A Crash Course for SU2 Hackers

Dr. Thomas D. Economon

3rd Annual SU2 Developers Meeting
University of Strathclyde
September 16, 2018
So, you want to be an SU2 developer?

Good news: it’s easy.

We leverage standard development processes and the latest tools for open-source projects.

You will be ready to hack at the end of this talk.
Anyone can be an SU2 developer.
Your starting point: https://github.com/su2code/SU2.
This is Git. It tracks collaborative work on projects through a beautiful distributed graph theory tree model.

Cool. How do we use it?

No idea. Just memorize these shell commands and type them to sync up. If you get errors, save your work elsewhere, delete the project, and download a fresh copy.

http://xkcd.com/1597/
Here’s that list of shell commands you should memorize:

- `$ git clone https://github.com/su2code/SU2.git`
- `$ git branch`
- `$ git checkout -b feature_awesome origin/feature_awesome`
- `$ git status`
- `$ git diff`
- `$ git commit -am “This is an awesome commit.”`
- `$ git push origin feature_awesome`
- `$ git checkout develop`
- `$ git pull origin develop`
- `$ git merge develop`

And their translations:

- Get a fresh copy of the entire repo (master branch to start)
- Check which branches I have locally
- Check out my feature branch that is already on the remote
- Check which files have changed since last commit
- Detailed diff of code changes since last commit
- While working, make commits frequently with messages
- Regularly push to the remote on GitHub
- Switch to the develop branch (assuming you have it locally)
- Merge the changes in the remote develop into local develop
- Merge the changes from local develop into current local branch
We use the popular **Gitflow** branching model.
See all of our public repo branches here.
Note that develop is a protected branch.

A current snapshot of active branches.
New branches can be made in the browser interface here or by pushing local branches to the remote with git.
C++ Source Code in SU2_*/src/, majority of lines in Common/src/ & SU2_CFD/src

IDE project files, e.g., Xcode

Python Scripts

Inviscid NACA 0012

Config files for tests

Run ./bootstrap to reset autotools

Template config file with all options

External source files, e.g., ParMETIS

Here is what you see inside the SU2/ repo.
C++ Executables
• SU2_CFD  -> Primary multiphysics PDE solver for primal and adjoint
• SU2_SOL  -> Solution export code
• SU2_DEF  -> Mesh deformation
• SU2_DOT  -> Gradient projection
• SU2_GEO  -> Geometry definition
• SU2_MSH  -> Mesh adaptation

Python Scripts (just a subset of them)
• parallel_computation.py
• mesh_deformation.py
• shape_optimization.py
• continuous_adjoint.py
• discrete_adjoint.py
• finite_differences.py
• direct_differentiation.py
C++ class abstractions encourage code reuse and data encapsulation ensures you can make localized changes easily.

Common base classes/methods (grid, linear solvers, output, etc.) are reused for many sets of physical governing equations.

For a particular PDE, we define iteration, numerics, solver, and variable classes that are customized for the particular methods and algorithms.

Files with *_structure.cpp contain base classes.

Files with solver_* .cpp, variable_* .cpp, numerics_* .cpp, contain child classes for a particular PDE, e.g., solver_direct_mean.cpp for mean flow.
Top-down walkthrough of some key classes instantiated for a RANS calculation in SU2_CFD.
Top-down walkthrough of some key classes instantiated for a RANS calculation in SU2_CFD.
Top-down walkthrough of some key classes instantiated for a RANS calculation in SU2_CFD.
Top-down walkthrough of some key classes instantiated for a RANS calculation in SU2_CFD.
Top-down walkthrough of some key classes instantiated for a RANS calculation in SU2_CFD.
Top-down walkthrough of some key classes instantiated for a RANS calculation in SU2_CFD.
Top-down walkthrough of some key classes instantiated for a RANS calculation in SU2_CFD.

/*--- Roe's Flux approximation ---*/

for (iVar = 0; iVar < nVar; iVar++) {
    val_residual[iVar] = kappa*(ProjFlux_i[iVar]+ProjFlux_j[iVar]);
    for (jVar = 0; jVar < nVar; jVar++) {
        Proj_ModJac_Tensor_ij = 0.0;
        /*--- Compute |Proj_ModJac_Tensor| = P x |Lambda| x inverse P ---*/
        for (kVar = 0; kVar < nVar; kVar++)
            Proj_ModJac_Tensor_ij += P_Tensor[iVar][kVar]*Lambda[kVar]*invP_Tensor[kVar][jVar];
        val_residual[iVar] -= (1.0-kappa)*Proj_ModJac_Tensor_ij*Diff_U[jVar]*Area*Dissipation_ij;
        if(implicit){
            val_Jacobian_i[iVar][jVar] += (1.0-kappa)*Proj_ModJac_Tensor_ij*Area;
            val_Jacobian_j[iVar][jVar] -= (1.0-kappa)*Proj_ModJac_Tensor_ij*Area;
        }
    }
}
Top-down walkthrough of some key classes instantiated for a RANS calculation in SU2_CFD.
Top-down walkthrough of some key classes instantiated for a RANS calculation in SU2_CFD.
• Now that you know the basics, you are ready to create a new branch for your awesome feature (feature_awesome) and start hacking.

• But you might say, "Wait, how do I coordinate my contribution with other ongoing work in the repository?"

  • Posting **issues on GitHub** and interacting with the SU2 IDS are great ways to discuss potential developments and coordinate among other developers in the open.

• And then maybe you’ll ask, “How can I make sure that my work doesn’t ‘break’ other capabilities that already exist in SU2?”

  • **Continuous integration will save your bacon.** Travis CI is free for open-source!
Our security blanket: a comprehensive suite of ~200 regression test cases for serial, parallel, physics, AD, python, etc.
Use continuous integration to guide your development by activating Travis CI for your own branches! Update SU2/.travis.yml with your own email and branch.

---

# Continuous Integration setup for SU2.

dist: trusty
sudo: required

language: python

compiler:
  - gcc

notifications:
  email:
    recipients:
      - your.email@here.com

branches:
  only:
    - feature_awesome
New capabilities in your feature branch should also have a test case to protect them in the future.

```python
# NACA0012
naca0012 = TestCase('naca0012')
naca0012.cfg_dir = "euler/naca0012"
naca0012.cfg_file = "inv_NACA0012_Roe.cfg"
naca0012.test_iter = 20
naca0012.test_vals = [-4.047448, -3.538057, 0.338691, 0.023131]  #last 4 columns
naca0012.su2_exec = "SU2_CFD"
naca0012.timeout = 1600
naca0012.tol = 0.00001

If the computed values after one of your commits don't match these values, you will get an email with details of the failed cases. Investigate it!
```

1. Add a new test case to serial_regression.py, parallel_regression.py, etc. Use others as a guide. See NACA 0012 example.
2. Put the config file and any supporting data in the corresponding locations. Travis CI combines the complementary sets.

<table>
<thead>
<tr>
<th>su2code / SU2</th>
<th>su2code / TestCases</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Branch:</strong> master</td>
<td><strong>Branch:</strong> master</td>
</tr>
<tr>
<td>SU2 / TestCases / euler / naca0012 /</td>
<td>TestCases / euler / naca0012 /</td>
</tr>
<tr>
<td>invol_NACA0012.cfg</td>
<td>Mesh_NACA0012_inv.su2</td>
</tr>
<tr>
<td>Merging develop.</td>
<td>Added all of the mesh files that are &lt; 30 MB</td>
</tr>
<tr>
<td>3 months ago</td>
<td>4 years ago</td>
</tr>
</tbody>
</table>

Lighter weight, more frequently updated files go in the code repo.

Larger, more static files that support the tests go in the TestCases repo.
• So, you’ve finished your awesome feature and the tests are passing. You’ve even added your own regression test (or two), and you checked that there are no new compiler warnings and the style conforms to the SU2 standard.

• At this point you are wondering, “I would like to contribute my feature to the open source, but how do I do that?”

• To get your work into an official open-source release of SU2, the modifications have to first be staged in the develop branch.

• To do so, we use the standard Pull Request (PR) approach.
Once you're ready to contribute, it's PR time.
A PR is a request to the project to pull in your contribution. Can be from an internal branch or from an external fork.
Submit the PR to the develop branch

Fill out the PR template questions that guide you along your way.
PRs keep community informed, offer opportunity for discussion, and are a **controlled gate** for quality assurance of contributions.

All regression tests must pass with your code integrated. Travis CI again takes care of this for us. Merging is blocked until passage.

Code is reviewed by fellow developers for content, organization, and style. PR is blocked until at least one approval! Our convention is 2 reviewer approvals.
Details of the tests for all PRs can be found over in Travis CI.

<table>
<thead>
<tr>
<th>PR #</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>#570</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Feature sst uq</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Jayant Mukhopadhyaya</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#2342</td>
<td>passed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>d7e059f</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7 hrs 54 min 5 sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>17 hours ago</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#565</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Feature hom</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>vdweide</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#2341</td>
<td>passed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>e82f700</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9 hrs 33 min 54 sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>a day ago</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#565</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Feature hom</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>vdweide</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#2338</td>
<td>passed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>e6418bb</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9 hrs 34 min 20 sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 days ago</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#579</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fix Commands for SQL_FSI, SQL and GEO in SU2_PY/SU2/</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Patrick</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#2335</td>
<td>passed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>e9f9513</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 hrs 1 min 48 sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 days ago</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#565</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Feature hom</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>vdweide</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#2334</td>
<td>passed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>be211e3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9 hrs 29 min 55 sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 days ago</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#565</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Feature hom</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>vdweide</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#2333</td>
<td>passed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>c3f74e9</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9 hrs 30 min 44 sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 days ago</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#570</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Feature sst uq</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Jayant Mukhopadhyaya</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#2332</td>
<td>failed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>134887</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 hrs 4 min 19 sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 days ago</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#565</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Feature hom</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>vdweide</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#2333</td>
<td>passed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7029bc1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11 hrs 27 min 48 sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 days ago</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#574</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Feature error message</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>vdweide</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#2330</td>
<td>passed</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2d4bd6f</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 hrs 8 min 38 sec</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 days ago</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Releases: we move develop to master, create tags, binaries, and advertise. Your awesome feature is released!
Git/Branching  Development  Regressions  Pull Request  Release

Documentation and tutorials are critical for amplifying the impact of your work. Good news: it's the same process to create it.
Keep up-to-date via email with all of the activity in the repo by “watching”
SU2 Development Survival Guide
A Best Practice Workflow

1. Clone main repository: $ git clone https://github.com/su2code/SU2.git

2. Create new feature branch (in remote and locally) for your development work. Work on this branch in the repo.

3. Activate the regressions for your branch by changing to your branch name and email in .travis.yml. Use this to guide development and correct any failures along the way that you will be informed of by email.

4. Work on your feature! Please mind white space issues, compiler warnings, and match the SU2 style.

5. If you are working on a single branch for an extended amount of time, merge the remote develop branch into your own branch at regular, frequent intervals. This ensures that, when the time comes, it will be easy to merge your contribution into develop, as you will have solved any conflicts on your side before a PR.

6. Once you feel your feature is finished, submit a PR. Fill out the PR template that is provided for you.

7. Get reviews and engage with the community concerning your contribution. Fix problems in your branch or address any feedback on the message boards. Note that any new commits will appear right there in the open PR and will kick-off the regressions again.

8. Once the reviewers approve and the regressions pass, the community will merge in your work.

9. Celebrate your contribution and proudly introduce yourself as an SU2 developer at your social engagements.
Pretty easy to be a developer, huh?

We have set up safety nets and removed overhead wherever possible.

So, try crazy ideas in your branches. Don’t be afraid to make big changes that push the boundaries of the code. The community and infrastructure will be there to help you. This is how we make progress.
3rd Annual SU2 Developers Meeting
September 16th-18th, 2018
University of Strathclyde, Scottish Universities Insight Institute (SUII)
Glasgow, Scotland, UK

Meeting Agenda for Sunday September 16th

0900 – 0915: Welcome & Agenda
0915 – 1045: Introduction to developing in SU2: Covering high level class design, how to modify the code, working with GitHub (branching, PRs, regressions), etc.
1045 – 1615: Hack session: Separate groups working on various problems (lunch and snacks/coffee offered in the room while working)
1615 – 1700: Wrap-up Presentations: Two slide presentations on major progress for the day, including discussion

1730 – open: Social at "The Counting House", 2 St Vincent Place, G1 2DH

Meeting Agenda for Monday September 17th

0800 – 0830: Welcome & Year in review, T. Economon (Bosch), J.J. Alonso (Stanford)
0900 – 0930: Toward optimization for reactive flows in SU2, N. Beishuizen (Bosch), D. Mayer, T. Economon
0930 – 1000: Conjugate heat transfer problems and computing coupled discrete adjoints using AD, O. Burghardt (TU Kaiserslautern), T. Albring, N. Gauger
1000 – 1030: Coffee break
1200 – 1300: Lunch
1300 – 1330: Unsteady optimization with SU2: application to turbomachinery design, A. Rubino (TU Delft), M. Pini, N. Anand, P. Colonna
1330 – 1400: Preliminary results on rotor-fuselage aerodynamics using SU2: status and challenges, M. Morelli (Politecnico di Milano), G. Gori, A. Guardone
1400 – 1430: Anisotropic mesh adaptation with the INRIA AMG library, A. Loseille (INRIA), V. Menier, B. Munguia, J.J. Alonso
1430 – 1500: Coffee break
1500 – 1530: Simulation and adjoint-based design for variable density incompressible flows with heat transfer, T. Economon (Bosch)
1530 – 1600: Implementation of pressure-based Navier-Stokes for wind energy applications, A. Ravishankara (ECN part of TNO), H. Ozdemir, E. van der Weide
1630 – 1700: Wrap up, J.J. Alonso (Stanford)

In order to participate (in-person or virtually), please register for the meeting by following the link on the SU2 home page (https://su2code.github.io).

*Please note that all stated times are British Summer Time (BST). **The presenter author is underlined