THE UNIVERSITY OF UTAH

# Workflow Managers
## Snakemake and Nextflow

Brett Milash

Center for High Performance Computing

University of Utah

# Logging in to Summit

Using XSEDE Credentials

1. ssh –l *your_xsede_login* login.xsede.org
   (enter your password, then 2FA using DUO)
2. gsissh rmacc-summit
   This gets you onto a login node on Summit
3. ssh scompile
   to get onto a compile node.

Using a temporary account:

1. ssh user0040@tlogin1.rc.colorado.edu
2. ssh scompile

# Install instructions

```
$ source ~bmilash@xsede.org/rmacc-setup
Loading modules.
Altering PATH.
Testing that nextflow is on path:
~/bin/nextflow
Testing snakemake version.
5.2.2
Cloning github repository.
Cloning into 'rmacc-workflows'...
remote: Counting objects: 38, done.
remote: Compressing objects: 100% (29/29), done.
Unpacking objects:   7% (3/38)
$ cd rmacc-workflows
```

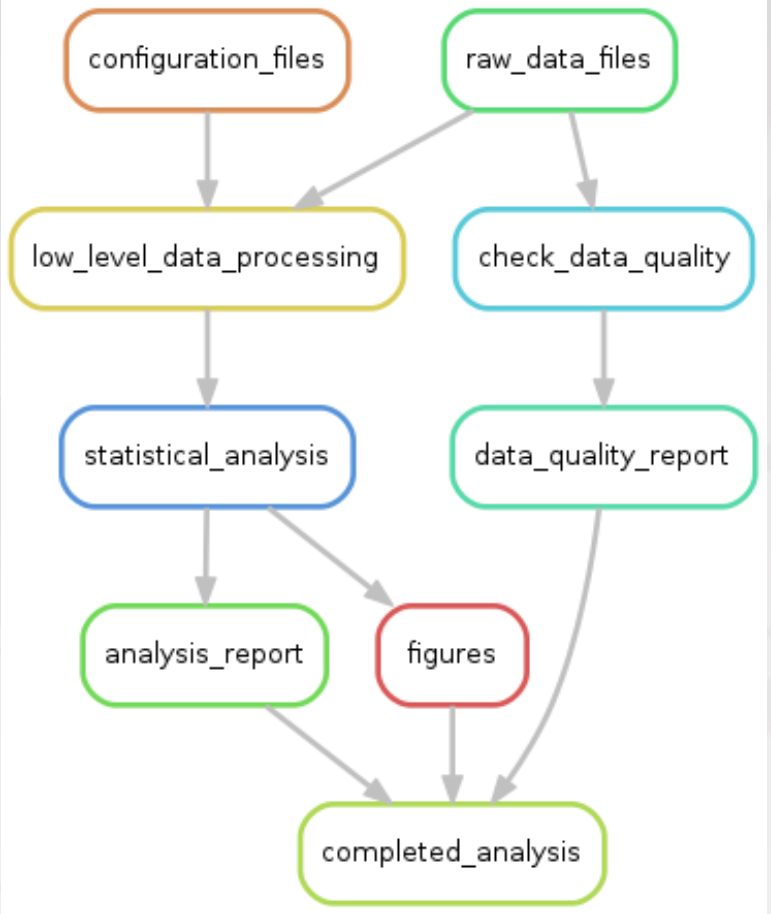
# Documentation

- Snakemake
  - https://snakemake.readthedocs.io/en/stable/
- Nextflow
  - https://www.nextflow.io/

# Workflow managers – what are they?

- Tools that conduct complex analyses
  - Track dependancies from analysis results back to raw data files
  - Statements to carry out analysis step-by-step
- Can be configured for local, cluster, remote execution
- Common in bioinformatics, not (necessarily) bioinformatics-specific
- Numerous! Over 100 in existence

https://github.com/pditommaso/awesome-pipeline



# Why use them?

- Reproducibility
- Portability between clusters, institutions
- Modularity – divide and conquer!
- Re-use
- Convenience

# (My) Criteria for selection

- Actively used and developed
- Native SLURM support
- No significant system administration support required
- General purpose (i.e. not just for bioinformatics)
- Significant functionality bang for your learning buck

# Snakemake and Nextflow

## Snakemake

- Similar to “make”
- Python
- Rules
- Files
- Execution:
  - Local (shell, python)
  - SLURM
  - LSF
- Other interesting bits:
  - Wildcards
  - Graphical output
  - Modular workflows
  - Python API
  - Common Workflow Language

## Nextflow

- Dataflow programming model
- Groovy (java 1.8)
- Processes
- Channels (files, values, pipes, sets of these … )
- Execution:
  - Local (shell, groovy, python, R, perl, … )
  - SLURM
  - LSF
  - HTCondor (Open Science Grid)
  - Sun Grid Engine
  - Kubernetes
  - AWS Batch
- Other interesting bits:
  - Common Workflow Language (in alpha)
  - *Process definitions part of workflow*

# Snakemake: a better make

Classical makefile:

Rule

Dependancies (input)

```
hello_world: hello_world.o
	gcc -o hello_world hello_world.o

hello_world.o: hello_world.c hello_world.h
	gcc -c hello_world.c
```

Target (output)

Action

# Simple snakefile example

```
rule link:
        input: "hello_world.o"
        output: "hello_world"
        shell: """
                module load gcc/6.1.0
                gcc -o {output} {input}
                """

rule compile:
        input: source="hello_world.c",headers=["hello_world.h",]
        output: "hello_world.o"
        shell: """
                module load gcc/6.1.0
                gcc -c {input.source}
                """
```

Rules have:

- inputs
- outputs
- actions (shell or python)
- names

Rules are:

- linked implicitly
- (or explicitly)
- executed in parallel if possible
- executed locally or on a cluster

# Snakemake exercise 1

1. In the “data” directory, manually run “fastqc --noextract” on one of the .fastq.gz files, to see what output files are created.
2. Create a snakefile with a single rule that runs “fastqc --noextract” on just one fastq file. Choose any one file, and hard-code its name into your snakefile as the input. Also hard-code the output file names. *Hint: output file names are quoted, and multiple output files are comma-separated.*
3. Run “snakemake –s *your_snakefile*”

```
rule fastqc:
        input: "SRR5934916.fastq.gz"
        output: "SRR5934916_fastqc.zip", "SRR5934916_fastqc.html"
        # Here's an optional message, printed to standard output:
        message: "Running Fastqc on input file {input}."
        shell:  "fastqc --noextract {input}"
```

# Snakemake wildcards

- Wildcards: filename patterns enable generalizable rules
  - These make workflows reusable
  - Parallelization!
- The trick:
  - Create one rule that handles a single input -> output action. This acts as a template.
  - Create a second rule whose input lists all the required output files using snakemake's  expand() function. This acts as a gatekeeper.
- The catch: you can't execute the template rule directly, only the gatekeeper rule.

# Snakemake wildcard example

```
# Calculate the number of lines and MD5 checksum for every .fastq.gz file.
# First, generate a list of the sample names embedded in the file names:
import os
samples=[f.split('.')[0] for f in os.listdir('.') if f.endswith('.fastq.gz')]

rule all_sizes_and_checksums:
        input: expand("{sample}.{format}", sample=samples, format=["linecount","md5"])
        output: touch("sizes_and_checksums.done")    # Creates empty output file

rule one_size_and_checksum:
        input: "{sample}.fastq.gz"
        output: linecount="{sample}.linecount", checksum="{sample}.md5"
        shell: """
                gunzip -c {input} | wc -l > {output.linecount}
                md5sum {input} > {output.checksum}
                """
```

# Snakemake exercise 2

Alter your snakefile ( or use solutions/exercise1.snakefile ) to:

- Process all fastq files with fastqc using a wildcard.
- Once fastqc has been run on all fastq files, run the command “multiqc .” to summarize all the fastqc output files. This will produce multiqc_report.html and multiqc_data.
- You can do this with 2 rules (one template, one “gatekeeper”), or 3 rules (one template, one gatekeeper, and a third to run multiqc).

# Snakemake exercise 2 solution ( 2 rules )

```
# Generate a list of the sample names embedded in the file names.
import os
samples=[f.split('.')[0] for f in os.listdir('.') if f.endswith('.fastq.gz')]

rule run_multiqc:
        input: expand("{sample}_fastqc.{format}", sample=samples, format=["zip","html"])
        output: "multiqc_report.html", "multiqc_data"
        shell:  """
                module load multiqc
                multiqc .
                """

rule fastqc_one_sample:
        input: "{sample}.fastq.gz"
        output: "{sample}_fastqc.zip", "{sample}_fastqc.html"
        shell:  """
                module load fastqc
                fastqc --noextract {input}
                """
```

# Snakemake on a cluster

- Any snakemake task can run on a cluster:

```
snakemake -s snakefile --cluster-config cluster.yaml --jobs 20
```

- Cluster configuration file:

```
# cluster.yaml - cluster configuration for my snakemake job.
__default__:
    partition: slurm_parition
    account: slurm_account
    time: 3:00:00
    nodes: 1
```

- Configuration can be in JSON or YAML format
- Can provide default and rule-specific configurations
- Can specify ”local” rules:

```
localrules: run_multiqc
```

# Snakemake exercise 3

- Create a cluster configuration file cluster.yaml:

```
# cluster.yaml - cluster configuration for my snakemake job.
__default__:
    partition: shas
    qos: debug
    time: 0:10:00
    nodes: 1
```

- Run your snakefile (or use solutions/exercise2-2.snakefile) using your cluster configuration file:

```
snakemake -s your_snakefile --cluster-config cluster.yaml --cluster
   sbatch --jobs 5
```

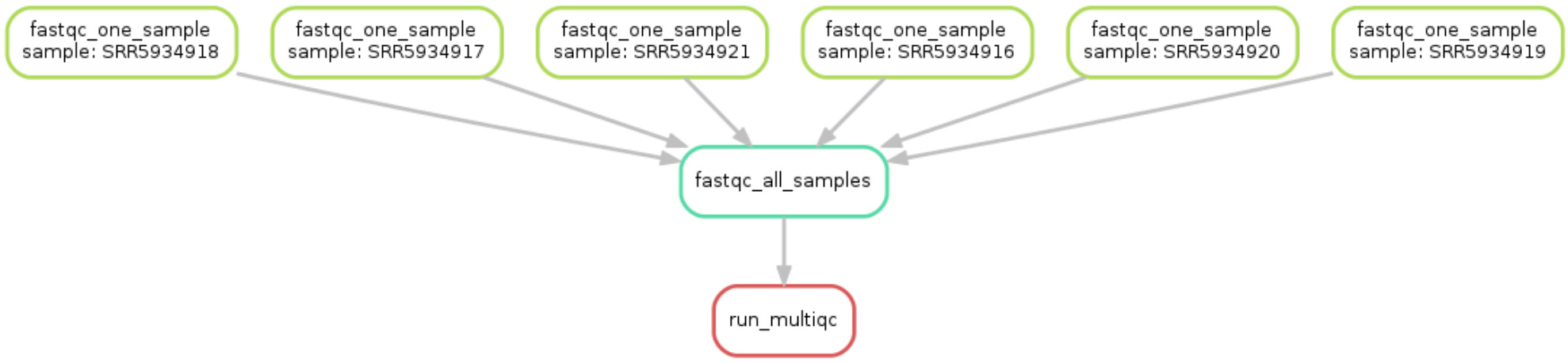
# Snakemake wrap-up

- Modular workflows

```
include: "path/to/other/snakefile"
```

- Graphical output

```
snakemake –s snakefile --rulegraph | dot –Tpng > rulegraph.png
snakemake –s snakefile --dag | dot –Tpng > dag.png
```

# Nextflow basics

Nextflow is based on the dataflow programming model.

Imagine your workflow as an assembly line built from:

- Channels
  - Carry data around
  - Not just files – also values, streaming data (pipes), “sets” of items -> data types
  - “Operators” allow channel filtering, forking, combining, transforming, …
- Processes
  - Input from channel(s)
  - Output to channel(s)
  - Scripts: shell, also python, perl, R, …
- Executors
  - “Plug-ins” that control where the work is done
  - Local (by default), also SLURM, Open Science Grid , SGE, LSF, PBS, etc …

# More nextflow basics

- Work directory
  - Each process invocation happens in its own directory
  - Input files linked there
  - Result files left there unless you explicitly “publish” them elsewhere
- Nextflow scripts written in groovy
  - // one-line comments
  - /* multi-line comments */
  - Start with “#!/usr/bin/env nextflow” for an executable nextflow script
  - http://docs.groovy-lang.org/docs/latest/html/documentation/

# Nextflow compile example

```
$ ls
a.c  b.c  main.c compile1.nf
$ cat compile1.nf
#!/usr/bin/env nextflow
c_files_channel = Channel.fromPath( "*.c" )
c_files_channel.subscribe { println "$it" }
$ nextflow run compile1.nf
N E X T F L O W  ~  version 0.29.1
Launching `./compile1.nf` [distraught_shockley] - revision:
1115256fab
.../examples/nextflow_ex1/a.c
.../examples/nextflow_ex1/b.c
.../examples/nextflow_ex1/main.c
```

# Nextflow compile example (2)

```
$ cat compile2.nf
#!/usr/bin/env nextflow
c_files_channel = Channel.fromPath( "*.c" )

process compile {
        module 'gcc/4.9.2'
        input: file c_file from c_files_channel
        output: file '*.o' into result
        script: "gcc -c -g ${c_file}"
}

result.subscribe { println "Compiled $it" }
```

# Nextflow compile example (2)

```
$ nextflow run compile2.nf
N E X T F L O W  ~  version 0.29.1
Launching `./compile2.nf` [angry_legentil] - revision: 922ae15375
[warm up] executor > local
[ea/588755] Submitted process > compile (3)
[d7/6bbcd1] Submitted process > compile (2)
[06/c5c40b] Submitted process > compile (1)
Compiled .../work/ea/5887555a65533fdccd0065a0673efa/main.o
Compiled .../work/d7/6bbcd194c33fe0a590c937eeb11ee3/b.o
Compiled .../work/06/c5c40b31665ca2eebba39acf0319bb/a.o
```

# Nextflow compile example (3)

```
$ cat compile2.nf
#!/usr/bin/env nextflow
c_files_channel = Channel.fromPath( "*.c" )

process compile {
        module 'gcc/6.1.0'
        input: file c_file from c_files_channel
        output: file '*.o' into result
        script: "gcc -c -g ${c_file}"
}

result.subscribe { println "Compiled $it" }
```

# Nextflow exercise 1

1. In the data directory, create a nextflow script that:
   a) Creates a channel that lists the .fastq.gz files
   b) Subscribes to that channel, and print the values from it.
2. Add to your nextflow script a process that:
   a) Runs “fastqc --noextract” on each .fastq.gz file from the channel
   b) Sends the name of the .zip and .html files generated to a result channel. You will need to use a set to do this:

      ```
      set file('*.html'), file('*.zip') into result_channel
      ```

   c) Run your script with “nextflow run *your_script_name* –resume” .
   d) Re-run the script, and observe how the output changes.

# Nextflow Exercise 2

- Alter your nextflow file from exercise 1 to publish the FastQC files to a directory named QcFiles, and then run ”multiqc QcFiles” on that directory.
- Multiqc will produce “multiqc_report.html” and “multi_data”. Publish those files to the QcFiles directory.

THE UNIVERSITY OF UTAH

# Thank you!