



THE UNIVERSITY  
of EDINBURGH



# Pulling everything together

Emmanouil (Manos) Farsarakis  
EPCC, The University of Edinburgh  
[farsarakis@epcc.ed.ac.uk](mailto:farsarakis@epcc.ed.ac.uk)

Imperial College London, April 2015

# Reusing this material



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

[http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en\\_US](http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en_US)

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.



|epcc|



<http://www.archer.ac.uk>  
[support@archer.ac.uk](mailto:support@archer.ac.uk)



PARTNERSHIP  
FOR ADVANCED COMPUTING  
IN EUROPE



<http://www.training.prace-ri.eu>

# Best practices for scientific computing

1. Write programs for people, not computers
  - A. A program should not require its readers to hold more than a handful of facts in memory at once
  - B. Make names consistent, distinctive, and meaningful
  - C. Make code style and formatting consistent

# Best practices for scientific computing

1. Write programs for people, not computers
2. Let the computer do the work
  - A. Make the computer repeat tasks
  - B. Save recent commands in a file for re-use
  - C. Use a build tool to automate workflows

# Best practices for scientific computing

1. Write programs for people, not computers
2. Let the computer do the work
3. Make incremental changes
  - A. Work in small steps with frequent feedback and course correction
  - B. Use a version control system
  - C. Put everything that has been created manually in version control

# Best practices for scientific computing

1. Write programs for people, not computers
2. Let the computer do the work
3. Make incremental changes
4. Don't repeat yourself (or others)
  - A. Every piece of data must have a single authoritative representation in the system
  - B. Modularise code rather than copying and pasting
  - C. Re-use code instead of rewriting it

# Best practices for scientific computing

1. Write programs for people, not computers
2. Let the computer do the work
3. Make incremental changes
4. Don't repeat yourself (or others)
5. Plan for mistakes
  - A. Add assertions to programs to check their operation
  - B. Use an off-the-shelf unit testing library
  - C. Turn bugs into test cases
  - D. Use a symbolic debugger

# Best practices for scientific computing

1. Write programs for people, not computers
2. Let the computer do the work
3. Make incremental changes
4. Don't repeat yourself (or others)
5. Plan for mistakes
6. Optimise software only after it works correctly
  - A. Use a profiler to identify bottlenecks
  - B. Write code in the highest-level language possible

# Best practices for scientific computing

1. Write programs for people, not computers
2. Let the computer do the work
3. Make incremental changes
4. Don't repeat yourself (or others)
5. Plan for mistakes
6. Optimise software only after it works correctly
7. Document design and purpose, not mechanics
  - A. Document interfaces and reasons, not implementations
  - B. Refactor code in preference to explaining how it works
  - C. Embed the documentation for a piece of software in that software

# Best practices for scientific computing

1. Write programs for people, not computers

2. Let the computer do the work

3. Make incremental changes

4. Don't repeat yourself (or others)

5. Plan for mistakes

6. Optimise software only after it works correctly

7. Document design and purpose, not mechanics

8. Collaborate

A. Use pre-merge code reviews

B. Use pair programming when bringing someone new up to speed and when tackling particularly tricky problems

C. Use an issue tracking tool

# 10 rules for reproducible computational research

1. For every result, keep track of how it was produced
2. Avoid manual data manipulation steps
3. Archive the exact versions of all external programs used
4. Version control all custom scripts
5. Record all intermediate results, when possible in standardized formats
6. For analyses that include randomness, note underlying random seeds
7. Always store raw data behind plots
8. Generate hierarchical analysis output, allowing layers of increasing detail to be inspected.
9. Connect textual statements to underlying results
10. Provide public access to scripts, runs, and results

Sandve GK, Nekrutenko A, Taylor J, Hovig E (2013) Ten Simple Rules for Reproducible Computational Research. PLoS Comput Biol 9(10): e1003285. doi:10.1371/journal.pcbi.1003285. <http://dx.doi.org/10.1371/journal.pcbi.1003285>.



# Iterative Development

- > What did you think of this workshop?
- > Go to:
  - > <http://www.archer.ac.uk/training/feedback>
- > Give us two sticky notes on your way out:



# Getting access to ARCHER

- > Standard research grant
  - > Request Technical Assessment using form on ARCHER website
  - > Submit completed TA with notional cost in Je-S
  - > Apply for time for maximum of 2 years
- > ARCHER Resource Allocation Panel (RAP)
  - > Request Technical Assessment using form on ARCHER website
  - > Submit completed TA with RAP form
  - > Every 4 months
- > Application for computer time only
  - > Instant Access - Pump-Priming Time
  - > Request Technical Assessment using form on ARCHER website
  - > Submit completed TA with 2 page description of work

# Support and Documentation

- > Helpdesk
  - > Email [support@archer.ac.uk](mailto:support@archer.ac.uk)
  - > via ARCHER SAFE <http://www.archer.ac.uk/safe>
  - > Phone: +44 (0) 131 650 5000
  - > By post, to: Liz Sim  
EPCC, University of Edinburgh  
JCMB, The King's Buildings  
Peter Guthrie Tait Road  
Edinburgh, EH9 3FD
- > <http://www.archer.ac.uk/community/techforum>
- > <http://www.archer.ac.uk/documentation/>

# Training opportunities

- > **ARCHER Training (free to academics)**
  - > <http://www.archer.ac.uk/training/>
- > **Online sessions (using Blackboard Collaborate)**
  - > **Technical Forum meetings (normally 15:00 last Wednesday of month)**
    - > technical presentations of interest to ARCHER users
    - > <http://www.archer.ac.uk/community/techforum>
  - > **Virtual Tutorials (normally 15:00 second Wednesday of month)**
    - > opportunity for discussion with ARCHER staff on any topic
    - > usually starts with a presentation of general interest
    - > <http://www.archer.ac.uk/training/virtual>
- > **EPCC MSc in HPC (scholarships available)**
  - > <http://www.epcc.ed.ac.uk/msc/>

# Funding Calls

- > **Embedded SCE support**
  - > **Through a series of regular calls, Embedded CSE (eCSE) support provides funding to the ARCHER user community to develop software in a sustainable manner for running on ARCHER. Funding will enable the employment of a researcher or code developer to work specifically on the relevant software to enable new features or improve the performance of the code**
  - > **Apply for funding for development effort**
  - > **Planned every 4 months**
- > See <http://www.archer.ac.uk/community/eCSE/> for details

# Getting involved in Software Carpentry



# Looking back...



# Leaving here you will have learned to:

- Write robust code
- Minimize repetitive tasks
- Appreciate the value of “sharable” code
- Share your work
- Protect your work
- Collaborate on other people’s work



# Leaving here you will (hopefully) have learned to:

- Write robust code
- Minimize repetitive tasks
- Appreciate the value of “sharable” code
- Share your work
- Protect your work
- Collaborate on other people’s work



# Thank you!



# Thank you!

and remember...

Good point  
I learned...  
(and/or)  
I enjoyed...



Bad Point  
...was confusing  
(and/or)  
...bothered me.