Makeflow Specification

August 19, 2013

`makeflow` is a *workflow engine* for executing workflows on clusters, clouds, and grids.

1 Workflow Consistency

A *workflow* is a directed acyclic bipartite finite graph $G = (V, E)$, with set of vertices $V = R \cup F$, $R \cap F = \emptyset$. We call R the set of *rules*, and F the set of *files*. For $r \in R$, and $f \in F$, if the edge $(f, r) \in E$, then we call f an *input file* for rule r, and we say that r depends on f. Likewise, if $(r, f) \in E$, then we call f an *output file*, or a *target* of r, and we say that r produces f. Further, for any $r \in R$, there is some $f \in F$ such that $(r, f) \in E$, and there is at most one such r for every f (that is, every rule has at least one output file, and no file can be the output of two different rules). It is convenient to define a special file $f_0 \in F$, that we call the *null file*. The file f_0 is an input file for every rule in R, but it is not the output of any rule.

For $r \in R$, let $S_r \subset F$ be the set of all of the input files for r, that is, $S_r = \{f \in F \mid (f, r) \in E\}$. Similarly, let $T_r = \{f \in F \mid (r, f) \in E\} \subset F$ be the set of all of the output files of r. Note that given the constraints on G, $S_r \neq \emptyset$ and $T_r \neq \emptyset$ for any $r \in R$. Also, we have that $T_{r_a} \cup T_{r_b} \neq \emptyset \Rightarrow r_a = r_b$.

The set of files can be partitioned into three sets: $F = S \cup C \cup T$. We call S the set of *source files*, which is the set of all of the files that are not an output file of some rule, $S = F \setminus \bigcup_{r \in R} T_r$. Likewise, T is the set of *sink files*, which is the set of all of the files that are not an input file of some rule, $T = F \setminus \bigcup_{r \in R} S_r$. Finally, $C = F \setminus (S \cup T)$, is the set of *intermediate* or *collectable* files.

Specifically, we can state the consistency of a workflow $G = (R \cup F, E)$ by verifying the following conditions:

- G is a directed acyclic graph.
- Every rule $r \in R$ has at least one output file.
- Each file $f \in F$ is the target of at most one rule.
- All the edges in E have the form (f, r) or (r, f), for some $f \in F$, and $r \in R$.

2 Workflow Execution

Let $\leq_G$ be the natural partial order on R defined by G. That is, for r_a, $r_b \in R$, $r_a \leq_G r_b$ if one of the following conditions is met:

- $r_a = r_b$
- $T_{r_a} \cap S_{r_b} \neq \emptyset$
- $\exists r \in R \ : \ r_a \leq_G r \leq_G r_b$

Let $s = [r_1, r_2, \ldots, r_n]$ be a sequence of all of the rules in R, with $n = |R|$ and $r_i \in R$, in which $r_i \leq_G r_j \Rightarrow i < j$. (Since $\leq_G$ is a partial order, this sequence is in general not unique.)

A *valid workflow execution trace* for sequence of rules s is a sequence of sets $D_0 \subset D_1 \subset D_2 \subset \ldots \subset D_n$, in which:

- The source files are a subset of D_0, $S \subseteq D_0$.
- The set of all of the files F is a subset of D_n, $F \subseteq D_n$.
- The next sequence element contains the output files of the corresponding rule, $T_{r_{i+1}} \subset (D_{i+1} \setminus D_i)$.

2.1 Workflow as a State Machine

Based on workflow execution traces, we can write the rules as a set of transition functions of a state machine:

$$r \ : \ \text{pow}(F) \times \mathcal{E} \mapsto \text{pow}(F),$$

in which we use the *space of possible environments*, $\mathcal{E}$, to specify arguments to r mapping that are not input files. We think of an environment $\epsilon \in \mathcal{E}$ as a collection of key-value pairs. For sequence of rules s, we say that environment $\epsilon \in \mathcal{E}$ is *compatible* with r_i if:

$$r_i(S_r \subset D_{r-1}, \epsilon) = T_r \subset D_r,$$

in which D_{r-1} and D_r both belong to some valid workflow execution trace.

3 Variable Scope Workflows in `makeflow`

There are two kinds of variable scope in `makeflow`: variables with dynamic extent and variables with rule lexical scope.

- A variable with dynamic extent is defined *outside* any rule definitions, and its value is used for any rule definition until the variable itself is redefined.
- A variable with lexical scope is binded inside a rule definition, and its value is only valid for the rule it was defined. If two variables with different scope share the same name, then the one with lexical scope *shadows* the variable with dynamic extent during the definition of the rule.

4 Specifiying Workflows in `makeflow`

`makeflow`'s language is similar to `make`. A rule r, is specified with the following general form:

ϵ environment description with dynamic extent
T_r set of output files description **colon character :** S_r set of input files description
(at character @)ϵ' environment description with rule lexical scope
(tab character) rule r description

We call the rule description the *command* of the rule. Given the constraints given for the workflow, only the description of T_r, the colon character, the tab character, and the command are required.

4.1 Comments

Any place where a newline is expected, may contain a string starting with with #, and ending with a newline. These strings are ignored by the `makeflow`.

4.2 Grammar

Next we describe the full grammar of `makeflow`:

⟨*start*⟩ ::= ⟨*rule*⟩ ⟨*start*⟩
| ⟨*environment dynamic extent*⟩ ⟨*start*⟩
| ⟨*blank line*⟩ ⟨*start*⟩
| ⟨*null*⟩

⟨*rule*⟩ ::= ⟨*output files*⟩ `:` ⟨*input files*⟩ ⟨*blank line*⟩
⟨*environment lexical scope*⟩ ⟨*tab*⟩ ⟨*command*⟩ ⟨*blank line*⟩

⟨*blank line*⟩ ::= ⟨*newline*⟩
| `#` [^\n]* ⟨*newline*⟩

⟨*environment dynamic extent*⟩ ::= ⟨*var name*⟩ ⟨*var operator*⟩ ⟨*expansible*⟩
| `export` ⟨*var name list*⟩
| `export` ⟨*var name*⟩ `=` ⟨*expansible*⟩ ⟨*blank line*⟩

⟨*environment lexical scope*⟩ ::= **@**⟨*var name*⟩ ⟨*var operator*⟩ ⟨*expansible*⟩ ⟨*blank line*⟩ ⟨*environment lexical scope*⟩
| ⟨*null*⟩

⟨*output files*⟩ ::= ⟨*file translation*⟩ ⟨*spaces*⟩ ⟨*output files*⟩
| ⟨*file translation*⟩ ⟨*null*⟩

⟨*input files*⟩ ::= ⟨*file translation*⟩ ⟨*spaces*⟩ ⟨*input files*⟩
| ⟨*file translation*⟩ ⟨*null*⟩
| ⟨*null*⟩

⟨*command*⟩ ::= ⟨*expansible*⟩ ⟨*command*⟩
| < ⟨*s-expansible*⟩ ⟨*command*⟩
| > ⟨*s-expansible*⟩ ⟨*command*⟩
| ⟨*s-expansible*⟩ ⟨*null*⟩

⟨*file transition*⟩ ::= ⟨*file*⟩
| ⟨*file*⟩ `->` ⟨*file*⟩

⟨*file*⟩ ::= ⟨*s-expansible*⟩

⟨*expansible*⟩ ::= ⟨*s-expansible*⟩ ⟨*spaces*⟩ ⟨*expansible*⟩
| ⟨*s-expansible*⟩ ⟨*null*⟩
| ⟨*null*⟩

⟨*s-expansible*⟩ ::= ⟨*escape*⟩
| ⟨*literal*⟩
| ⟨*s-quote*⟩
| ⟨*d-quote*⟩
| ⟨*var expansion*⟩

⟨*escape*⟩ ::= \ [#$\n]

⟨*literal*⟩ ::= [A-z0-9_.,?[]{}]+
| ⟨*s-quote*⟩

⟨*s-quote*⟩ ::= ' [^']* '

⟨*d-quote*⟩ ::= " ⟨*dr-quote*⟩

⟨*dr-quote*⟩ ::= ⟨*escape*⟩
| ⟨*literal*⟩ ⟨*dr-quote*⟩
| ⟨*s-quote*⟩ ⟨*dr-quote*⟩
| ⟨*var expansion*⟩
| ⟨*spaces*⟩ ⟨*dr-quote*⟩
| ⟨*empty lines*⟩ ⟨*dr-quote*⟩
| "

⟨*var expansion*⟩ ::= $ ⟨*varname*⟩
| $(⟨*varname*⟩)

⟨*var name list*⟩ ::= ⟨*var name*⟩
| ⟨*var name*⟩ ⟨*spaces*⟩ ⟨*var name list*⟩
| ⟨*null*⟩

⟨*var name*⟩ ::= [A-z0-9_.] ⟨*var name*⟩
| ⟨*var name*⟩ ⟨*null*⟩

⟨*var operator*⟩ ::= = | +=

⟨*spaces*⟩ ::= ⟨*space*⟩ ⟨*spaces*⟩
| ⟨*tab*⟩ ⟨*spaces*⟩
| ⟨*space*⟩
| ⟨*tab*⟩

⟨*empty lines*⟩ ::= ⟨*newline*⟩ ⟨*empty lines*⟩
| ⟨*null*⟩

4.3 Meaning of `export`

... - export only valid for the rules after the variable was exported.

4.4 Special Variables, Categories, and Resources

... CATEGORY, CORES, MEMORY, DISK, BATCH_OPTIONS