Introduction to Scientific Software Deployment and Development

damien.francois@uclouvain.be
October 2015
What is this?

[Image of the Periodic Table of DevOps Tools]

Goal of this session:

“Give you access to the same tools the professionals are using for developing and deploying programs.”
Dev's toolkit:

1. Programming languages
2. Good practices / Code Style Guides
3. Text editor
4. Source control management
5. Debuggers / Profilers
6. Databases
7. Packaging / Distributing tools
1. Programming language

- **Good reasons** for choosing language X:
  - it offers useful paradigms for your problem
  - it offers high-level constructs/tools - e.g. for parsing arguments
  - it offers (directly or indirectly) useful libraries - e.g. for linear algebra

- **Ok reasons** for choosing language X:
  - standard in your community – easier to get accepted

- **Bad reasons** for choosing language X:
  - it runs fast – probably needs high skills
  - it is the language you already know – screwing with a hammer?
A word about paradigms

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Example Languages</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imperative – “Do this”</td>
<td>BASIC, Assembly</td>
<td>good for explicit computing</td>
</tr>
<tr>
<td>Structured – Subroutines, scopes</td>
<td>C, FORTRAN77</td>
<td>algorithms + data : good for explicit computing</td>
</tr>
<tr>
<td>Object-Oriented – Encapsulation, Inheritance, ...</td>
<td>C++, Python</td>
<td>objects + messages : good for modeling</td>
</tr>
<tr>
<td>Declarative – “I need this”</td>
<td>SQL</td>
<td>good for reasoning</td>
</tr>
<tr>
<td>Functional – Pure functions, lazy evaluation</td>
<td>Haskell, Scala</td>
<td>functions o functions : good for reasoning</td>
</tr>
<tr>
<td>Logic – Predicates and rules</td>
<td>Prolog, Datalog</td>
<td>facts + rules : good for searching</td>
</tr>
</tbody>
</table>
ex.: Haskell

C

```c
void f(int a[], int lo, int hi)
{
    int h, l, p, t;
    if (lo < hi) {
        l = lo;
        h = hi;
        p = a[hi];
        do {
            while ((l < h) && (a[l] <= p))
                l = l+1;
            while ((h > l) && (a[h] >= p))
                h = h-1;
            if (l < h) {
                t = a[l];
                a[l] = a[h];
                a[h] = t;
            }
        } while (l < h);
        a[hi] = a[l];
        a[l] = p;
        f(a, lo, l-1);
        f(a, l+1, hi);
    }
}
```

Haskell

```haskell
qsort []     = []
qsort (p:xs) = (qsort lesser) ++ [p] ++ (qsort greater)
    where
        lesser = filter (< p) xs
        greater = filter (>= p) xs
```

Purely functional
Static strong typing
Lazy evaluation
1. Purely functional

- **Every** input has a corresponding output
- $f(x) = x^2 + 1$
- Powerful **function compositions**
  - $g(x) = x - 1$
  - $g(f(x)) = x^2$
- **PURE**
  - That means *no side effects*
  - A function will never modify a *global variable*
  - Order doesn't matter!
  - Easy concurrency
2. Statically Typed

- \( f \ x = x^2 + 1 \)
- \( f :: \text{Int} \rightarrow \text{Int} \)
- There is never confusion about types (Boo, Int, Char, etc)
- Strong formalism. The proof is the code.
- If your code compiles, you're 99% done

And types can be used to model dimensional numbers (units) or include semantic (e.g. Temperature, Velocity, which will not add together for instance)
Haskell

Lazy?

- Nothing is evaluated unless necessary
  
  ```haskell
  head (sort ls)
  ```

  The list will only be sorted enough to find the minimum

- Allows infinite data structures
  
  ```haskell
  [1..]
  ```
2. Good practices and style

- Write for humans, not for computers
- Organize for change, and make incremental changes
- Plan for mistakes, automate testing
- Automate repetitive tasks
- Write a script rather than a compiled program
- Use modern source-code management system
- Use the appropriate language(s)
- Document the design and purpose, not the implementation
- Optimize only when it works already
Good practices - good reads


Good coding principles

- Don't repeat yourself (DRY)
- Keep it simple, Stupid (KISS)
- One level of abstraction
- Single responsibility principle
- Separation of concern
- Avoid premature optimization
- Many others...

---

Bill Mitchell View profile More options Sep 26 1991, 1:57 am In article <5...@ksr.com> j...@ksr.com (John F. Woods) writes:

[...] Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live. Code for readability.

Damn right!
Coding style

- Makes sure the code is readable by all
  - easily
  - quickly
- Depends on
  - the language
  - the project

```cpp
if (hours < 24 && minutes < 60 && seconds < 60)
{
    return true;
}
else
{
    return false;
}
```

VS

```cpp
if (hours < 24 && minutes < 60 && seconds < 60 )
{
    return true;
}
else
{
    return false;
}
```
Coding style

Google C++ Style Guide

Table of Contents

Header Files
- Self-contained Headers
- The #define Guard
- Forward Declarations
- Inline Functions
- Function Parameter Ordering
- Names and Order of Includes

Scope
- Namespaces
- Nested Classes
- Nonmember, Static Member, and Global Functions
- Static and Global Variables

Classes
- Doing Work in Constructors
- Initialization
- Exotic Constructors
- Copyable and Movable Types
- Delegating and Inheriting Constructors
- Strict vs. Classes
- Inheritance
- Multiple Inheritance
- Operator Overloading
- Access Control
- Declaration Order
- Write Short Functions

Linux kernel coding style

This is a short document describing the preferred coding style for the Linux kernel. Coding style is very personal, and I won't force my views on anybody, but this is what goes for anything that I have to be able to maintain, and I'd prefer it for most other things too. Please at least consider the points made here.

First off, I'd suggest printing out a copy of the GNU coding standards, Burn them, it's a great symbolic gesture.

Chapter 1: Indentation

Statements, and thus indentations are also 8 characters. The movements that try to make indentations 4 (or even 2) spaces and that is akin to trying to define the value of PI to the whole idea behind indentation is to clearly define where lines start and ends. Especially when you've been looking for 20 straight hours, you'll find it a lot easier to see what works if you have large indentations.

I will claim that having 8-character indentations makes for a much easier to read a terminal screen. The answer to that is if you need a lot of indentation, you're screwed anyway, and should fix your problem and you are done.

https://github.com/SalGnt/cscs
3. Text editor

- Many editors with time-saving features
  - Sublime text: http://www.sublimetext.com/
  - Notepad++: https://notepad-plus-plus.org/
  - Text Wrangler: http://www.barebones.com/
  - Textmate: https://macromates.com/
  - Atom: https://atom.io/
- Choose one and learn it from inside out
- And also learn a text-UI editor:
  - Vim or Emacs or Nano
Dev's toolkit:

1. Programming language
2. Good practices / Code Style Guides
3. Text editor / IDE
4. Source control management
5. Debuggers / Profilers
6. Databases
7. Packaging / Distributing tools

Own dedicated sessions
6. Packaging Fortran/C/C++ code


GNU build system

From Wikipedia, the free encyclopedia

This article needs additional citations for verification. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. (September 2009)

The GNU build system, also known as the Autotools, is a suite of programming tools designed to assist in making source code packages portable to many Unix-like systems.

It can be difficult to make a software program portable: the C compiler differs from system to system; certain library functions are missing on some systems; header files may have different names. One way to handle this is to write conditional code, with code blocks selected by means of preprocessor directives (#ifdef); but because of the wide variety of build environments this approach quickly becomes unmanageable. Autotools is designed to address this problem more manageable.

Autotools is part of the GNU toolchain and is widely used in many free software and open source projects. The component tools are software licensed under the GNU General Public License (GPL) and the GNU Lesser General Public License (LGPL).
Licensing your code: Why?

- **Bad reason**:  
  - you want to make money out of it – forbid distribution  
    - forbid reverse engineering

- **Good reason**:  
  - you want to it to be used and get citations  
    - you need to allow usage, and/or modification, etc.  
    - you require others to cite your work  
  - you want to protect yourself from liability claims
Licensing your code: How?

- Choose a license type, e.g.
  - Apache License 2.0
  - BSD 3-Clause "New" or "Revised" license
  - BSD 2-Clause "Simplified" or "FreeBSD" license
  - GNU General Public License (GPL)
  - GNU Library or "Lesser" General Public License (LGPL)
  - MIT license
  - Mozilla Public License 2.0
  - Common Development and Distribution License
  - Eclipse Public License

- Copy/adapt the text
- Distribute a LICENSE file with your code
MIT license

Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

<table>
<thead>
<tr>
<th>Can</th>
<th>Cannot</th>
<th>Must</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commercial Use</td>
<td>Hold Liable</td>
<td>Include Copyright</td>
</tr>
<tr>
<td>Modify</td>
<td></td>
<td>Include License</td>
</tr>
<tr>
<td>Distribute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sublicense</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Private Use</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Other popular licenses

**BSD**

**Can**
- Commercial Use
- Modify
- Distribute
- Place Warranty

**Cannot**
- Use Trademark
- Hold Liable

**Must**
- Include Copyright
- Include License

**GPL**

**Can**
- Commercial Use
- Modify
- Distribute
- Place Warranty
- Use Patent Claims

**Cannot**
- Sublicense
- Hold Liable

**Must**
- Include Original
- State Changes
- Disclose Source
- Include License
- Include Copyright
- Include Install Instructions

Reproduced from https://tldrlegal.com
Ops' toolkit:

1. Virtualization platforms
2. Multi-host connexions
3. Configuration management
4. Installing
5. Automatic build tests
6. Monitoring
1. Virtualization

- Install on your laptop an environment similar to that of the cluster to test your workflow
- With
  - VirtualBox: https://www.virtualbox.org/
  - Vagrant: https://www.vagrantup.com/

you can build a virtual cluster in one command:

“vagrant up”
2. Multi-host SSH

```

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
3. Configuration Management

```
# Configurations

[ceci]
hmem  partition_list=High,Medium,Low
lemaitre2 partition_list=def,PostP
dragon1  partition_list=def,Long
vega    partition_list=defq
hercules partition_list=default
nic4    partition_list=deq

dfr@ncois:~ $ cat /Users/dfr/Confis/Inventory

# Slurm submit template

#SBATCH --partition={%partition_list%}

srun ./myprog

dfr@ncois:~ $ cat Desktop/submit.sh

#! /bin/bash

# Slurm submit template

#SBATCH --partition={%partition_list%}

srun ./myprog

dfr@ncois:~ $ cat Desktop/playbook.yml

---
- hosts: all
tasks:
  - name: Upload default submission script
template: src=./Desktop/submit.sh dest=. mode=750
```

ANSIBLE
3. Configuration Management

```
dfr@ncolis:~ $ ansible-playbook Desktop/playbook.yml

PLAY [all] ********************************************

GATHERING FACTS ********************************************
ok: [hmem]
ok: [lemaitre2]
ok: [hercules]
ok: [vega]
ok: [dragon1]
ok: [nic4]

TASK: [Upload default submission script] ********************************************
changed: [hmem]
changed: [lemaitre2]
changed: [vega]
ok: [hercules]
ok: [dragon1]
ok: [nic4]

PLAY RECAP ********************************************
dragon1 : ok=2    changed=0    unreachable=0    failed=0
hercules : ok=2    changed=0    unreachable=0    failed=0
hmem     : ok=2    changed=1    unreachable=0    failed=0
lemaitre2: ok=2    changed=1    unreachable=0    failed=0
nic4     : ok=2    changed=0    unreachable=0    failed=0
vega     : ok=2    changed=1    unreachable=0    failed=0
```
3. Configuration Management

```bash
# Configuration Management Example

# Slurm submit template

#SBATCH --partition=High,Medium,Low

srun ./myprog

dfr@ncois:~ $ ssh lemaitre2 cat submit.sh
#!/bin/bash

# Slurm submit template

#SBATCH --partition=def,PostP

srun ./myprog

dfr@ncois:~ $ 
```
4. Easy installing

**EasyBuild** is a software build and installation framework that allows you to manage (scientific) software on High Performance Computing (HPC) systems in an efficient way.

---

**Latest news**

- 20150902 - **EasyBuild v2.3.0 is available**
- 20150622 - **10th EasyBuild/Lmod hackathon @ Austin** (before SC15)
- 20150315 - **ISC'15 BoF "Getting Scientific Software Installed" accepted**
- 20141104 - **Revamped documentation @ easybuild.readthedocs.org**
- 20141020 - **pre-print of HUST-14 workshop paper available**

---

**Documentation**

Read the fine manual (RTFM!) at [http://easybuild.readthedocs.org/](http://easybuild.readthedocs.org/).

**Getting started**

The recommended way of installing EasyBuild is via the documented bootstrap procedure. You should **configure** EasyBuild to behave as you prefer, subsequently.
4. Easy installing

```bash
dfer@anneback:-- $ eb -S . 2>/dev/null | head -20
== temporary log file in case of crash /tmp/eb_y2XSC/easybuild-18L_LH.log
== Searching (case-insensitive) for '.' in /usr/lib/python2.6/site-packages/easybuild_easyconfigs-2.3.9-py2.6.egg/easybuild/easyconfigs
  $CFGSI=/usr/lib/python2.6/site-packages/easybuild_easyconfigs-2.3.9-py2.6.egg/easybuild/easyconfigs
  * $CFGSI/TEMPLATE.eb
  * $CFGSI/a/ABAQUS/ABAQUS-6.12.1-linux-x86_64.eb
  * $CFGSI/a/ABAQUS/ABAQUS-6.13.5-linux-x86_64.eb
  * $CFGSI/a/ABAQUS/ABAQUS-6.14.1-linux-x86_64.eb
  * $CFGSI/a/ABINIT/ABINIT-7.0.3-x86_64_linux_gnu4.5.eb
  * $CFGSI/a/ABINIT/ABINIT-7.0.5-x86_64_linux_gnu4.5.eb
  * $CFGSI/a/ABINIT/ABINIT-7.10.4-intel-2015a-incl-deps.eb
  * $CFGSI/a/ABINIT/ABINIT-7.10.4-intel-2015a.eb
  * $CFGSI/a/ABINIT/ABINIT-7.11.6-intel-2015a.eb
  * $CFGSI/a/ABINIT/ABINIT-7.2.1-x86_64_linux_gnu4.5.eb
  * $CFGSI/a/ABINIT/ABINIT-7.4.3-goolf-1.4.10-ETSF_10-1.0.4.eb
  * $CFGSI/a/AbiSS/AbiSS-1.3.4-goolf-1.1.0-no-OFED-Python-2.7.3.eb
  * $CFGSI/a/AbiSS/AbiSS-1.3.4-goolf-1.4.10-Python-2.7.3.eb
  * $CFGSI/a/AbiSS/AbiSS-1.3.4-ictc4-4.0.6-Python-2.7.3.eb
  * $CFGSI/a/AbiSS/AbiSS-1.3.4-ictc4-5.3.0-Python-2.7.3.eb
  * $CFGSI/a/AbiSS/AbiSS-1.3.6-goolf-1.4.10-Python-2.7.5.eb
  * $CFGSI/a/AbiSS/AbiSS-1.3.7-intel-2015a-Python-2.7.9.eb
```

dfer@anneback:-- $ eb -S . | cut -d/ -f3 | sort -u | wc -l
742
dfer@anneback:-- $
Packaging compiled code

CDE: Automatically create portable Linux applications

CDE (formerly known as CDEpack) automatically packages up the Code, Data, and Environment required to deploy and run your Linux programs on other machines without any installation or configuration. CDE is the easiest way to completely eliminate dependency hell.

To get started, download the CDE binary (32-bit or 64-bit) and follow these steps:

1. Package
   Prepend any set of Linux commands with the "cde" binary, and CDE will run them and automatically package up all files (e.g., executables, libraries, plug-ins, config/data files) accessed during execution.

2. Deliver
   A package is simply a directory that can be compressed and delivered to any x86-Linux machine. It contains all the files and environment variables required to run your original commands. Packages can range from 10 to 100 MB in size.

3. Run
   After receiving the package, the user can now run those same commands from within the package on any modern x86-Linux distro. The user does not need to first compile, install, or configure anything.

http://www.pgbovine.net/cde.html
5. Automatic build tests

Jenkins

An extensible open source continuous integration server

- Meet Jenkins
  Find out what Jenkins is and get started.

- Use Jenkins
  See how to get more out of your Jenkins.

- Customize Jenkins
  Choose from 1092 plugins to customize Jenkins exactly to your needs.

Download Jenkins

- Java Web Archive (.war)
  Latest and greatest (1.631)
  changelog | past releases
  upgrading from Hudson?

- Native packages
  Windows
  Ubuntu/Debian
  Red Hat/Fedora/CentOS
  Mac OS X
  openSUSE
  FreeBSD
  OpenBSD
6. Monitoring

Useful with command 'screen'

```
multitail -q 2 "**/res"
```
6. Monitoring

Useful with command 'screen'
Dev's toolkit:

1. Programming language
2. Good practices / Code Style Guides
3. Text editor / IDE
4. Source control management
5. Debuggers / Profilers
6. Databases
7. Packaging / Distributing tools
Ops' toolkit:

1. Virtualization platforms (Virtual box, Vagrant)
2. Multi-host connexion (pdsh)
3. Configuration management/ (ansible)
4. Installing (easybuild)
5. Automatic build tests (jenkins)
6. Monitoring (multitail)
The 'Phillip' test

- 12 simple questions
- ordered by 'difficulty'
- measures quality of organization
- for research programming

If you do not score at least a 7 there is room for improvement using the tools presented here

1. Do you have reliable ways of taking, organizing, and reflecting on notes as you're working?
2. Do you have reliable to-do lists for your projects?
3. Do you write scripts to automate repetitive tasks?
4. Are your scripts, data sets, and notes backed up on another computer?
5. Can you quickly identify errors and inconsistencies in your raw data sets?
6. Can you write scripts to acquire and merge together data from different sources and in different formats?
7. Do you use version control for your scripts?
8. If you show analysis results to a colleague and they offer a suggestion for improvement, can you adjust your script, re-run it, and produce updated results within an hour?
9. Do you use assert statements and test cases to sanity check the outputs of your analyses?
10. Can you re-generate any intermediate data set from the original raw data by running a series of scripts?
11. Can you re-generate all of the figures and tables in your research paper by running a single command?
12. If you get hit by a bus, can one of your lab-mates resume your research where you left off with less than a week of delay?

http://pgbovine.net/research-programming-workflow.htm
Work quicker & more reliably