath] =  otherpath_fq
    SINGLE_SAMPLES.append(samp_name_otherpath)

ALL_SAMPLES_dict[samp_name_otherpath] = otherpath_fq
```

Anatomy of a Snakemake rule



```
rule sort_index_vcf:
    input:
        rules.run_freebayes.output.vcf
    output:
        vcf = temp("sorted/{bed}.sorted.vcf.gz"),
        idx = temp("sorted/{bed}.sorted.vcf.gz.tbi")
    message:
        "Rule {rule} processing"
    shell:
        """
        module load bcftools samtools
        bcftools sort -Oz -m 2G {input} > {output.vcf}
        tabix -p vcf {output.vcf}
        """
```

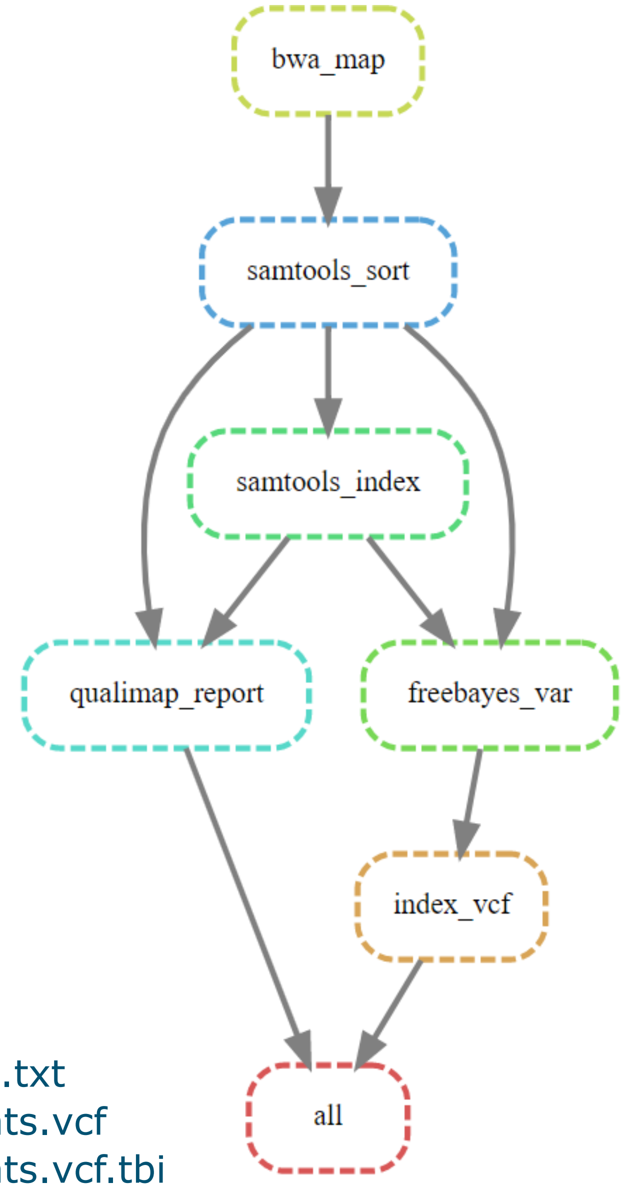
Rule name
Use output files from other rules as input
Generalize rules with named “wildcards” ({bed})
Multiple input and output files
Named input and output files
How to create output from input
Refer to input and output in the shell command
Refer by name

Show config and Snakefile example

WAGENINGEN
UNIVERSITY & RESEARCH

Job execution

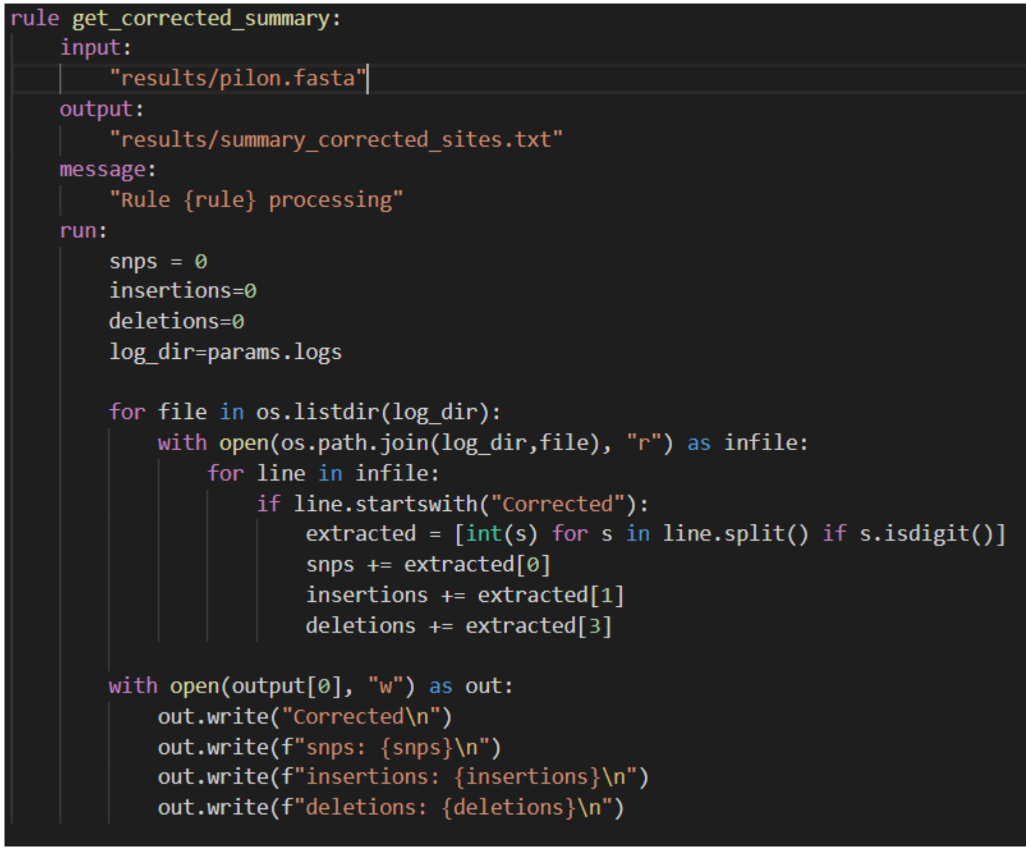
- A job (rule) is executed if
  - output file is target (rule all) and does not exist
  - output file needed by another executed job and does not exist
  - input file newer than output file
  - input file will be updated by other job
  - Snakemake is specifically told to run it



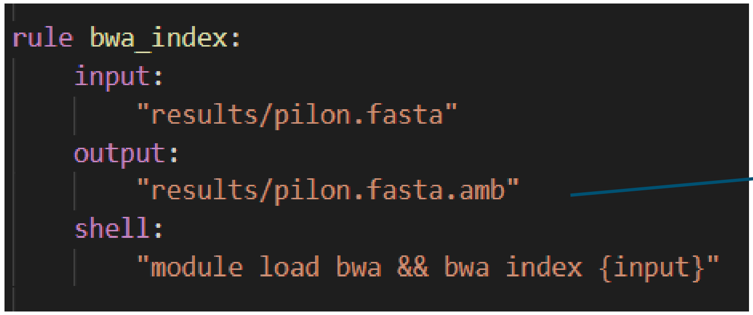
Rule all:
report.txt
Variants.vcf
Variants.vcf.tbi

Advantages

- Python based
- The pipeline contains the programs and commands you are used to
- Can execute different types of commands:
  - Bash/command line tools
  - Python
  - Rscript
- Computes dependencies between rules



```
rule get_corrected_summary:
    input:
        "results/pilon.fasta"
    output:
        "results/summary_corrected_sites.txt"
    message:
        "Rule {rule} processing"
    run:
        snps = 0
        insertions=0
        deletions=0
        log_dir=params.logs

        for file in os.listdir(log_dir):
            with open(os.path.join(log_dir,file), "r") as infile:
                for line in infile:
                    if line.startswith("Corrected"):
                        extracted = [int(s) for s in line.split() if s.isdigit()]
                        snps += extracted[0]
                        insertions += extracted[1]
                        deletions += extracted[3]

        with open(output[0], "w") as out:
            out.write("Corrected\n")
            out.write(f"snps: {snps}\n")
            out.write(f"insertions: {insertions}\n")
            out.write(f"deletions: {deletions}\n")
```



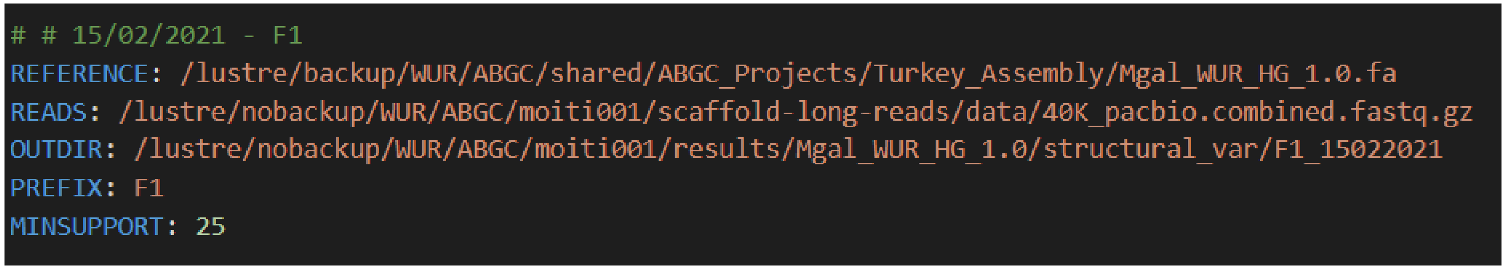
```
rule bwa_index:
    input:
        "results/pilon.fasta"
    output:
        "results/pilon.fasta.amb"
    shell:
        "module load bwa && bwa index {input}"
```

rule example:
    input:
        rules.bwa_index.output
    output:
        “example.txt”
    shell:
        “some command {input} > {output}”

bwa_index will run before *example*

Advantages

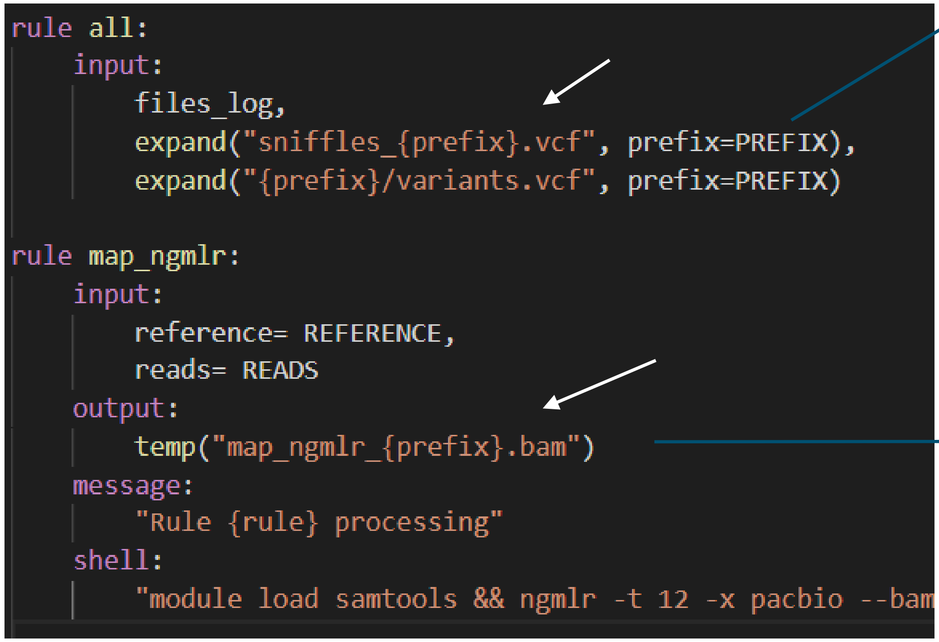
- Run several instances of the same pipeline without conflict:
  - Different config files for each run
  - Specify different working directories



```
# # 15/02/2021 - F1
REFERENCE: /lustre/backup/WUR/ABGC/shared/ABGC_Projects/Turkey_Assembly/Mgal_WUR_HG_1.0.fa
READS: /lustre/nobackup/WUR/ABGC/moiti001/scaffold-long-reads/data/40K_pacbio.combined.fastq.gz
OUTDIR: /lustre/nobackup/WUR/ABGC/moiti001/results/Mgal_WUR_HG_1.0/structural_var/F1_15022021
PREFIX: F1
MINSUPPORT: 25
```

Advantages

- **Wildcards** – allow us to generalize rules



```
rule all:
    input:
        files_log,
        expand("sniffles_{prefix}.vcf", prefix=PREFIX),
        expand("{prefix}/variants.vcf", prefix=PREFIX)

rule map_ngmlr:
    input:
        reference= REFERENCE,
        reads= READS
    output:
        temp("map_ngmlr_{prefix}.bam")
    message:
        "Rule {rule} processing"
    shell:
        "module load samtools && ngmlr -t 12 -x pacbio --bam
```

If PREFIX = parent1

Rule all:
    input:
        sniffles_parent1.vcf
        parent1/variants.vcf

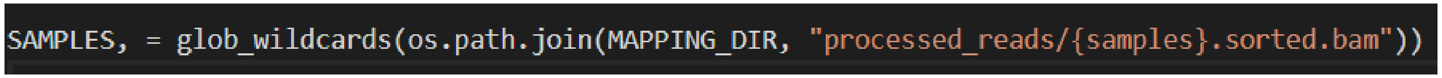
The wildcard “prefix” will be “parent1”

Output of map_ngmlr will be map_ngmlr_**parent1**.bam

{prefix} is resolved from the rule all input files

Advantages

- **glob_wildcards** - infer name of wildcards from files in the system



```
SAMPLES, = glob_wildcards(os.path.join(MAPPING_DIR, "processed_reads/{samples}.sorted.bam"))
```

Looks into the processed_reads directory for files ending in .sorted.bam and create a list SAMPLES with everything in the filename before .sorted.bam

Files:

sample1.sorted.bam, sample2.sorted.bam, sample3.sorted.bam, sample4.sorted.bam

SAMPLES = [sample1, sample2, sample3, sample4] – list of values for wildcard {samples}

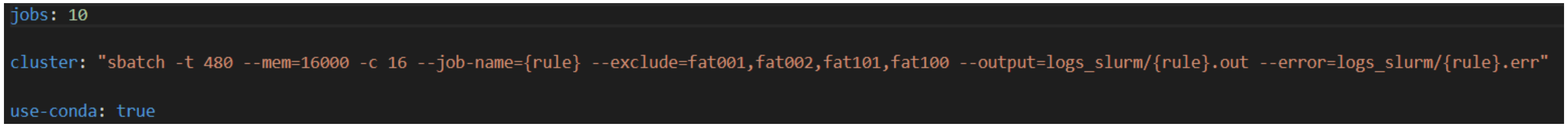
- **expand** – combine different variables, expand file names, etc

expand("sniffles_{prefix}.vcf", prefix=SAMPLES)

Output: [sniffles_sample1.vcf, sniffles_sample2.vcf, sniffles_sample3.vcf, sniffles_sample4.vcf]

Advantages

- Handles **job submission** to the HPC (through a slurm config file)



```
jobs: 10

cluster: "sbatch -t 480 --mem=16000 -c 16 --job-name={rule} --exclude=fat001,fat002,fat101,fat100 --output=logs_slurm/{rule}.out --error=logs_slurm/{rule}.err"

use-conda: true
```

  - Determines which rules to run and auto-submits jobs

- Ability to **group rules** so they run together (saves HPC queue time and limits the number of jobs)

- **Parallelization** – determines which jobs can be run at the same time

- **Logging**

- Doesn’t re-run rules that already ran successfully

- Use **conda environments and HPC modules**

- Easy to share pipelines with others

- Creates directories as needed

WAGENINGEN
UNIVERSITY & RESEARCH

How to run

Snakemake --profile profile1 [other flags]

- Snakemake will look in the current directory for a file named Snakefile
- It will determine which rules to run and in which order
- It will submit jobs to slurm with the slurm parameters specified in the profile1 config file

Snakemake --profile profile1 --configfile my_config.yaml [other flags]

Run localy:

Snakemake --cores 1

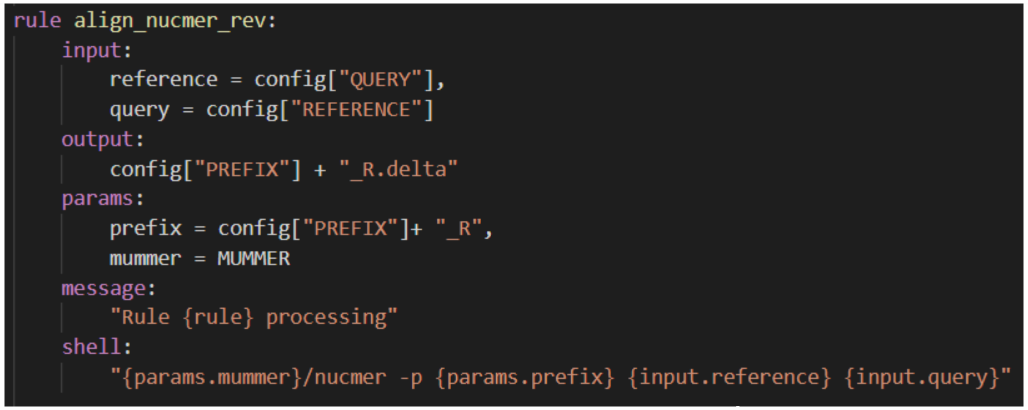
How to run

Snakemake –np

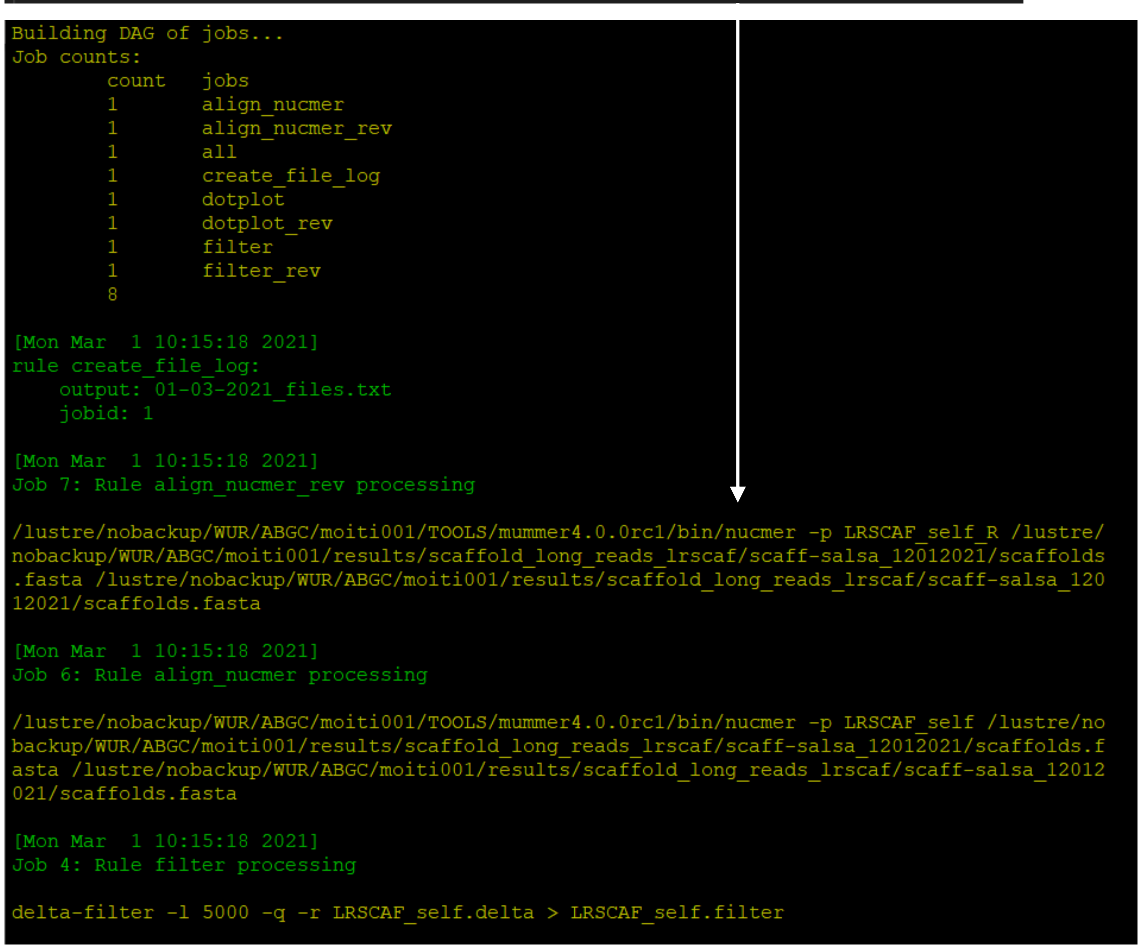
Shows which jobs will run and the resolved file names

(instead of {input} and {output})

- **ALWAYS** run this first and check the commands and file names for mistakes



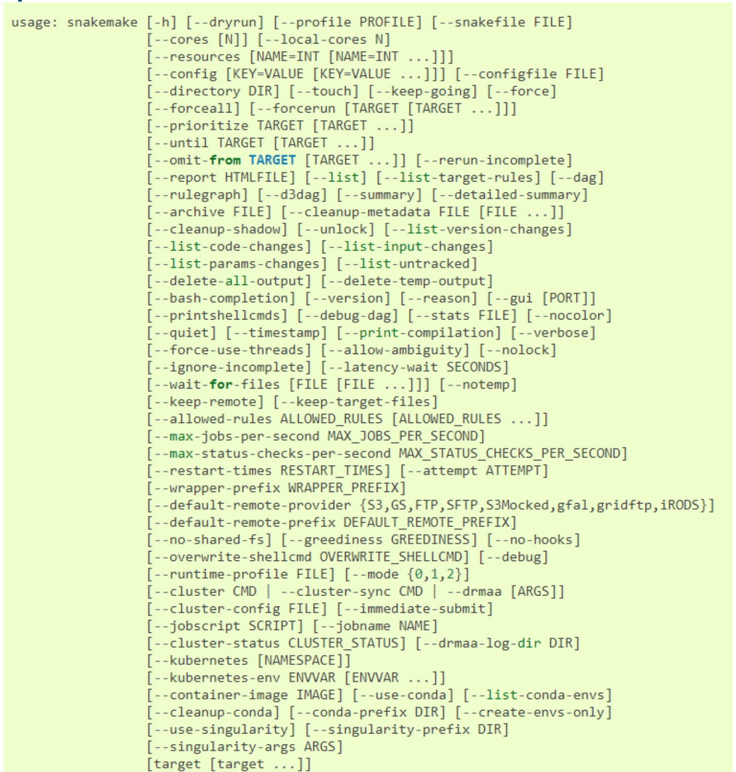
```
rule align_nucmer_rev:
    input:
        reference = config["QUERY"],
        query = config["REFERENCE"]
    output:
        config["PREFIX"] + "_R.delta"
    params:
        prefix = config["PREFIX"]+ "_R",
        mummer = MUMMER
    message:
        "Rule {rule} processing"
    shell:
        "{params.mummer}/nucmer -p {params.prefix} {input.reference} {input.query}"
```



```
Building DAG of jobs...
Job counts:
        count   jobs
        1       align_nucmer
        1       align_nucmer_rev
        1       all
        1       create_file_log
        1       dotplot
        1       dotplot_rev
        1       filter
        1       filter_rev
        8

[Mon Mar  1 10:15:18 2021]
rule create_file_log:
    output: 01-03-2021_files.txt
    jobid: 1

[Mon Mar  1 10:15:18 2021]
Job 7: Rule align_nucmer_rev processing

/lustre/nobackup/WUR/ABGC/moiti001/TOOLS/mummer4.0.0rc1/bin/nucmer -p LRSCAF_self_R /lustre/
nobackup/WUR/ABGC/moiti001/results/scaffold_long_reads_lrscaf/scaff-salsa_12012021/scaffolds
.fasta /lustre/nobackup/WUR/ABGC/moiti001/results/scaffold_long_reads_lrscaf/scaff-salsa_120
12021/scaffolds.fasta

[Mon Mar  1 10:15:18 2021]
Job 6: Rule align_nucmer processing

/lustre/nobackup/WUR/ABGC/moiti001/TOOLS/mummer4.0.0rc1/bin/nucmer -p LRSCAF_self /lustre/no
backup/WUR/ABGC/moiti001/results/scaffold_long_reads_lrscaf/scaff-salsa_12012021/scaffolds.f
asta /lustre/nobackup/WUR/ABGC/moiti001/results/scaffold_long_reads_lrscaf/scaff-salsa_12012
021/scaffolds.fasta

[Mon Mar  1 10:15:18 2021]
Job 4: Rule filter processing

delta-filter -l 5000 -q -r LRSCAF_self.delta > LRSCAF_self.filter
```

Other useful flags

- *--configfile config_1.yaml* : use config_1.yaml as config file (with files to be used, etc)
- *--rerun-incomplete* : if one of the jobs fails and is marked as incomplete you can run only that one
- *--ignore-incomplete* : if the file is created outside of Snakemake, Snakemake might think it's not complete/correct. Use those files anyway
- *--forceall* : run all rules
- *--until <rule name>* : run all rules until (including) this rule



```
usage: snakemake [-h] [--dryrun] [--profile PROFILE] [--snakefile FILE]
                 [--cores [N]] [--local-cores N]
                 [--resources [NAME=INT [NAME=INT ...]]]
                 [--config [KEY=VALUE [KEY=VALUE ...]]] [--configfile FILE]
                 [--directory DIR] [--touch] [--keep-going] [--force]
                 [--forceall] [--forcerun [TARGET [TARGET ...]]]
                 [--prioritize TARGET [TARGET ...]]
                 [--until TARGET [TARGET ...]]
                 [--omit-from TARGET [TARGET ...]] [--rerun-incomplete]
                 [--report HTMLFILE] [--list] [--list-target-rules] [--dag]
                 [--rulegraph] [--d3dag] [--summary] [--detailed-summary]
                 [--archive FILE] [--cleanup-metadata FILE [FILE ...]]
                 [--cleanup-shadow] [--unlock] [--list-version-changes]
                 [--list-code-changes] [--list-input-changes]
                 [--list-params-changes] [--list-untracked]
                 [--delete-all-output] [--delete-temp-output]
                 [--bash-completion] [--version] [--reason] [--gui [PORT]]
                 [--printshellcmds] [--debug-dag] [--stats FILE] [--nocolor]
                 [--quiet] [--timestamp] [--print-compilation] [--verbose]
                 [--force-use-threads] [--allow-ambiguity] [--nolock]
                 [--ignore-incomplete] [--latency-wait SECONDS]
                 [--wait-for-files [FILE [FILE ...]]] [--notemp]
                 [--keep-remote] [--keep-target-files]
                 [--allowed-rules ALLOWED_RULES [ALLOWED_RULES ...]]
                 [--max-jobs-per-second MAX_JOBS_PER_SECOND]
                 [--max-status-checks-per-second MAX_STATUS_CHECKS_PER_SECOND]
                 [--restart-times RESTART_TIMES] [--attempt ATTEMPT]
                 [--wrapper-prefix WRAPPER_PREFIX]
                 [--default-remote-provider {S3,GS,FTP,SFTP,S3Mocked,gfal,gridftp,iRODS}]
                 [--default-remote-prefix DEFAULT_REMOTE_PREFIX]
                 [--no-shared-fs] [--greediness GREEDINESS] [--no-hooks]
                 [--overwrite-shellcmd OVERWRITE_SHELLCMD] [--debug]
                 [--runtime-profile FILE] [--mode {0,1,2}]
                 [--cluster CMD | --cluster-sync CMD | --drmaa [ARGS]]
                 [--cluster-config FILE] [--immediate-submit]
                 [--jobscript SCRIPT] [--jobname NAME]
                 [--cluster-status CLUSTER_STATUS] [--drmaa-log-dir DIR]
                 [--kubernetes [NAMESPACE]]
                 [--kubernetes-env ENVVAR [ENVVAR ...]]
                 [--container-image IMAGE] [--use-conda] [--list-conda-envs]
                 [--cleanup-conda] [--conda-prefix DIR] [--create-envs-only]
                 [--use-singularity] [--singularity-prefix DIR]
                 [--singularity-args ARGS]
                 [target [target ...]]
```

Other information

Not all rules need to be run in the HPC, **local rules** can be specified in the snakefile

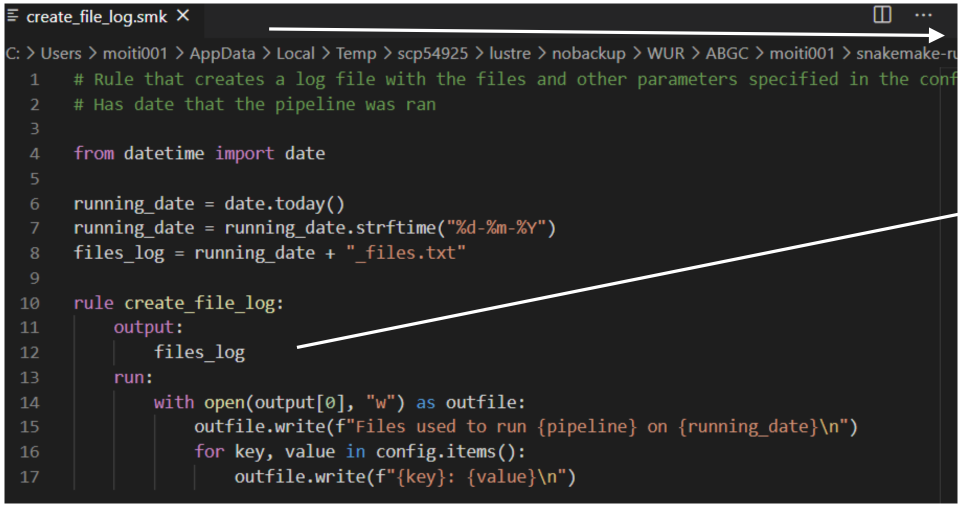
    localrules: <rule name>

Mark intermediate files as temporary – they'll be removed once they're used by the next rule

    Output:
        temp("intermediate_file.bam"**)**

Possible to **reuse rules** by saving them to a file and calling that file from the Snakefile

Rule file

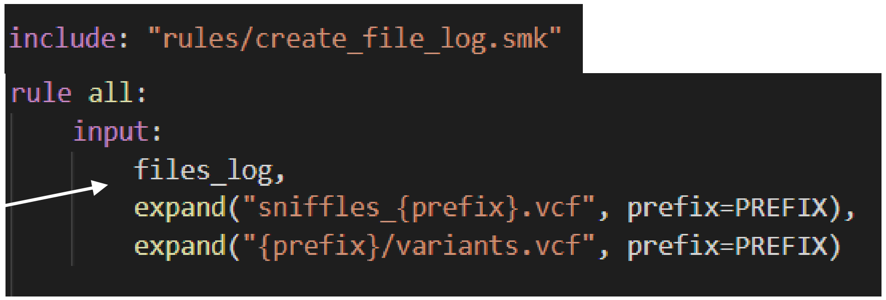


create_file_log.smk

C: > Users > moiti001 > AppData > Local > Temp > scp54925 > lustre > nobackup > WUR > ABGC > moiti001 > snakemake-ru

```
# Rule that creates a log file with the files and other parameters specified in the conf
# Has date that the pipeline was ran

from datetime import date

running_date = date.today()
running_date = running_date.strftime("%d-%m-%Y")
files_log = running_date + "_files.txt"

rule create_file_log:
    output:
        files_log
    run:
        with open(output[0], "w") as outfile:
            outfile.write(f"Files used to run {pipeline} on {running_date}\n")
            for key, value in config.items():
                outfile.write(f"{key}: {value}\n")
```

Snakefile



```
include: "rules/create_file_log.smk"

rule all:
    input:
        files_log,
        expand("sniffles_{prefix}.vcf", prefix=PREFIX),
        expand("{prefix}/variants.vcf", prefix=PREFIX)
```

Other information

Good to run in a Screen session

screen –S snakemake_run

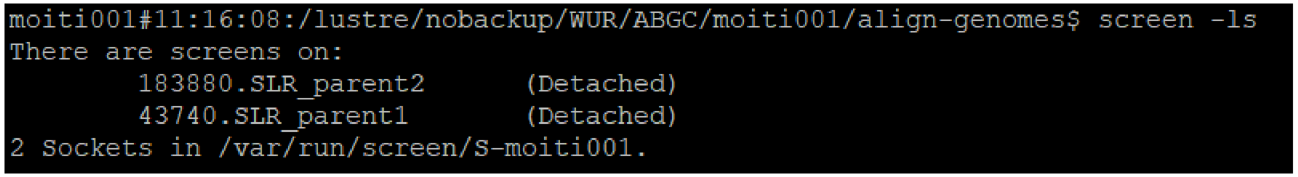
Snakemake stays “active” in the shell while the pipeline runs. If you connection goes down, snakemake will stop (but active jobs continue running).

Using screen you can leave the screen session while snakemake runs and there’s no danger that it will stop if your connection goes down.

I created a screen session named SLR_parent2 where my snakemake pipeline is running

Screen –S SLR_parent2

screen –ls shows me the screen sessions I’ve created

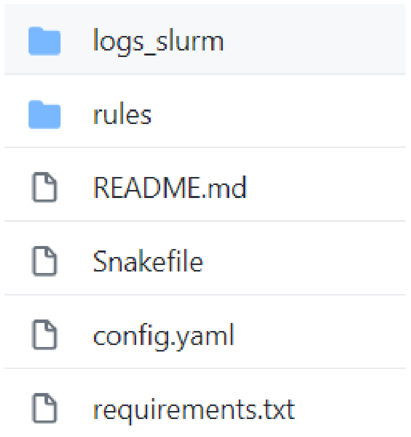


```
moiti001#11:16:08:/lustre/nobackup/WUR/ABGC/moiti001/align-genomes$ screen -ls
There are screens on:
        183880.SLR_parent2      (Detached)
        43740.SLR_parent1       (Detached)
2 Sockets in /var/run/screen/S-moiti001.
```

screen –r 183880.SLR_parent2 takes me to that screen session

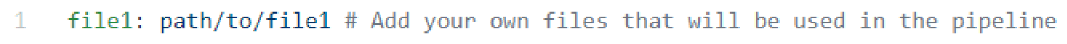
Snakemake template

https://github.com/CarolinaPB/snakemake-template



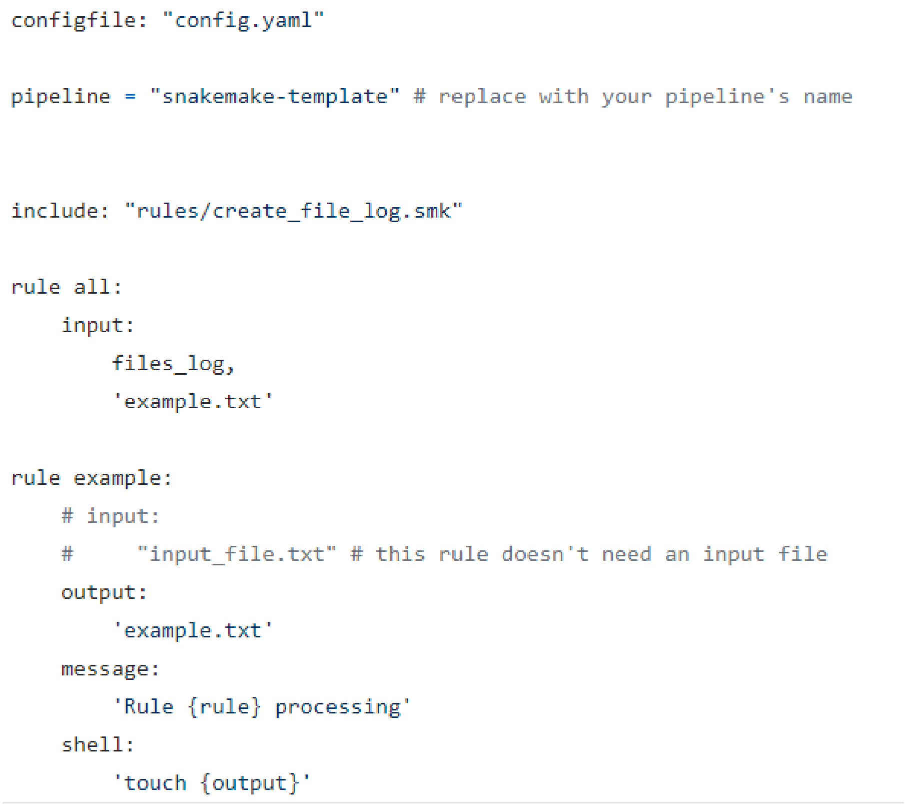
- logs_slurm
- rules
- README.md
- Snakefile
- config.yaml
- requirements.txt

config.yaml



```
file1: path/to/file1 # Add your own files that will be used in the pipeline
```

Snakefile



```
configfile: "config.yaml"

pipeline = "snakemake-template" # replace with your pipeline's name


include: "rules/create_file_log.smk"

rule all:
    input:
        files_log,
        'example.txt'

rule example:
    # input:
    #     "input_file.txt" # this rule doesn't need an input file
    output:
        'example.txt'
    message:
        'Rule {rule} processing'
    shell:
        'touch {output}'
```

Questions?