Workflow Management Software

CMSE 890-402

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?

- Shell scripting is not portable- OS dependent!
  - Very difficult to make modular
- What about portable scripting languages?
  - Very hard to parallelize
  - Difficult to interact directly with the system

A segue into bash scripting

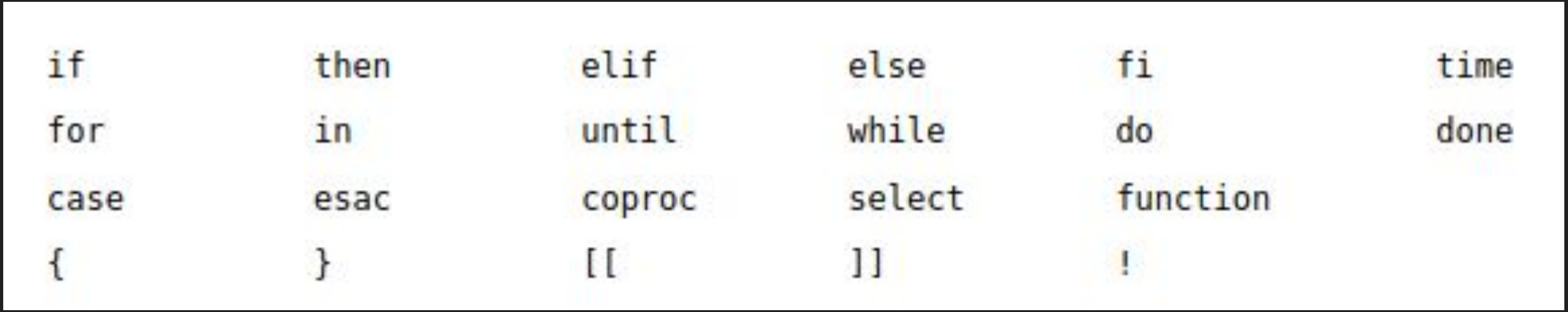
- Bash is the most common Linux terminal language
- Nearly all high-performance computing systems run Linux
- You probably want to run your workflow on a HPC eventually
- So you should learn enough bash to get by!
- https://www.gnu.org/software/bash/manual/bash.html

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

Common workflow manager features

- Input-output based flow (DAG!)
- Shell scripting support
  - Bash, etc
- Scripting language support
  - Python, R, etc
- Custom 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

```
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

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!