GSoC RDKit-MolVS Integration

Susan Leung, University of Oxford PhD

7th RDKit UGM, Cambridge

19th September 2018

https://github.com/susanhleung/rdkit

UNIVERSITY OF OXFORD

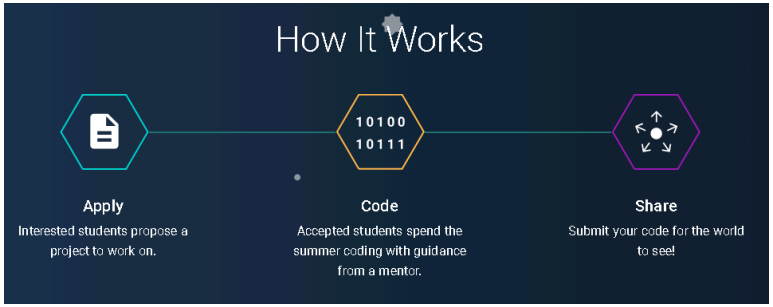
Google Summer of Code

Outline

- GSoC
- The Project: RDKit-MolVS Integration
  - Jupyter-notebook demo
  - RDKit directory structure
  - Additional Features
- Things I've learnt
- Conclusions

GSoC

Google
Summer of Code



- Google Summer of Code
- Global program that aims to introduces students to open source software development.
- 3 months to work on project with an open source organisation.
- Each student is paired with a mentor from the participating organisation.
- Open Chemistry project (started 2016)

Open Chemistry

- Avogadro
- cclib
- DeepChem
- 3Dmol.js
- MSDK
- NWChem
- Open Babel
- RDKit

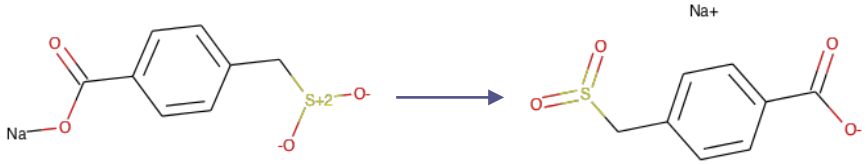
The Project: RDKit-MolVS Integration

- MolVS
  - Molecular standardization and validation tool
  - Written in Python, built on RDKit framework
  - Open source
    - https://molvs.readthedocs.io/en/latest/
    - https://github.com/mcs07/MolVS

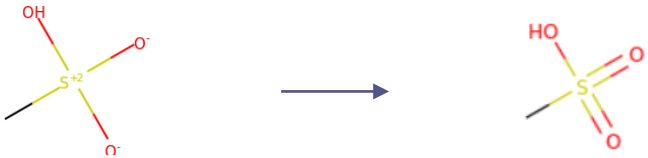
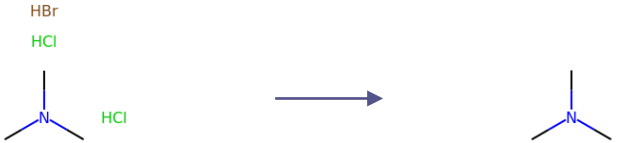
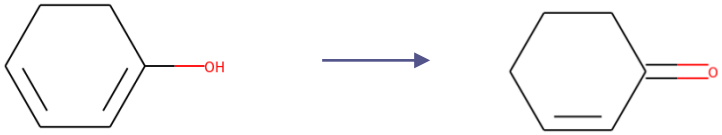
- Aim
  - Integrate MolVS into the RDKit C++ source.
  - Additionally: expand the current capabilities of MolVS.

Matt Swain
MolVS author

MolVS Modules

	Example
Standardization	

Validation	>>> from molvs import validate_smiles
	>>> validate_smiles('O=C([O-])c1ccccc1')
	['INFO: [NeutralValidation] Not an overall neutral system (-1)']

Charges	
Fragments	
Tautomers	

Demo with jupyter-notebook

- Based on IPython notebook found at $RDBASE/rdkit/Chem/MolStandardize/tutorial/MolStandardize.ipynb

What is looks like in the RDKit directories …

$RDBASE/Code/GraphMol/MolStandardize

```
$RDBASE/Code/GraphMol/MolStandardize
├── CMakeLists.txt
├── Charge.cpp
├── Charge.h
├── Fragment.cpp
├── Fragment.h
├── Metal.cpp
├── Metal.h
├── MolStandardize.cpp
├── MolStandardize.h
├── Normalize.cpp
├── Normalize.h
├── Tautomer.cpp
├── Tautomer.h
├── Validate.cpp
├── Validate.h
├── test1.cpp
├── test2.cpp
├── testCharge.cpp
├── testFragment.cpp
├── testNormalize.cpp
├── testPCS.cpp
├── testTautomer.cpp
├── testValidate.cpp
├── FragmentCatalog
│   ├── FragmentCatalogEntry.cpp
│   ├── FragmentCatalogEntry.h
│   ├── FragmentCatalogParams.cpp
│   ├── FragmentCatalogParams.h
│   ├── FragmentCatalogUtils.cpp
│   └── FragmentCatalogUtils.h
├── AcidBaseCatalog
│   ├── AcidBaseCatalogEntry.cpp
│   ├── AcidBaseCatalogEntry.h
│   ├── AcidBaseCatalogParams.cpp
│   ├── AcidBaseCatalogParams.h
│   ├── AcidBaseCatalogUtils.cpp
│   └── AcidBaseCatalogUtils.h
├── TautomerCatalog
│   ├── TautomerCatalogEntry.cpp
│   ├── TautomerCatalogEntry.h
│   ├── TautomerCatalogParams.cpp
│   ├── TautomerCatalogParams.h
│   ├── TautomerCatalogUtils.cpp
│   └── TautomerCatalogUtils.h
├── TransformCatalog
│   ├── TransformCatalogEntry.cpp
│   ├── TransformCatalogEntry.h
│   ├── TransformCatalogParams.cpp
│   ├── TransformCatalogParams.h
│   ├── TransformCatalogUtils.cpp
│   └── TransformCatalogUtils.h
└── Wrap
    ├── Charge.cpp
    ├── CMakeLists.txt
    ├── Fragment.cpp
    ├── Metal.cpp
    ├── Normalize.cpp
    ├── rdMolStandardize.cpp
    ├── testMolStandardize.py
    └── Validate.cpp

5 directories, 55 files
```

$RDBASE/Code/GraphMol/MolStandardize

```
$RDBASE/Code/GraphMol/MolStandardize
├── CMakeLists.txt
├── Charge.cpp
├── Charge.h
├── Fragment.cpp
├── Fragment.h
├── Metal.cpp
├── Metal.h
├── MolStandardize.cpp
├── MolStandardize.h
├── Normalize.cpp
├── Normalize.h
├── Tautomer.cpp
├── Tautomer.h
├── Validate.cpp
├── Validate.h
├── test1.cpp
├── test2.cpp
├── testCharge.cpp
├── testFragment.cpp
├── testNormalize.cpp
├── testPCS.cpp
├── testTautomer.cpp
├── testValidate.cpp
├── FragmentCatalog
│   ├── FragmentCatalogEntry.cpp
│   ├── FragmentCatalogEntry.h
│   ├── FragmentCatalogParams.cpp
│   ├── FragmentCatalogParams.h
│   ├── FragmentCatalogUtils.cpp
│   └── FragmentCatalogUtils.h
├── AcidBaseCatalog
│   ├── AcidBaseCatalogEntry.cpp
│   ├── AcidBaseCatalogEntry.h
│   ├── AcidBaseCatalogParams.cpp
│   ├── AcidBaseCatalogParams.h
│   ├── AcidBaseCatalogUtils.cpp
│   └── AcidBaseCatalogUtils.h
├── TautomerCatalog
│   ├── TautomerCatalogEntry.cpp
│   ├── TautomerCatalogEntry.h
│   ├── TautomerCatalogParams.cpp
│   ├── TautomerCatalogParams.h
│   ├── TautomerCatalogUtils.cpp
│   └── TautomerCatalogUtils.h
├── TransformCatalog
│   ├── TransformCatalogEntry.cpp
│   ├── TransformCatalogEntry.h
│   ├── TransformCatalogParams.cpp
│   ├── TransformCatalogParams.h
│   ├── TransformCatalogUtils.cpp
│   └── TransformCatalogUtils.h
└── Wrap
    ├── Charge.cpp
    ├── CMakeLists.txt
    ├── Fragment.cpp
    ├── Metal.cpp
    ├── Normalize.cpp
    ├── rdMolStandardize.cpp
    ├── testMolStandardize.py
    └── Validate.cpp

5 directories, 55 files
```

Code structure and naming is similar to MolVS modules

$RDBASE/Code/GraphMol/MolStandardize

```
$RDBASE/Code/GraphMol/MolStandardize
├── CMakeLists.txt
├── Charge.cpp
├── Charge.h
├── Fragment.cpp
├── Fragment.h
├── Metal.cpp
├── Metal.h
├── MolStandardize.cpp
├── MolStandardize.h
├── Normalize.cpp
├── Normalize.h
├── Tautomer.cpp
├── Tautomer.h
├── Validate.cpp
├── Validate.h
├── test1.cpp
├── test2.cpp
├── testCharge.cpp
├── testFragment.cpp
├── testNormalize.cpp
├── testPCS.cpp
├── testTautomer.cpp
├── testValidate.cpp
├── FragmentCatalog
│   ├── FragmentCatalogEntry.cpp
│   ├── FragmentCatalogEntry.h
│   ├── FragmentCatalogParams.cpp
│   ├── FragmentCatalogParams.h
│   ├── FragmentCatalogUtils.cpp
│   └── FragmentCatalogUtils.h
├── AcidBaseCatalog
│   ├── AcidBaseCatalogEntry.cpp
│   ├── AcidBaseCatalogEntry.h
│   ├── AcidBaseCatalogParams.cpp
│   ├── AcidBaseCatalogParams.h
│   ├── AcidBaseCatalogUtils.cpp
│   └── AcidBaseCatalogUtils.h
├── TautomerCatalog
│   ├── TautomerCatalogEntry.cpp
│   ├── TautomerCatalogEntry.h
│   ├── TautomerCatalogParams.cpp
│   ├── TautomerCatalogParams.h
│   ├── TautomerCatalogUtils.cpp
│   └── TautomerCatalogUtils.h
├── TransformCatalog
│   ├── TransformCatalogEntry.cpp
│   ├── TransformCatalogEntry.h
│   ├── TransformCatalogParams.cpp
│   ├── TransformCatalogParams.h
│   ├── TransformCatalogUtils.cpp
│   └── TransformCatalogUtils.h
└── Wrap
    ├── Charge.cpp
    ├── CMakeLists.txt
    ├── Fragment.cpp
    ├── Metal.cpp
    ├── Normalize.cpp
    ├── rdMolStandardize.cpp
    ├── testMolStandardize.py
    └── Validate.cpp

5 directories, 55 files
```

```
$MolVS/molvs/
├── charge.py
├── cli.py
├── errors.py
├── fragment.py
├── __init__.py
├── metal.py
├── normalize.py
├── resonance.py
├── standardize.py
├── tautomer.py
├── utils.py
├── validate.py
└── validations.py
```

Code structure and naming is similar to MolVS modules

$RDBASE/Code/GraphMol/MolStandardize

```
$RDBASE/Code/GraphMol/MolStandardize
├── CMakeLists.txt
├── Charge.cpp
├── Charge.h
├── Fragment.cpp
├── Fragment.h
├── Metal.cpp
├── Metal.h
├── MolStandardize.cpp
├── MolStandardize.h
├── Normalize.cpp
├── Normalize.h
├── Tautomer.cpp
├── Tautomer.h
├── Validate.cpp
├── Validate.h
├── test1.cpp
├── test2.cpp
├── testCharge.cpp
├── testFragment.cpp
├── testNormalize.cpp
├── testPCS.cpp
├── testTautomer.cpp
├── testValidate.cpp
├── FragmentCatalog
│   ├── FragmentCatalogEntry.cpp
│   ├── FragmentCatalogEntry.h
│   ├── FragmentCatalogParams.cpp
│   ├── FragmentCatalogParams.h
│   ├── FragmentCatalogUtils.cpp
│   └── FragmentCatalogUtils.h
├── AcidBaseCatalog
│   ├── AcidBaseCatalogEntry.cpp
│   ├── AcidBaseCatalogEntry.h
│   ├── AcidBaseCatalogParams.cpp
│   ├── AcidBaseCatalogParams.h
│   ├── AcidBaseCatalogUtils.cpp
│   └── AcidBaseCatalogUtils.h
├── TautomerCatalog
│   ├── TautomerCatalogEntry.cpp
│   ├── TautomerCatalogEntry.h
│   ├── TautomerCatalogParams.cpp
│   ├── TautomerCatalogParams.h
│   ├── TautomerCatalogUtils.cpp
│   └── TautomerCatalogUtils.h
├── TransformCatalog
│   ├── TransformCatalogEntry.cpp
│   ├── TransformCatalogEntry.h
│   ├── TransformCatalogParams.cpp
│   ├── TransformCatalogParams.h
│   ├── TransformCatalogUtils.cpp
│   └── TransformCatalogUtils.h
└── Wrap
    ├── Charge.cpp
    ├── CMakeLists.txt
    ├── Fragment.cpp
    ├── Metal.cpp
    ├── Normalize.cpp
    ├── rdMolStandardize.cpp
    ├── testMolStandardize.py
    └── Validate.cpp

5 directories, 55 files
```

Unit tests

$RDBASE/Code/GraphMol/MolStandardize

```
$RDBASE/Code/GraphMol/MolStandardize
├── CMakeLists.txt
├── Charge.cpp
├── Charge.h
├── Fragment.cpp
├── Fragment.h
├── Metal.cpp
├── Metal.h
├── MolStandardize.cpp
├── MolStandardize.h
├── Normalize.cpp
├── Normalize.h
├── Tautomer.cpp
├── Tautomer.h
├── Validate.cpp
├── Validate.h
├── test1.cpp
├── test2.cpp
├── testCharge.cpp
├── testFragment.cpp
├── testNormalize.cpp
├── testPCS.cpp
├── testTautomer.cpp
├── testValidate.cpp
├── FragmentCatalog
│   ├── FragmentCatalogEntry.cpp
│   ├── FragmentCatalogEntry.h
│   ├── FragmentCatalogParams.cpp
│   ├── FragmentCatalogParams.h
│   ├── FragmentCatalogUtils.cpp
│   └── FragmentCatalogUtils.h
├── AcidBaseCatalog
│   ├── AcidBaseCatalogEntry.cpp
│   ├── AcidBaseCatalogEntry.h
│   ├── AcidBaseCatalogParams.cpp
│   ├── AcidBaseCatalogParams.h
│   ├── AcidBaseCatalogUtils.cpp
│   └── AcidBaseCatalogUtils.h
├── TautomerCatalog
│   ├── TautomerCatalogEntry.cpp
│   ├── TautomerCatalogEntry.h
│   ├── TautomerCatalogParams.cpp
│   ├── TautomerCatalogParams.h
│   ├── TautomerCatalogUtils.cpp
│   └── TautomerCatalogUtils.h
├── TransformCatalog
│   ├── TransformCatalogEntry.cpp
│   ├── TransformCatalogEntry.h
│   ├── TransformCatalogParams.cpp
│   ├── TransformCatalogParams.h
│   ├── TransformCatalogUtils.cpp
│   └── TransformCatalogUtils.h
└── Wrap
    ├── Charge.cpp
    ├── CMakeLists.txt
    ├── Fragment.cpp
    ├── Metal.cpp
    ├── Normalize.cpp
    ├── rdMolStandardize.cpp
    ├── testMolStandardize.py
    └── Validate.cpp

5 directories, 55 files
```

Catalogs allow standardizations to be read from an input file
(allows user to specify/customise their own)

$RDBASE/Code/GraphMol/MolStandardize

```
$RDBASE/Code/GraphMol/MolStandardize
├── CMakeLists.txt
├── Charge.cpp
├── Charge.h
├── Fragment.cpp
├── Fragment.h
├── Metal.cpp
├── Metal.h
├── MolStandardize.cpp
├── MolStandardize.h
├── Normalize.cpp
├── Normalize.h
├── Tautomer.cpp
├── Tautomer.h
├── Validate.cpp
├── Validate.h
├── test1.cpp
├── test2.cpp
├── testCharge.cpp
├── testFragment.cpp
├── testNormalize.cpp
├── testPCS.cpp
├── testTautomer.cpp
├── testValidate.cpp
├── FragmentCatalog
│   ├── FragmentCatalogEntry.cpp
│   ├── FragmentCatalogEntry.h
│   ├── FragmentCatalogParams.cpp
│   ├── FragmentCatalogParams.h
│   ├── FragmentCatalogUtils.cpp
│   └── FragmentCatalogUtils.h
├── AcidBaseCatalog
│   ├── AcidBaseCatalogEntry.cpp
│   ├── AcidBaseCatalogEntry.h
│   ├── AcidBaseCatalogParams.cpp
│   ├── AcidBaseCatalogParams.h
│   ├── AcidBaseCatalogUtils.cpp
│   └── AcidBaseCatalogUtils.h
├── TautomerCatalog
│   ├── TautomerCatalogEntry.cpp
│   ├── TautomerCatalogEntry.h
│   ├── TautomerCatalogParams.cpp
│   ├── TautomerCatalogParams.h
│   ├── TautomerCatalogUtils.cpp
│   └── TautomerCatalogUtils.h
├── TransformCatalog
│   ├── TransformCatalogEntry.cpp
│   ├── TransformCatalogEntry.h
│   ├── TransformCatalogParams.cpp
│   ├── TransformCatalogParams.h
│   ├── TransformCatalogUtils.cpp
│   └── TransformCatalogUtils.h
└── Wrap
    ├── Charge.cpp
    ├── CMakeLists.txt
    ├── Fragment.cpp
    ├── Metal.cpp
    ├── Normalize.cpp
    ├── rdMolStandardize.cpp
    ├── testMolStandardize.py
    └── Validate.cpp
```

5 directories, 55 files

Python wrap

Now to delve into some of the code…

Additional features

- Validation
  - 4 modes
    - RDKitValidation (detects incorrect atom valency)
    - MolVSValidation (same as current MolVS)
    - AllowedAtoms (throws up message if atom in not in given allowed atoms list)
    - DisallowedAtoms (throws up message if atom is in given disallowed atoms list)
- Client can define their own customised files for:
  - Fragments
  - Normalization
  - Charge

Additional Features: Validation

$RDBASE/Code/GraphMol/MolStandardize/Validate.h

```
class ValidationMethod {
 public:
  virtual std::vector<ValidationErrorInfo> validate(
      const ROMol &mol, bool reportAllFailures) const = 0;
};

class RDKitValidation : public ValidationMethod {
 public:
  std::vector<ValidationErrorInfo> validate(
      const ROMol &mol, bool reportAllFailures) const override;
};

[...]

class MolVSValidation : public ValidationMethod {
 public:
  [...]
  std::vector<ValidationErrorInfo> validate(
      const ROMol &mol, bool reportAllFailures) const override;
 private:
 [...]
};
```

4 validation modes

The **RDKitValidation** class detects incorrect valencies

The **MolVSValidation** class has same functionality as MolVS Validation

Additional Feature

```
class AllowedAtomsValidation : public ValidationMethod {
 public:
  [...]
  std::vector<ValidationErrorInfo> validate(
      const ROMol &mol, bool reportAllFailures) const override;
 private:
  [...]
};

class DisallowedAtomsValidation : public ValidationMethod {
 public:
  [...]
  std::vector<ValidationErrorInfo> validate(
      const ROMol &mol, bool reportAllFailures) const override;
 private:
  [...]
};
```

The **AllowedAtomsValidation** class lets the user input a list of atoms, anything not on the list throws an error.

The **DisallowedAtomsValidation** class lets the user input a list of atoms and as long as there are no atoms from the list it is deemed acceptable.

Additional Features: Validation

$RDBASE/Code/GraphMol/MolStandardize/Validate.h

```
class ValidationMethod {
 public:
  virtual std::vector<ValidationErrorInfo> validate(
      const ROMol &mol, bool reportAllFailures) const = 0;
};

class RDKitValidation : public ValidationMethod {
 public:
  std::vector<ValidationErrorInfo> validate(
      const ROMol &mol, bool reportAllFailures) const override;
};

[...]

class MolVSValidation : public ValidationMethod {
 public:
  [...]
  std::vector<ValidationErrorInfo> validate(
      const ROMol &mol, bool reportAllFailures) const override;
 private:
  [...]
};

class AllowedAtomsValidation : public ValidationMethod {
 public:
  [...]
  std::vector<ValidationErrorInfo> validate(
      const ROMol &mol, bool reportAllFailures) const override;
 private:
  [...]
};

class DisallowedAtomsValidation : public ValidationMethod {
 public:
  [...]
  std::vector<ValidationErrorInfo> validate(
      const ROMol &mol, bool reportAllFailures) const override;
 private:
  [...]
};
```

4 validation modes

The **RDKitValidation** class detects incorrect valencies

```
std::vector<unsigned int> atoms = {6, 7, 8};
 std::vector<shared_ptr<Atom>> atomList;

 for (auto &atom : atoms) {
   shared_ptr<Atom> a(new Atom(atom));
   atomList.push_back(a);
 }

 AllowedAtomsValidation vm(atomList);
 std::string smi1;

 smi1 = "CC(=O)CF";
 unique_ptr<ROMol> m1(SmilesToMol(smi1));
 vector<ValidationErrorInfo> errout1 = vm.validate(*m1,
true);
```

Additional Features: specifying/customising own standardization files

$RDBASE/Code/GraphMol/MolStandardize/MolStandardize.h

```
struct CleanupParameters {
  std::string rdbase = std::getenv("RDBASE");
  std::string normalizations;
  std::string acidbaseFile;
  std::string fragmentFile;
  std::string tautomerTransforms;
  int maxRestarts;     // The maximum number of times to attempt to apply the
                       // series of normalizations (default 200).
  int maxTautomers;    // The maximum number of tautomers to enumerate (default
                       // 1000).
  bool preferOrganic;  // Whether to prioritize organic fragments when choosing
                       // fragment parent (default False).

  CleanupParameters()
      :
      normalizations(rdbase + "/Data/MolStandardize/normalizations.txt"),
      acidbaseFile(rdbase + "/Data/MolStandardize/acid_base_pairs.txt"),
      fragmentFile(rdbase + "/Data/MolStandardize/fragmentPatterns.txt"),
      tautomerTransforms(rdbase +
                         "/Data/MolStandardize/tautomerTransforms.in"),
      maxRestarts(200),
      maxTautomers(1000),
      preferOrganic(false) {}
};
```

The data structure **CleanupParameters** lets you customize your standardization…

Default standardization files.
You can set these to your own.

Additional Features: specifying/customising own standardization files

$RDBASE/Code/GraphMol/MolStandardize/MolStandardize.h

```
struct CleanupParameters {
  std::string rdbase = std::getenv("RDBASE");
  std::string normalizations;
  std::string acidbaseFile;
  std::string fragmentFile;
  std::string tautomerTransforms;
  int maxRestarts;     // The maximum number of times to attempt to apply the
                       // series of normalizations (default 200).
  int maxTautomers;    // The maximum number of tautomers to enumerate (default
                       // 1000).
  bool preferOrganic;  // Whether to prioritize organic fragments when choosing
                       // fragment parent (default False).

  CleanupParameters()
      :
        normalizations(rdbase + "/Data/MolStandardize/normalizations.txt"),
        acidbaseFile(rdbase + "/Data/MolStandardize/acid_base_pairs.txt"),
        fragmentFile(rdbase + "/Data/MolStandardize/fragmentPatterns.txt"),
        tautomerTransforms(rdbase +
                           "/Data/MolStandardize/tautomerTransforms.in"),
        maxRestarts(200),
        maxTautomers(1000),
        preferOrganic(false) {}
};
```

The data structure **CleanupParameters** lets you customize your standardization…

```
MolStandardize::CleanupParameters params;

  std::string rdbase = getenv("RDBASE");
  std::string transformFile =
      rdbase + "/Data/MolStandardize/normalizations.txt";
  params.normalizations = transformFile;

  // Normalize nitro group.
  smi1 = "C1(=CC=CC=C1)[N+](=O)[O-]";
  unique_ptr<RWMol> m1(SmilesToMol(smi1));
  unique_ptr<RWMol> res1(MolStandardize::cleanup(*m1,
params));
```

Things I learnt as a new contributor (apologies to the old hands)

How I got started…

Good blog post:
https://www.blopig.com/blog/2013/02/how-to-install-rdkit-on-ubuntu-12-04/

```
> Git clone https://github.com/rdkit/rdkit.git
> export RDBASE=/opt/RDKit_20XX_XX_X
> export LD_LIBRARY_PATH=$RDBASE/lib:$LD_LIBRARY_PATH
> export PYTHONPATH=$RDBASE:$PYTHONPATH
> cd $RDBASE
> mkdir build
> cd build
> cmake ..
> make
> make install
> ctest
```

Run this every time I made a change to source code

Also run this when I do something with wrapping

To run a single test:
> ctest –V –R molStandardizeTest

New tools and tricks (for me)

- Valgrind
  - Detects memory management and threading bugs, and profiles your programs (mainly used to detect incorrect pointer usage for me)
  - Run by:
    `> valgrind Code/GraphMol/MolStandardize/molStandardizeTest`
- Unit testing
  - Tests in MolVS
  - PubChem Substance
  - Run by:
    `> ctest -R molValidateTest -V`
- Git (with an organisation such as RDKit)
  - Refer to Landrum/Schneider 2016 UGM talk, slide 8
- Wrapping

Conclusions and outlook

- Tautomer piece is still to do
  - Enumeration code written but not tested.
  - Canonicalization to do.
- Write and wrap functions not covered by the categories.

Thank you for your attention