Workflow Management Software

CMSE 890-602

Workflow languages

- Common Workflow Language
- Workflow Description Language
- Yet Another Workflow Language (business focus)
- **All require interpreters to run them!**

Workflow managers

- SnakeMake
  - Python based
- NextFlow
  - Java (Groovy) based
- Pegasus
- SciPipe
  - Go based
- Galaxy
  - biomedical focus
- Parsl
  - parallel Python focus

Why use a workflow manager?

- Designed to work on many different systems
- Automatic interface with job schedulers on HPC
- Helps manage file paths
- Automated logging
- Environment control (containers etc)

A segue into bash scripting

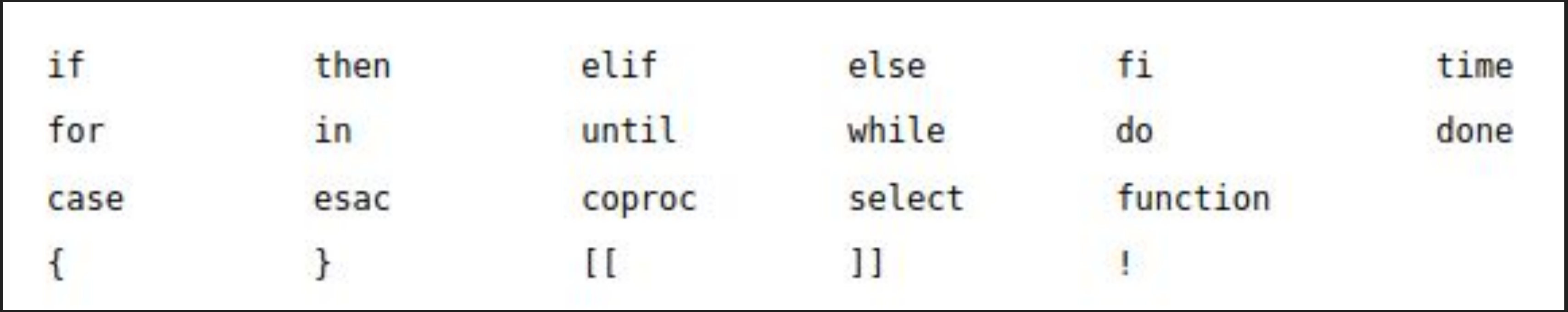
- Bash is the most common Linux terminal language
- Nearly all high-performance computing systems run Linux
- You probably want to run your workflow on a HPC eventually
- So you should learn enough bash to get by!
- https://www.gnu.org/software/bash/manual/bash.html

Typical command syntax:

```
command --option1 --option2 optargument -opt3 argument1 argument2
```

A segue into bash scripting

Reserved words:



if	then	elif	else	fi	time
for	in	until	while	do	done
case	esac	coproc	select	function	
{	}	[[	]]	!	

Basic loop

```
for name [ [in [words …] ] ; ] do commands; done

for i in 1 2 3 4 5; do echo $i; done
```

- If you are doing much more bash than this, you may want to consider a full scripting language instead
- Basic launch commands for most software *should* be OS-independent

Some bash examples

Check if a file exists

```
#!/bin/bash
file="example.txt"
# Check if the file exists
if [ -e "$file" ]; then
echo "File exists: $file"
else
echo "File not found: $file"
fi
```

Some bash examples

Convert all file names in a directory to lower case

```
#!/bin/bash
directory="$1"
if [ -z "$directory" ]; then
echo "Usage: $0 <directory>"
exit 1
fi

if [ ! -d "$directory" ]; then

echo "Error: '$directory' is not a valid directory."
exit 1
fi

cd "$directory" || exit 1

for file in *; do
if [ -f "$file" ]; then
newname=$(echo "$file" | tr 'A-Z' 'a-z')
[ "$file" != "$newname" ] && mv "$file" "$newname"
fi
done
```

Common workflow manager features

- Input-output based flow (DAG!)
- Shell scripting support
    - Bash, etc
- Scripting language support
    - Python, R, etc
- Custom format or language for configuration
- Automatic parallelization of jobs
- Container support
- DAG visualization

SnakeMake

- SnakeMake is an *extension* of Python
- Scales from single to multiple compute cores
- Automatically links inputs and outputs based on “rules” to create a DAG
- Can handle installation of environments via conda integration
- Handles Jupyter notebooks internally
- Big downside: need to list *all* inputs/outputs

SnakeMake file composition

1. Constants definitions `CONSTANT = value`
2. Rules `rule` *`name`*`:`
   a. Input files `input:`
   b. Output files `output:`
   c. Conda environment definition `conda:`
   d. Shell commands `shell:`
   e. Script files (bash, python, R, Julia, Rust…) `script:`
   f. Compute threads `thread:`
   g. Resources e.g. memory usage `resources:`
   h. Print a message `message:`
   i. And more!

SnakeMake rule syntax

```
rule NAME:
    input: "path/to/inputfile",
           "path/to/other/inputfile"
    output: "path/to/outputfile",
            "path/to/another/outputfile"
    shell: "somecommand {input} {output}"
```

Execute Python code inline:

```
    run:
        for f in input:
            ...
            with open(output[0], "w") as out:
                out.write(...)
        with open(output.somename, "w") as out:
            out.write(...)
```

Common default rule example:

```
rule all:
 input:
   expand("{dataset}/file.A.txt", dataset=DATASETS)
```

For each dataset in DATASETS, a corresponding rule is run

At the top of the Snakefile (first rule is always executed)

SnakeMake wildcards

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

Wildcards

Wildcard “group” passed to shell

- Wildcards take some work to understand!
- Be careful using multiple wildcards in a row
- Wildcards act as *regular expressions* of type .+

SnakeMake aggregation

```
rule aggregate:
    input:
        expand("{dataset}/a.txt", dataset=DATASETS)
    output:
        "aggregated.txt"
    shell:
        ...
```

Note {dataset} is not an empty .+ wildcard here. DATASETS is defined at the top of the Snakefile as a global variable e.g. `DATASETS = [“first”, “second”]`

Modular snakemake

- Wrappers
  - Often premade, web-distributed workflow files
- `Include` syntax
  - Allows adding other Snakefiles to the workflow
  - `include: "path/to/other/snakefile"`
- Import Snakefiles as modules and use specific rules from them
  - `module other_workflow:`
  - `    snakefile:`
  - `        # here, plain paths, URLs and the special markers for code hosting providers (see below) are possible.`
  - `        "other_workflow/Snakefile"`
  -
  - `use rule * from other_workflow exclude ruleC as other_*`

Best practices

- `snakemake --lint` to check your workflow for minor errors
- `snakefmt` to format your workflow
- Use wrappers if a rule appears more than once

In-class assignment

Go to https://msu-cmse-courses.github.io/CMSE_890-602_snakemake/

Follow the directions up to the end of part 3.15

We will go through this together!

Screenshot your results for each output in part 3

Upload the screenshots to D2L. LABEL THEM WITH THE PART NUMBER e.g. 3.09

Homework

Work on your semester project!

- Start putting together the workflow using your pseudo code as a basis
  - Start with a familiar language if necessary
  - Play around with using SnakeMake for your workflow code
  - You may prefer NextFlow when we try it out next week
- Next milestone is DOCUMENTATION, Nov 19th
  - Remember to check the rubric
  - Leave placeholders for parts that require code to be run
  - Try to get a github.io site set up