ducttape

Simple and Reliable Workflow management for replicable scientific computing

Caitlin Cassidy

May 7, 2017

The University of Illinois at Urbana-Champaign

Introduction: What is ducttape?

ducttape is workflow management for scientific computing in Unix.

It enables you to:

- Write tasks using bash and have dependencies among tasks automatically managed
- Track the versions of the software used, the data used, and even the task descriptions themselves in a very detailed way
- Quickly and easily run your workflow under multiple input conditions while re-using intermediate results, without copy-pasting

ducttape can be downloaded and installed at `http://jhclark.github.com/ducttape`.

The Basics

Running a simple command

Commands are encased in `tasks` and follow bash syntax.

```
task hello_world {
  echo hello
  echo >&2 $someone
}
```

Global variables are defined in a `global` block.

```
global {
  someone=world
  ducttape_structure=flat
}
```

Input and Output

Tasks can take any number of input and output files.

```
task hello_input < a=/etc/passwd b=/etc/hosts {
  echo "I will be doing nothing with files $a and $b"
  cat $a>/dev/null
  cat $b>/dev/null
}

task hello_output > x=x.txt y=y.txt {
  echo writing files $x and $y...
  echo hello > $x
  echo world > $y
}
```

Ducttape will assign the paths for output files as `./hello_output/x.txt` and `./hello_output/y.txt`.

Running tasks with dependencies

Dependencies are introduced when one task takes the output of another as input.

```
task first > out {
  for i in {1..10}; do
    echo $i >> $out
  done
}

task and_then < in=$out@first > out {
  cat < $in > $out
}
```

Using parameters

Parameters are for variables that aren't files. They are listed after inputs and outputs, using a double colon.

```
task param_step < in=/etc/passwd > out :: N=5 {
  echo "$in has $(wc -l < $in) lines"
  echo "The parameter N is $N"
  echo $N > $out
}
```

This distinction means that parameters don't introduce temporal dependencies, so `no_dep` can start running in parallel with `param_step`.

```
task no_dep :: X=$N@param_step {
  #echo "X=$N" # a bug! this would be caught by
               # ducttape's static analysis of bash
  echo "X=$X"
}
```

Advanced Features

Hyperworkflows

Experimentation often requires one to run similar sequences of tasks in a variety of configurations. To compactly represent such workflows, ducttape provides HyperWorkflows.

```
task one < in=(Size: sm=small.txt lg=large.txt) > out {
  cat < $in > $out
}

task two < in=$out@one :: N=(N: one=1 two=2) {
  head -n $N < $in
}
```

In this example `Size` is a branch point with branches `sm` and `lg`, which allows ducttape to run `one` twice: once with `small.txt` as input, and once with `large.txt`. Because `two` has a temporal dependency with `one`, it will also be run for each of `Size`'s branches.

Sequence Branch Points

Branches can also be created based on re-running the task multiple times, which can be useful when the underlying task makes calls to a random number generator

```
task run_several_times > out :: trial=(WhichTrial: 1..10) {
  # Use bash's built-in random number generator
  echo $RANDOM > $out
}
```

With branching, ducttape adds another layer to the directory structure, e.g. `./run_several_times/WhichTrial.1/out`.

Custom realization plans

ducttape allows us to specify a number of configurations in a `plan` block. The following is a plan that specifies three goals and their branches. The * indicates either a crossproduct or a glob operator. In the latter case, it indicates that all branches of a branch point should be run.

```
plan Basics {
  reach two via (Size: smaller) * (N: one two) * (M: 1..10)
  reach one, two via (Size: bigger) * (N: one) * (M: 2 8)
  reach two via (Size: bigger) * (N: *) * (M: 1)
}
```

Branch Grafting

Branch points allow you to run a single task and all of its dependents under a particular set of conditions (the realization). But what if you want some dependents to only use a specific configuration? For this, ducttape provides branch grafts.

```
task prep < in=(DataSet: train=lg.txt test=sm.txt) > out {
  cat < $in > $out
}

task trainer < in=$out@prep[DataSet:train] > model {
   cat < $in > $model
}

task tester < in=$out@prep[DataSet:test] > hyps {
  cat < $in > $hyps
}
```

Nested Branch Points

ducttape supports nesting of branch points, such that branches of one branch point can themselves consist of another barnch point.

```
task one :: o="world" {}

task two
   :: t=(Size: s="sm" m="md" l=(Set: trn tst=$o@one)) {}

task three :: a="foo" b="bar" c=$o@one d=$t@two > out {
  echo "Hello" > ${out}
}
```

What’s New?

Globbing

Sometimes it is useful to have a single task that has access to multiple branches; for example, when evaluating statistical significance of several simulations as compared to a baseline. We can do this with the glob operator in place of a graft.

```
task foo :: param=(Letter: a b c d) > letter {
   echo ${param} > ${letter}
}

task bar < in=$letter@foo[Letter:*] {
   echo ${in}
}
```

The Problem

Running a globbed `.tape` slows significantly as the number of branches increases. Until recently, a glob of 10 branches would seemingly cause ducttape to hang. An analysis of the source code revealed that duplicate hyperedges were being added to the hypergraph for each branch, causing the traversal to visit the same vertex n**n times.

By preventing the hypergraph builder from adding duplicate edges, the globbed `.tape` file runs in a fraction of the time.

bar graphs oooooooooh

Conclusion

Summary

ducttape is a workflow management system that automatically resolves dependencies and realizations of tasks. Features such as branching, grafting, and globbing allow the user to implement a variety of scientific computing tasks compactly.

The ducttape source code can be downloaded and installed from

`https://github.com/jhclark/ducttape`

Questions?

References i