# ROBOT Tutorial

James Overton & Becky Tauber

## Overview

- Manual editing
- Repetitive tasks
- Automated workflows
- Quality assurance
- Modular development
- Modular releases
- Other commands

## Manual Editing

### Status Quo

- development mainly done in Protégé
- requires manual editing and review

### The Solution

- ROBOT lets us automate many of those tasks
- ... and provides better quality assurance!

### ROBOT Releases

- v1.0.0 released Feb 8, 2018 ...
- v1.4.0 released Mar 15, 2019
- v1.4.1 released Jun 27, 2019

## Cite ROBOT

R.C. Jackson, J.P. Balhoff, E. Douglass, N.L. Harris, C.J. Mungall, and J.A. Overton. **ROBOT: A tool for automating ontology workflows**. BMC Bioinformatics, vol. 20, July 2019.

## Let's get started!

- download the **tutorial repository (click)**
- get the latest ROBOT via Docker:
  - `sh run.sh robot --version`
- or install it following **the documentation (click)**
- navigate to examples: `cd examples`
  - if using Docker, start each command with `sh ../run.sh`

# ROBOT for Repetitive Tasks

Click on the command name for documentation!

- **merge**
- **reason**
- **annotate**
- **convert**

## Merge

1. merge two separate ontologies
2. merge imports into one ontology

### Merge Separate Ontologies

```
robot merge --input edit.owl \
  --input foo.owl \
  --output results/merged.owl
```

### Merge Imports

ROBOT will automatically merge imports. To *not* merge imports, include `--collapse-import-closure false`. This option is not supported in v1.0.0.

```
robot merge --input with-import.owl \
  --output results/merged_imports.owl
```

## Reason

1. logical validation
2. automatic classification

### Reasoners

- ELK: a very fast reasoner, but not as powerful as HermiT
- HermiT: optimized to classify complex ontologies
- EMR: materializes anonymous expressions
- Structural: a simple reasoner

### Logical Validation

Validates that the ontology contains no unsatisfiable entities and that it is not inconsistent.

```
robot reason --input unsatisfiable.owl
```

### Automatic Classification

```
robot reason --input non-reasoned.owl \
  --output results/reasoned.owl
```

## Annotate

1. add metadata to an ontology for release

### Add Metadata

```
robot annotate --input edit.owl \
  --version-iri \
    https://github.com/ontodev/robot/releases/2019-07-31/edit.owl \
  --annotation oboInOwl:date "07:31:2019 12:00" \
  --output results/annotated.owl
```

## Convert

1. convert an editor's file to verbose RDF/XML
2. convert a released ontology to OBO

#### Ontology Formats

- default format is RDF/XML but...
- ontologies are shared in many formats
  - OWL functional
  - OBO
  - Turtle
  - and more...

#### Convert the Edit File

```
robot convert --input edit.owl \
  --format owl \
  --output results/release.owl
```

**NOTE**: OWL functional syntax is defined by the suffix `.ofn` - if you want to convert to that format, use `--format ofn` or an output ending in `.ofn`. The `.owl` suffix can be used to represent any OWL ontology. Using `--format owl` or excluding the `--format` option and specifying an output with `.owl` will convert to RDF/XML.

#### Convert to OBO

```
robot convert --input edit.owl \
  --output results/release.obo
```

**NOTE**: You do not always need to include the `--format` if the extension of the `--output` matches the desired format.

# Automated Workflows

## Chaining Commands

Output ontologies can be used as the input to subsequent commands. Only the first command uses an `--input`, and only the last command uses an `--output`.

```
robot merge --input edit.owl --collapse-import-closure true \
  reason --reasoner ELK \
  annotate --version-iri http://purl.obolibrary.org/obo/robot/2018-08-07/release.owl \
  convert --output results/chained_release.ttl
```

### Makefiles

A Makefile contains a set of rules to make target objects.Here, we use it to create the release files.

`make release`

**NOTE**: This Makefile will fail on v1.0.0 and v1.1.0 - alternatively, run `make build`.

## Quality Assurance

Click on the command name for documentation!

- **diff**
- **query (part 1)**
- **verify**
- **report**

### Diff

1. compare the axioms in two version of an ontology
2. generate a summary page of changes

`diff` supports different formats:

- `plain`: default, shows removed and added axioms
- `pretty`: plain, plus entity labels for IRIs
- `html`: HTML summary page (easy for humans!)
- `markdown`: markdown summary (great for GitHub!)

#### Compare Axioms (Default)

```
robot diff --left non-reasoned.owl \
  --right results/reasoned.owl \
  --output results/diff.txt
```

**NOTE**: If you do not include an `--output`, the results will be printed to the terminal.

#### Human-Readable Diff

```
robot diff --left non-reasoned.owl \
  --right results/reasoned.owl \
  --format html --output results/diff.html
```

## Query

1. run a SPARQL SELECT query
2. run a SPARQL ASK query

### Select Query

SELECT queries can be useful for statistics on the ontology, and for sharing data. For example, you could get a list of all the terms containing just the IDs and labels.

```
robot query --input edit.owl \
  --query select.rq results/select.tsv
```

### Ask Query

ASK queries can help determine if something exists in the ontology or not. For example, if you want to make sure that "trunk" is in the "vertebrate core" subset.

```
robot query --input edit.owl \
  --query ask.rq results/ask.txt
```

## Verify

1. perform a successful verification
2. perform an unsuccessful verification

### Successful Verification

Sometimes, classes can be accidentally orphaned. This verification ensures that all classes, aside from the top-level "anatomical entity", have asserted parents. If they do not, the verification fails.

```
robot verify --input edit.owl \
  --queries verify.rq \
  --output-dir results
```

### Unsuccessful Verification

This verify query checks that all classes have an equivalent class statement. In our edit ontology, only some of the classes have equivalent classes, so this will fail.

```
robot verify --input edit.owl \
  --queries verify_fail.rq \
  --output-dir results
```

### Report

1. run a series of SPARQL queries to find common violations

#### Violations

- annotation, logical, or metadata violations
- three logging levels: ERROR, WARN, and INFO

#### Generate a Report

```
robot report --input edit.owl --output results/report.tsv
```

### Continuous Integration

- Travis CI automatically builds and tests changes pushed to GitHub
- Often uses `make test` from the Makefile
- If any part of the test fails, the build will fail

## Modular Development

Click on the command name for documentation!

- **extract**
- **template**

### Extract

1. create an import module with SLME
2. create an import module with MIREOT

#### Import Modules

- many bioontologies use terms from external sources
- these sources contain more terms than needed

- extract ensures the necessary terms and their dependencies are included in module

#### Extract with SLME

STAR: fixpoint-nested BOT: bottom module TOP: top module

```
robot extract \
  --input-iri http://purl.obolibrary.org/obo/obi.owl \
  --term OBI:0000443 \
  --method BOT \
  --output results/obi_bot.owl
```

**NOTE**: You can also include a list of terms to extract in a text file with `--term-file`. **NOTE 2**: To use a local file, use `--input <file>` instead of `--input-iri`.

#### Extract with MIREOT

Creates a simple hierarchy of terms. Requires lower term(s) and optional upper term(s).

```
robot extract --input-iri http://purl.obolibrary.org/obo/obi.owl \
  --method MIREOT --lower-terms obi_terms.txt \
  --output results/obi_mireot.owl
```

**NOTE**: Without specifiying any `--upper-terms`, the MIREOT method will include all ancestors up to `owl:Thing`. **NOTE 2**: To just specify *one* term to extract, use `--lower-term`

### Template

1. create a module
2. add a class to an ontology

#### Create a Module

This will create a standalone module that can be included in the edit file with an import statement. To update the module, editors only need to update the spreadsheet and run this command to remake the module.

```
robot template --input edit.owl --template module.tsv \
  --ontology-iri http://purl.obolibrary.org/obo/robot/module.owl \
  --output results/module.owl
```

**NOTE**: `template` gets all the entity labels from `edit.owl` so we are able to use the labels in the spreadsheet, instead of always specifying the ID. If we didn't include the `--input`, the labels would not resolve.

#### Add a Class to an Ontology

For one-time class creation (especially if many classes need to be created), a temporary template can be created and the results immediately merged into the edit ontology.

```
robot template --input edit.owl --merge-before \
  --template new_class.tsv \
  --output results/new_class.owl
```

**NOTE**: if your ontology includes imports, use `--collapse-import-closure false` with any merge option to maintain the closure.

## Modular Releases

Click on the command name for documentation!

- **query (part 2)**
- **remove**
- **filter**

### Querying with SPARQL CONSTRUCT

- CONSTRUCT produces RDF data in Turtle format
- allows creation of new sets of triples

#### Update Annotations with CONSTRUCT

```
robot query --input edit.owl \
  --query construct.rq results/construct.ttl \
  merge --input results/construct.ttl \
  --output results/construct.owl
```

The `construct.ttl` file isn't much use on its own; we need to merge it. For `query`, the output ontology is the *unchanged* input ontology, so we can chain this with the `merge` command to merge `construct.ttl`.

## Querying with SPARQL UPDATE

- UPDATE changes the RDF data and outputs an updated ontology
- allows deletion and insertion of triples
- no need to merge after running the UPDATE

### Update Annotations with UPDATE

```
robot query --input edit.owl \
  --update update.ru \
  --output results/updated.owl
```

Here, we replace 'definition' (`IAO:0000115`) with 'external_definition' (`UBPROP:0000001`) for all UBERON terms.

## Remove

1. remove a class and its descendants
2. create a 'simple' version of an ontology

### Configuration

- remove a term (or terms) from an ontology and any related terms
- highly configurable `--select` option to remove related terms
- specify types of axioms to remove with `--axioms`

### Remove a Class + Descendants

```
robot remove --input edit.owl \
  --term UBERON:0000475 \
  --select "self descendants" \
  --output results/removed.owl
```

### Create a 'Simple' Version

```
robot remove --input edit.owl --axioms equivalent \
  remove --select parents --select anonymous --select imports \
  --output results/simple.owl
```

**NOTE**: `--select` accepts a string of options, or can be passed multiple times. For a string of options (previous example) the selected set is the union of all options (both the input term and all descendants of the term). For passing in

multiple select options (as above), the options are processed in order (first the parents are selected, and then only the anonymous parents are selected).

### Filter

Released with v1.2.0-alpha - previously, `filter` only filtered for object properties.

1. extract a branch of an ontology
2. create a subset based on annotations

#### Extract a Branch

```
robot filter --input edit.owl \
  --term UBERON:0000475 \
  --select "self descendants annotations" \
  --output results/branch.owl
```

**NOTE**: in order to include annotations on the filtered entities, `--select annotations` must be included. Otherwise, you muist include *all* annotation properties in the set of input terms.

#### Create a Subset

```
robot filter --input edit.owl \
  --select \
  "oboInOwl:inSubset=<http://purl.obolibrary.org/obo/uberon/core#uberon_slim>" \
  --select annotations \
  --output-iri http://purl.obolibrary.org/robot/uberon_slim.owl
  --output results/uberon_slim.owl
```

**NOTE**: selecting for annotations is highly configurable: `CURIE=CURIE CURIE="literal"^^datatype CURIE=<IRI> CURIE=~"regex pattern"`

## Other Commands

Click on a command name to navigate to the documentation!

- **explain**
- **materialize**
- **mirror**
- **reduce**
- **relax**
- **rename**
- **repair**

- **unmerge**
- **validate**