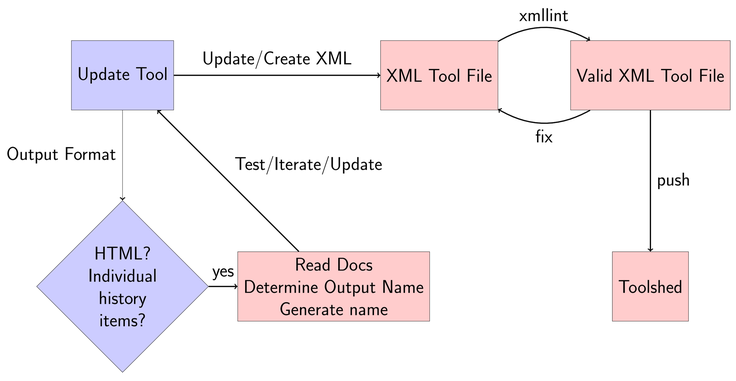
Argument Parsing Libraries for Automatic Galaxy XML Generation

Eric Rasche

Center for Phage Technology, Texas A&M University, College Station, Texas, USA

Current Suboptimal Workflow



Rationale

We want our tools available *both* in Galaxy *and* at the command line–for cluster users. We want to focus on writing useful tools, not spending time updating XML files every time a new option is added.
Furthermore, there are pitfalls and hurdles to deploying tools to Galaxy, not to mention a significant time investment.

Problems Identified

- Galaxy XML Tool files are not DRY! The data and structure behind the XML file is identical to that which defines the command line interface
- Hand crafted XML is error prone and requires validation
- Galaxy is picky about how files should be named
- Command line users should not be second class citizens

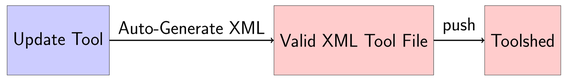
What This Does and Does Not Do

These libraries are capable of a lot, but not everything:

Component	Ability	Implemented
Parameter Types	Integers	✓
	Float	✓
	Bools (Flags)	✓
	Files (Input/Output)	✓
	Strings	✓
	Select/Option/ComboBox	✓
Parameter Options	Multiple Values	✓
	Default Values	✓
	Galaxy/CLI Specificity	✓
	Hidden	✓
	Type-specific validation (e.g., min/-max)	✓
Galaxy Specific	Conditional/Which	X
	`.loc` files	X
	Custom Value Validation Code	X
	`<repeat/>`	✓
	Test Cases Generated	✓
	Documentation extracted from scripts	✓
	100% Valid XML, 100% of the time	✓
CLI Specific	Test Scripts Generated	✓
	GNU GetOpt Parameter Parsing	✓
Output Files	Single API for both Galaxy/CLI	✓
	Handle files destined for new History Items	✓
	Handle files that are part of a single item (e.g, pictures in HTML)	✓

Conditional/Which support is planned for `v2.1` release

Improved Workflow



Solution

- Wrap `GetOpt::Long::Descriptive` in Perl and `argparse` in Python
- Use the developer's specified options to automatically generate the XML code.
- Provide routines to automatically handle naming and processing of output data
- Automatically provide *sane* interfaces in Galaxy and at CLI by using standard `argparse`/`GetOpt` facilities.

Example - Perl

```perl
#!/usr/bin/perl
use strict;
use warnings;
use CPT::GalaxyGetOpt;
my $ggo  = CPT::GalaxyGetOpt->new();
my $options = $ggo->getOptions(
  'options' => [
    [ 'file', 'Input file', { validate => 'File/Input' } ],
    [ 'int', 'An integer', { validate => 'Int', min => 10, deafult => 30}],
    [ 'option', 'Select an option!',{ validate => 'Option', options => { 'a' => 'Alpha', 'b' => 'Bravo' }, multiple => 1 } ],
  ],
  'outputs' => [
    ['test_output','Output Data', { validate => 'File/Output', required => 1, default => 'out', data_format => 'text/plain', default_format => 'TXT'
    }
    ],
  ],
  'defaults' => {
    'appid' => 'org.cpt.examples.GGOPoster', 'appname' => 'Example Utility', 'appdesc' => 'prints out options passed to it', 'appvers' => '1.0.0',
  },
  'tests' => [
    {
      test_name => "Default", params => {}, outputs => { 'test_output' =>
        ["out.txt", 'test-data/outputs/template.default.txt' ],},
    },
  ],
);
=head1 DESCRIPTION

Print out options passed to it

=cut
```

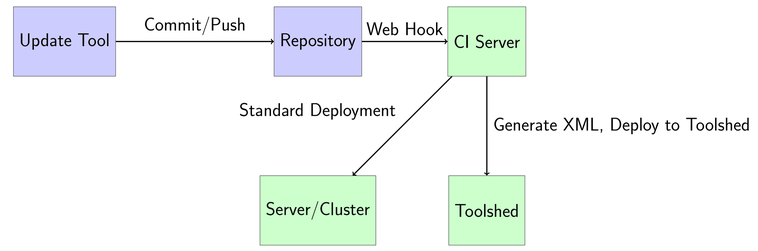
Example - Python

```python
#!/usr/bin/env python
"""
DESCRIPTION
===========

Print out options passed to it
"""
from galaxygetopt.ggo import GalaxyGetOpt as GGO
def main():
    c = GGO(
        options=[
            ['file', 'Input file', {'required': True, 'validate': 'File/Input'}],
            ['int', 'An integer', {'required': True, 'validate': 'Int', 'min': 10, 'default': 30}],
            ['option', 'A selection type parameter', {'validate': 'Option', 'options': {'a': 'Alpha', 'b': 'Bravo'}, 'multiple': True }],
        ],
        outputs=[
            ['test_output', 'Output Data',
                {
                    'validate': 'File/Output',
                    'required': True,
                    'default': 'ggo_out_complex',
                    'data_format': 'text/tabular',
                    'default_format': 'TSV_U',
                }
            ]
        ],
        defaults={
            'appid': 'org.cpt.examples.GGOPoster',
            'appname': 'Example Utility',
            'appdesc': 'prints out options passed to it',
            'appvers': '1.0.0',
        },
        tests=[
            {'test_name': 'Default', 'params': {}, 'outputs': {
                    'test_output': ['ggo_out_complex_Sheet1.tsv',
                'galaxygetopt/tests/test_file.tsv'],}
            }
        ],
        doc=__doc__
    )
```

Example Result XML

```xml
<?xml version="1.0"?>
<tool id="org_cpt_examples_GGOPoster" name="Example Utility" version="1.0.0">
  <description>prints out options passed to it</description>
  <version_command>python example.py --version</version_command>
  <stdio>
    <exit_code level="fatal" range="1:"/>
  </stdio>
  <command interpreter="python">example.py
--galaxy
--outfile_supporting $__new_file_path__
--file "${file}"
--int "${int}"
#for $item in $repeat_option:
--option "${item.option}"
#end for
--test_output "${test_output}"
--test_output_files_path "${test_output_files_path}"
--test_output_format "${test_output_format}"
--test_output_id "test_output_id"
</command>
  <inputs>
    <param help="Input file" label="file" name="file" optional="False" type="data"/>
    <param help="An integer" label="int" min="10" name="int" optional="False" type="integer" value="30"/>
    <repeat name="repeat_option" title="Option">
      <param help="A selection type parameter" label="option" name="option" optional="True" type="select">
        <option value="a">Alpha</option>
        <option value="b">Bravo</option>
      </param>
    </repeat>
    <param help="Output Data" label="Format of test_output" name="test_output_format" optional="False" type="select">
      <option value="CSV">CSV</option>
      <option value="CSV_U">CSV_U</option>
      <option value="TSV">TSV</option>
      <option selected="True" value="TSV_U">TSV_U</option>
    </param>
  </inputs>
  <outputs>
    <data format="TSV_U" name="test_output">
      <change_format>
        <when format="tabular" input="test_output_format" value="CSV" />
        <when format="tabular" input="test_output_format" value="CSV_U" />
        <when format="tabular" input="test_output_format" value="TSV" />
        <when format="tabular" input="test_output_format" value="TSV_U" />
      </change_format>
    </data>
  </outputs>
  <help>DESCRIPTION
===========

Print out options passed to it
</help>
  <tests>
    <test>
      <output file="galaxygetopt/tests/test_file.tsv" name="test_output" />
    </test>
  </tests>
</tool>
```

Ideal, CI Server Based Workflow



Great for Automation, Peace of Mind

- Focus on writing tools, not on galaxy integration
- Support a larger user base (CLI+Galaxy)
- CI Servers like Jenkins can generate the XML automatically
- No more issues of XML matching up with the latest tool version
- No more time spent validating XML

How to Start Using It

- `CPT::GalaxyGetOpt` in perl
- `galaxygetopt` in python

When you need to generate galaxy XML files:

```
python script.py --generate_galaxy_xml > script.xml
  perl script.pl --generate_galaxy_xml > script.xml
```

When you want to generate test scripts for your tools:

```
python script.py --gen_tests > test_script.py
  perl script.pl --gen_tests > script.t
```

These tests are based on information you store in your tool's GalaxyGetOpt call. By specifying the inputs and outputs of the tool, this library can construct both the Galaxy XML test cases and generalized command line test cases

Get the code!

Perl version will require you to check out the `build/develop` branch, unless you have DistZilla installed.
`https://cpt.tamu.edu/gitlab/cpt/libcpt`
Python version
`pip install galaxygetopt`

Future Work

These libraries receive heavy use internally. If they spark that sort of interest within the galaxy community, we're considering looking into R and C/C++ ports of the code.

Galaxy and the CPT

At the Center for Phage Technology, we face a large number of problems not faced by the eukaryotic community; hundreds of small genomes, lack of existing toolsets, and comparatively transient datasets. In response to these challenges and others, to date we have added over 100 brand new, phage specific, tools to our Galaxy instance.

In the past year the CPT has migrated from a bespoke bioinformatics platform to a 100% Galaxy organization with extremely tangible benefits, especially in our novel phage annotation course. Bacteriophage Genomics at the CPT, taught by Dr. Ryland Young, has been able to successfully utilise Galaxy to significantly accelerate our analysis due to the increased ability to produce and synthesize results within Galaxy. Use of workflows has allowed for reallocation of teaching time to actual teaching and improvement of genome annotations, away from time wasted shepherding students through the command line.