Introduction to Scientific Software Deployment and Development

damien.francois@uclouvain.be
November 2024

http://www.ceci-hpc.be/training.html

PERIODIC TABLE OF DEVOPS TOOLS (V1)

XebiaLabs
Deliver Faster

Os	Open Source
Fr	Free
Fm	Freemium
Pd	Paid
En	Enterprise

Database	SCM	Build
CI	Repo Mgmt	Testing
Deployment	Config / Provisioning	Containerization
Cloud / Iaas / Paas	Release Mgmt	Collaboration
BI / Monitoring	Logging	Security

1 En O 12c																		2 Fm Aws Amazon Web Services
3 Os My MySQL	4 Os Gt Git											5 En Ch Chef	6 En Pu Puppet	7 Os An Ansible	8 En Sl Salt	9 Os Dk Docker	10 Pd Az Azure	
11 En Mq MSSQL	12 Os Sv Subversion											13 Fr Ssh SSH	14 En Bl BladeLogic	15 Os Va Vagrant	16 Fr Tf Terraform	17 Os Rk rkt	18 Fm Hk Heroku	
19 Os Pq PostgreSQL	20 Fm Gh Github	21 Os Mv Maven	22 Os Gr Gradle	23 En Mr Meister	24 Os Jn Jenkins	25 Pd Ba Bamboo	26 Os Tr Travis CI	27 Fr Ar Archiva	28 Os Fn FitNesse	29 Fr Se Selenium	30 Os Gn Gatling	31 Pd Gd Deployment Manager	32 Os Sf SmartFrog	33 Fr Cb Cobbler	34 Os Bc Bcfg2	35 Os Kb Kubernetes	36 En Rs Rackspace	
37 Os Mg MongoDB	38 Fm Bb Bitbucket	39 Os Br Buildr	40 Os At ANT	41 Fm Bm BuildMaster	42 Fm Cs Codeship	43 Fm Sn Snap CI	44 Fm Cr CircleCI	45 Os Nx Nexus	46 Fr Cu Cucumber	47 Os Cj Cucumberjs	48 Fr Qu Qunit	49 Fr Cp Capistrano	50 Fr Ju JuJu	51 Os Rd Rundeck	52 Os Cf CFEngine	53 Fr Pk Packer	54 Fm Bx Bluemix	
55 En Db DB2	56 Os Hg Mercurial	57 Fm Qb QuickBuild	58 En Ub UrbanCode Build	59 Pd Ta Visual Build	60 Fm Tc TeamCity	61 Fm Sh Shippable	62 Os Cc CruiseControl	63 Os Ay Artifactory	64 Fr Jt JUnit	65 Fr Jm JMeter	66 Fr Tn TestNG	67 En Ry RapidDeploy	68 Fm Cy CodeDeploy	69 En Oc Octopus Deploy	70 No CA Nolio	71 En Eb ElasticBox	72 En Ad Apprenda	
73 Fr Cs Cassandra	74 En Hx Helix	75 Os Msb MSBuild	76 Os Rk Rake	77 Os Lb LuntBuild	78 Os Co Continuum	79 Fm Ca Continua CI	80 Os Gu Gump	81 Os Ng NuGet	82 Os Ap Appium	83 En Xltv XL TestView	84 En Tc TestComplete	85 Os Go Go	86 En Ef ElectricFlow	87 En Xld XL Deploy	88 En Ud UrbanCode Deploy	89 Os Mo Mesos	90 Os Cf Cloud	

Share

Embed

Become Excellent!

Subscribe here!

91 En Xlr XL Release	92 En Ur UrbanCode Release	93 En Ls CA Service Virtualization	94 En Bm BMC Release Process	95 En Hp HP Codar	96 Pd Ex Excel	97 En Pl Plutora Release	98 En Sr Serena Release	99 Fm Tr Trello	100 Pd Jr Jira	101 Fm Rf HipChat	102 Fm Sl Slack	103 Fm Fd Flowdock	104 Pd Pv Pivotal Tracker	105 En Sn ServiceNow				
106 Os Ki Kibana	107 Fm Nr New Relic	108 Os Ni Nagios	109 Os Gg Ganglia	110 Os Ct Cacti	111 Os Gr Graphite	112 Os Ic Icinga	113 En Sp Splunk	114 Fm Sl Sumo Logic	115 Os Ls Logstash	116 Fm Lg Loggly	117 Os Gr Graylog	118 Os Sn Snort	119 Os Tr Tripwire	120 En Cy CyberArk				

https://xebialabs.com/periodic-table-of-devops-tools/

Goal of this session:

“Promote the tools
the professionals are using for
developing and **deploying** programs,
to make them **correct**, **maintainable**, **shareable**, and **fast**,
efficiently.”

“...to make them **correct** and **maintainable**, …, *efficiently*”

Paul F. Dubois. 1999. **Ten Good Practices in Scientific Programming**. *Computing in Science and Eng.* 1, 1 (January 1999), 7-11. DOI=10.1109/MCISE.1999.743610 http://dx.doi.org/10.1109/MCISE.1999.743610

Wilson G, Aruliah DA, Brown CT, Chue Hong NP, Davis M, Guy RT, et al. (2014) **Best Practices for Scientific Computing**. *PLoS Biol* 12(1): e1001745. doi:10.1371/journal.pbio.1001745

Dubois PF, Epperly T, Kumfert G (2003) **Why Johnny can't build (portable scientific software)**. *Comput Sci Eng* 5: 83–88. doi: 10.1109/mcise.2003.1225867

Prlić A, Procter JB (2012) **Ten Simple Rules for the Open Development of Scientific Software**. *PLoS Comput Biol* 8(12): e1002802. doi:10.1371/journal.pcbi.1002802

Victor R. Basili, Jeffrey C. Carver, Daniela Cruzes, Lorin M. Hochstein, Jeffrey K. Hollingsworth, Forrest Shull, Marvin V. Zelkowitz, "**Understanding the High-Performance-Computing Community: A Software Engineer's Perspective**," *IEEE Software*, vol. 25, no. 4, pp. 29-36, July/August, 2008

Wilson G, Bryan J, Cranston K, Kitzes J, Nederbragt L, Teal TK (2017) **Good enough practices in scientific computing**. *PLoS Comput Biol* 13(6): e1005510. https://doi.org/10.1371/journal.pcbi.1005510

Koehler Leman J *et al* “**Better together: Elements of successful scientific software development in a distributed collaborative community**. *PLoS Comput Biol.* 2020 doi: 10.1371/journal.pcbi.1007507.

Arvanitou E.-M., Ampatzoglou A, Chatzigeorgiou A, Carver J, **Software engineering practices for scientific software development: A systematic mapping study**, *Journal of Systems and Software*, Volume 172, 2021 https://doi.org/10.1016/j.jss.2020.110848.

Wilson G, Bryan J, Cranston K, Kitzes J, Nederbragt L, Teal TK (2017) **Good enough practices in scientific computing**. *PLoS Comput Biol* 13(6): e1005510. https://doi.org/10.1371/journal.pcbi.1005510

“...to make them **correct** and **maintainable**, …, *efficiently*”

Follow programming good practices:

1. Write for humans, not for computers
2. Use the appropriate language
3. Organize for change and make incremental changes
4. Follow good coding principles
5. Plan for mistakes, automate testing
6. Use modern source-code management system
7. Document the design and purpose, not the implementation
8. Optimize only when it works already
9. Debug cleverly

1. Write for humans, not for computers

"Indeed, the ratio of time spent reading versus writing is well over 10 to 1. We are constantly reading old code as part of the effort to write new code. ...[Therefore,] making it easy to read makes it easier to write."

Robert C. Martin *Clean code A Handbook of Agile Software Craftsmanship*, 2009

1. Write for humans, not for computers

Structure clear but goal not obvious

```
for i in range(n):
    for j in range(m):
        for k in range(l):
            temp_value = X[i][j][k] * 12.5
            new_array[i][j][k] = temp_value + 150
```

vs

Sweet spot in-between...

Goal clear but structure less obvious

```
PIXEL_NORMALIZATION_FACTOR = 12.5
PIXEL_OFFSET_FACTOR = 150

for row_index in range(row_count):
    for column_index in range(column_count):
        for color_channel_index in range(color_channel_count):
            normalized_pixel_value = (
                original_pixel_array[row_index][column_index][color_channel_index]
                * PIXEL_NORMALIZATION_FACTOR
            )
            transformed_pixel_array[row_index][column_index][color_channel_index] = (
                normalized_pixel_value + PIXEL_OFFSET_FACTOR
            )
```

https://towardsdatascience.com/data-scientists-your-variable-names-are-awful-heres-how-to-fix-them-89053d2855be

1. Write for humans, not for computers

Avoid naming anti-patterns:

A.1	*"Get" - more than an accessor*	A getter that performs actions other than returning the corresponding attribute without documenting it. Example: method `getImageData` which, no matter the attribute value, every time returns a new object (see Fig. 1).
A.2	*"Is" returns more than a Boolean*	The name of a method is a predicate suggesting a true/false value in return. However the return type is not Boolean but rather a more complex type thus allowing a wider range of values without documenting them. Example: `isValid` with return type `int` (see Fig. 6).
A.3	*"Set" method returns*	A set method having a return type different than `void` and not documenting the return type/values with an appropriate comment (see Fig. 7).
A.4	*Expecting but not getting a single instance*	The name of a method indicates that a single object is returned but the return type is a collection. Example: method `getExpansion` returning `List` (see Fig. 9).
B.1	*Not implemented condition*	The comments of a method suggest a conditional behavior that is not implemented in the code. When the implementation is default this should be documented (see Fig. 10).
B.2	*Validation method does not confirm*	A validation method (e.g., name starting with "validate", "check", "ensure") does not confirm the validation, i.e., the method neither provides a return value informing whether the validation was successful, nor documents how to proceed to understand (see Fig. 11).
B.3	*"Get" method does not return*	The name suggests that the method returns something (e.g., name starts with "get" or "return") but the return type is `void`. The documentation should explain where the resulting data is stored and how to obtain it (see Fig. 12).
B.4	*Not answered question*	The name of a method is in the form of predicate whereas the return type is not Boolean. Example: method `isValid` with return type `void` (see Fig. 13).
B.5	*Transform method does not return*	The name of a method suggests the transformation of an object but there is no return value and it is not clear from the documentation where the result is stored. Example: method `javaToNative` with return type `void` (see Fig. 14).
B.6	*Expecting but not getting a collection*	The name of a method suggests that a collection should be returned but a single object or nothing is returned. Example: method `getStats` with return type Boolean (see Fig. 15).
C.1	*Method name and return type are opposite*	The intent of the method suggested by its name is in contradiction with what it returns. Example: method `disable` with return type `ControlEnableState`. The inconsistency comes from "disable" and "enable" having opposite meanings (see Fig. 16).
C.2	*Method signature and comment are opposite*	The documentation of a method is in contradiction with its declaration. Example: method `isNavigateForwardEnabled` is in contradiction with its comment documenting "a back navigation", as "forward" and "back" are antonyms (see Fig. 17).
D.1	*Says one but contains many*	The name of an attribute suggests a single instance, while its type suggests that the attribute stores a collection of objects. Example: attribute `target` of type `Vector`. It is unclear whether a change affects one or multiple instances in the collection (see Fig. 18).
D.2	*Name suggests Boolean but type does not*	The name of an attribute suggests that its value is true or false, but its declaring type is not Boolean. Example: attribute `isReached` of type `int[]` where the declared type and values are not documented (see Fig. 19).
E.1	*Says many but contains one*	The name of an attribute suggests multiple instances, but its type suggests a single one. Example: attribute `stats` of type Boolean. Documenting such inconsistencies avoids additional comprehension effort to understand the purpose of the attribute (see Fig. 20).
F.1	*Attribute name and type are opposite*	The name of an attribute is in contradiction with its type as they contain antonyms. Example: attribute `start` of type `MAssociationEnd`. The use of antonyms can induce wrong assumptions (see Fig. 21).
F.2	*Attribute signature and comment are opposite*	The declaration of an attribute is in contradiction with its documentation. Example: attribute `INCLUDE_NAME_DEFAULT` whose comment documents an "exclude pattern". Whether the pattern is included or excluded is thus unclear (see Fig. 22).

Arnaoudova, *et al*. Linguistic antipatterns: what they are and how developers perceive them. Empir Software Eng 21, 104–158 (2016).

1. Write for humans, not for computers

stackoverflow Products Search...

Home

PUBLIC

Questions

Tags

Users

Companies

COLLECTIVES

Explore Collectives

TEAMS

Create free Team

What does the ??!??! operator do in C?

Asked 11 years ago Modified 1 year, 2 months ago Viewed 379k times

2471

Highly active question. You have enough reputation to answer or unprotect this question.

I saw a line of C that looked like this:

```
!ErrorHasOccured() ??!??! HandleError();
```

It compiled correctly and seems to run ok. It seems like it's checking if an error has occurred, and if it has, it handles it. But I'm not really sure what it's actually doing or how it's doing it. It does look like the programmer is trying express their feelings about errors.

I have never seen the `??!??!` before in any programming language, and I can't find documentation for it anywhere. (Google doesn't help with search terms like `??!??!`). What does it do and how does the code sample work?

https://stackoverflow.com/questions/7825055/what-does-the-operator-do-in-c

1. Write for humans, not for computers

stackoverflow Products Search...

Home

PUBLIC

Questions

Tags

Users

Companies

COLLECTIVES

Explore Collectives

TEAMS

Create free Team

What does the ??!??! operator do in C?

Asked 11 years ago Modified 1 year, 2 months ago Viewed 379k times

Highly active question. You have enough reputation to answer or unprotect this question.

2471

```
if (ErrorHasOccured())
    HandleError();
```

I saw a line of C that looked like this:

```
!ErrorHasOccured() ??!??! HandleError();
```

It compiled correctly and seems to run ok. It seems like it's checking if an error has occurred, and if it has, it handles it. But I'm not really sure what it's actually doing or how it's doing it. It does look like the programmer is trying express their feelings about errors.

I have never seen the `??!??!` before in any programming language, and I can't find documentation for it anywhere. (Google doesn't help with search terms like `??!??!`). What does it do and how does the code sample work?

https://stackoverflow.com/questions/7825055/what-does-the-operator-do-in-c

2. Use the appropriate language

are all valid choices in a scientific context.

2. Use the appropriate language

What they have in common:

- Computation-efficiency concern
- Optimized libraries available for linear algebra, signal processing, learning, etc.
- Support for parallel computing
- Extensions/libraries for using accelerators (GPUs)

2. Use the appropriate language

Haskell

Scala

“Functional programming”

Very close to mathematical formulation

Imposes constraints that make code less prone to bugs and easier to make parallel

Not very popular in HPC (yet)

https://madhadron.com/programming/seven_ur_languages.html

3. Organize for change and make incremental changes

Scientific software specifications are always changing:

- Work from working state to another working state
- Document the changes and why they were made
- Refactor upon “code smell”

Keyword: **modularity:** small independent interchangeable building blocks (e.g. functions)

3 ½ . Avoid “code smells” / anti-patterns

PRAGMATIC WAYS
TIPS FOR CLEANER CODE

DUPLICATE CODE

DUPLICATE CODE CAN OCCUR AS A RESULT OF A SHORT DEADLINE, LACK OF COMMUNICATION, OR JUST OUT OF PURE LAZINESS BY THE DEVELOPER.

LEARN ABOUT THE DIFFERENT REFACTORING METHODS TO CLEAN UP DUPLICATE CODE.

THE D.R.Y. PRINCIPLE STANDS FOR DON'T REPEAT YOURSELF

LEARN MORE CLEAN CODE TIPS AT PRAGMATICWAYS.COM

PRAGMATIC WAYS
TIPS FOR CLEANER CODE

SPECULATIVE GENERALTIY

CODE THAT'S CREATED FOR FUTURE USE IS SPECULATIVE GENERALITY. THIS CODE IS NOT CURRENTLY TIED TO ANY REQUIREMENTS BUT THE INTENTIONS ON FUTURE REQUIREMENTS.

THIS CODE CROWDS THE PROJECT AND WILL PROBABLY NEVER END UP GETTING USED. DELETE ANY CODE THAT ISN'T BEING UTILIZED TODAY, EVEN IF YOU "MIGHT USE IT LATER."

CREATE CLASSES FOR CURRENT REQUIREMENTS ONLY

LEARN MORE CLEAN CODE TIPS AT PRAGMATICWAYS.COM

PRAGMATIC WAYS
TIPS FOR CLEANER CODE

CONDITIONAL COMPLEXITY

WE'VE ALL SEEN IT, THE LARGE CONDITIONAL BLOCKS OF 10+ IF-ELSE STATEMENTS.

THESE COMPLEX CONDITIONALS ARE USUALLY A RESULT OF POOR (OR JUST LACK OF) DESIGN PLANNING, OR JUST NATURALLY GREW WITH THE LIFE OF THE PROJECT (NEW REQUIREMENTS, FEATURES, ETC.)

CONSIDER REFACTORING WITH A DESIGN PATTERN

LEARN MORE CLEAN CODE TIPS AT PRAGMATICWAYS.COM

PRAGMATIC WAYS
TIPS FOR CLEANER CODE

SOLUTION SPRAWL

"YOU BECOME AWARE OF THIS SMELL WHEN ADDING OR UPDATING A SYSTEM FEATURE CAUSES YOU TO MAKE CHANGES TO MANY DIFFERENT PIECES OF CODE."

IF THE PROBLEM IS OF SPRAWLING OBJECT CREATION RESPONSIBILITY, THEN REFACTOR USING THE FACTORY DESIGN PATTERN.

REFACTOR CODE TO LIMIT A FEATURE'S EXPOSURE

LEARN MORE CLEAN CODE TIPS AT PRAGMATICWAYS.COM

PRAGMATIC WAYS
TIPS FOR CLEANER CODE

LONG PARAMETER LIST

METHODS WITH LONG PARAMETER LISTS (OVER 4 PARAMETERS LONG) ARE HARDER TO READ AND UNDERSTAND.

THE FUNCTION IS PROBABLY DOING MORE THAN ONE THING, IN WHICH CASE YOU COULD SPLIT THE METHOD UP INTO MULTIPLE TARGETED FUNCTIONS. OTHERWISE CONSIDER GROUPING THE PARAMETERS TOGETHER IN A DATA OBJECT.

USE LESS THAN 4 PARAMETERS PER FUNCTION

LEARN MORE CLEAN CODE TIPS AT PRAGMATICWAYS.COM

PRAGMATIC WAYS
TIPS FOR CLEANER CODE

DEAD CODE

AS JEFF ATWOOD PUTS IT, "RUTHLESSLY DELETE CODE THAT ISN'T BEING USED. THAT'S WHY WE HAVE SOURCE CONTROL SYSTEMS!"

DEAD CODE IN A PROJECT JUST ADDS MORE CONFUSION TO THE CODE BASE, MORE NONSENSE TO MAINTAIN. CLEAN UP OLD CODE AND DELETE DEAD CODE - YOU'LL KNOW YOU BROKE SOMETHING IF YOUR TEST CASES FAIL!

DELETE DEAD CODE, INCLUDING OLD COMMENTED OUT CODE

LEARN MORE CLEAN CODE TIPS AT PRAGMATICWAYS.COM

PRAGMATIC WAYS
TIPS FOR CLEANER CODE

ODDBALL SOLUTION

WHEN YOU HAVE MULTIPLE SOLUTIONS TO THE SAME PROBLEM, YOU HAVE AN ODDBALL SOLUTION.

THERE SHOULD ONLY BE ONE WAY TO SOLVE THE SAME PROBLEM. ASSESS IF THE OTHER SOLUTION(S) ARE ACTUALLY NEEDED. IF NOT, REMOVE THEM. IF SO, CONSIDER IMPLEMENTING THE ADAPTER DESIGN PATTERN TO UNIFY THE INTERFACES.

CREATE ONE AND ONLY ONE WAY TO SOLVE A PROBLEM

LEARN MORE CLEAN CODE TIPS AT PRAGMATICWAYS.COM

PRAGMATIC WAYS
TIPS FOR CLEANER CODE

PRIMITIVE OBSESSION

ADDING PRIMITIVE VARIABLES IS LIKE SNEAKING CHOCOLATE ON A DIET, "JUST ONE LITTLE PIECE WON'T HURT." ONE AFTER THE OTHER, BEFORE YOU KNOW IT, YOU'VE BLOATED THE PROGRAM WITH A MESS OF PRIMITIVE OBSESSION.

INTRODUCE A PARAMETER OBJECT OR CREATE A WHOLE OBJECT TO CLEAN UP THE PRIMITIVE OBSESSION IN YOUR PROJECT.

CREATE OBJECTS TO REPRESENT DATA

LEARN MORE CLEAN CODE TIPS AT PRAGMATICWAYS.COM

https://pragmaticways.com/31-code-smells-you-must-know/

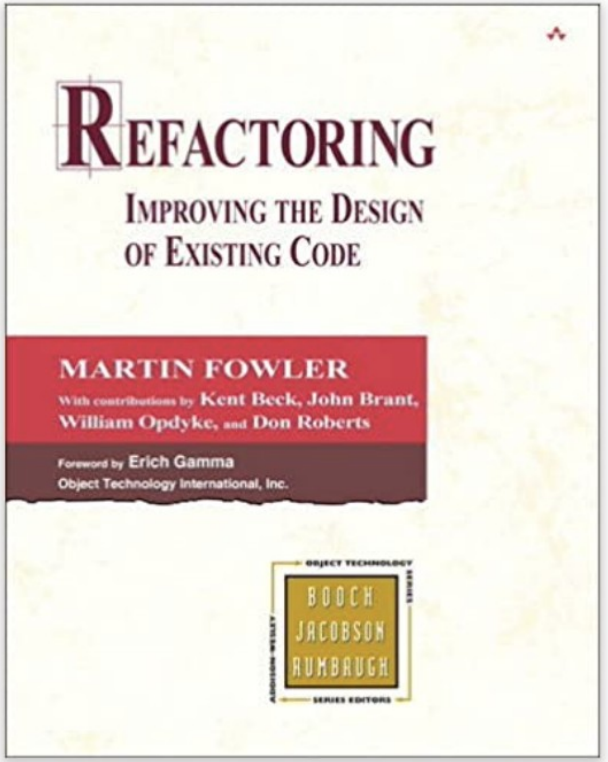
3 ½ . Avoid “code smells” e.g. nested `if`’s

don’t

```
function getSign(x) {
  result = NULL;
  if (x == 0)
    result = “zero”;
  else {
    if (x > 0)
      result = “positive”;
    else {
      if (x < 0)
        result = “negative”
      else
        result = “NaN”;
    }
  }
  return result;
}
```

do

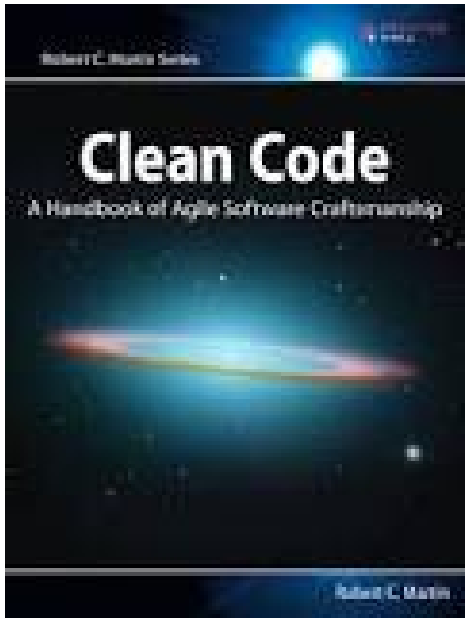
```
function getSign(x) {
    if (x == 0)
        return “zero”;
    if (x > 0)
        return “positive”;
    if (x == 0)
        return “negative”;
    return “Nan”;
}
```



https://refactoring.com/catalog/replaceNestedConditionalWithGuardClauses.html

4. Follow good coding principles

- Don't repeat yourself (DRY)
- Keep it simple (KISS)
- One level of abstraction
- Single responsibility principle
- Separation of concern
- Avoid premature optimization
- Follow style guidelines
- 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!

Clean Code: A Handbook of Agile Software Craftsmanship, R. C. Martin, Prentice Hall, 2008

4. Follow good coding principles and style

stackoverflow

Search…

Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

Teams
Q&A for work

Learn More

What is the "-->" operator in C++?

7883

1831

After reading Hidden Features and Dark Corners of C++/STL on `comp.lang.c++.moderated`, I was completely surprised that the following snippet compiled and worked in both Visual Studio 2008 and G++ 4.4.

Here's the code:

```
#include <stdio.h>
int main()
{
    int x = 10;
    while (x --> 0) // x goes to 0
    {
        printf("%d ", x);
    }
}
```

I'd assume this is C, since it works in GCC as well. Where is this defined in the standard, and where has it come from?

https://stackoverflow.com/questions/1642028/what-is-the-operator-in-c

4. Follow good coding principles and style

stackoverflow

Search...

Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

Teams
Q&A for work

Learn More

What is the "-->" operator in C++?

7883

1831

After reading Hidden Features and Dark Corners of C++/STL on `comp.lang.c++.moderated`, I was completely surprised that the following snippet compiled and worked in both Visual Studio 2008 and G++ 4.4.

Here's the code:

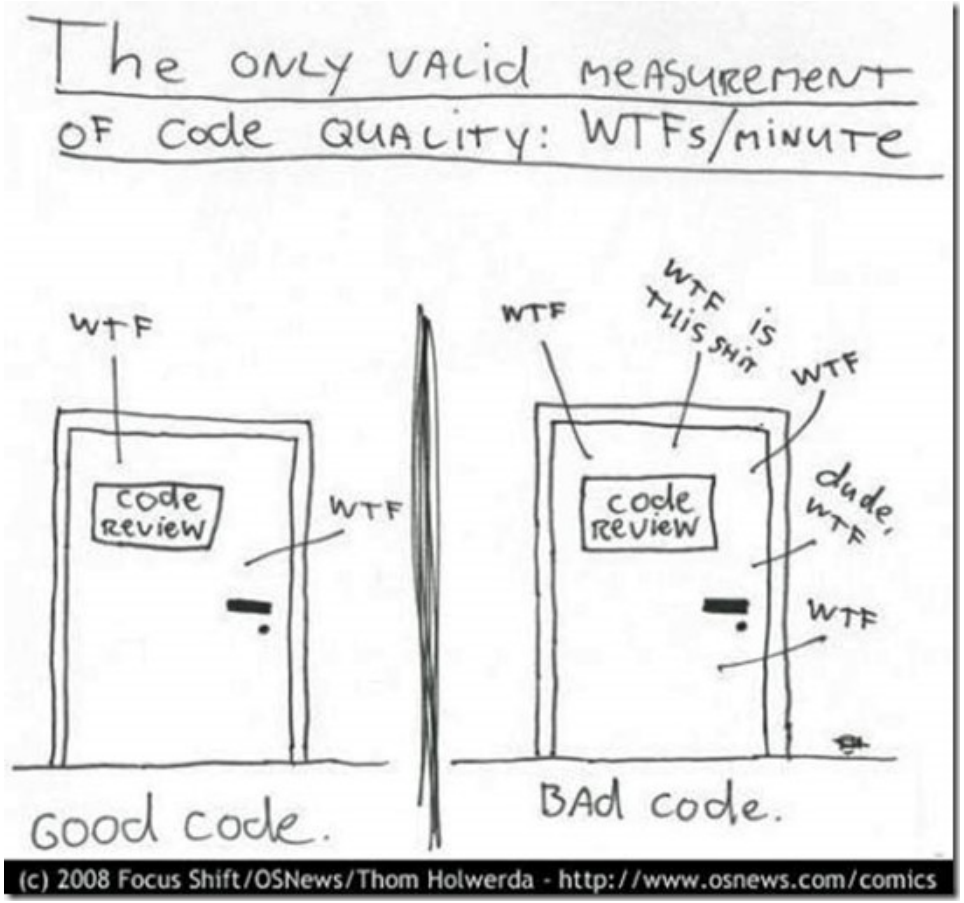
```
while(x-- > 0)
```

```
#include <stdio.h>
int main()
{
    int x = 10;
    while (x --> 0) // x goes to 0
    {
        printf("%d ", x);
    }
}
```

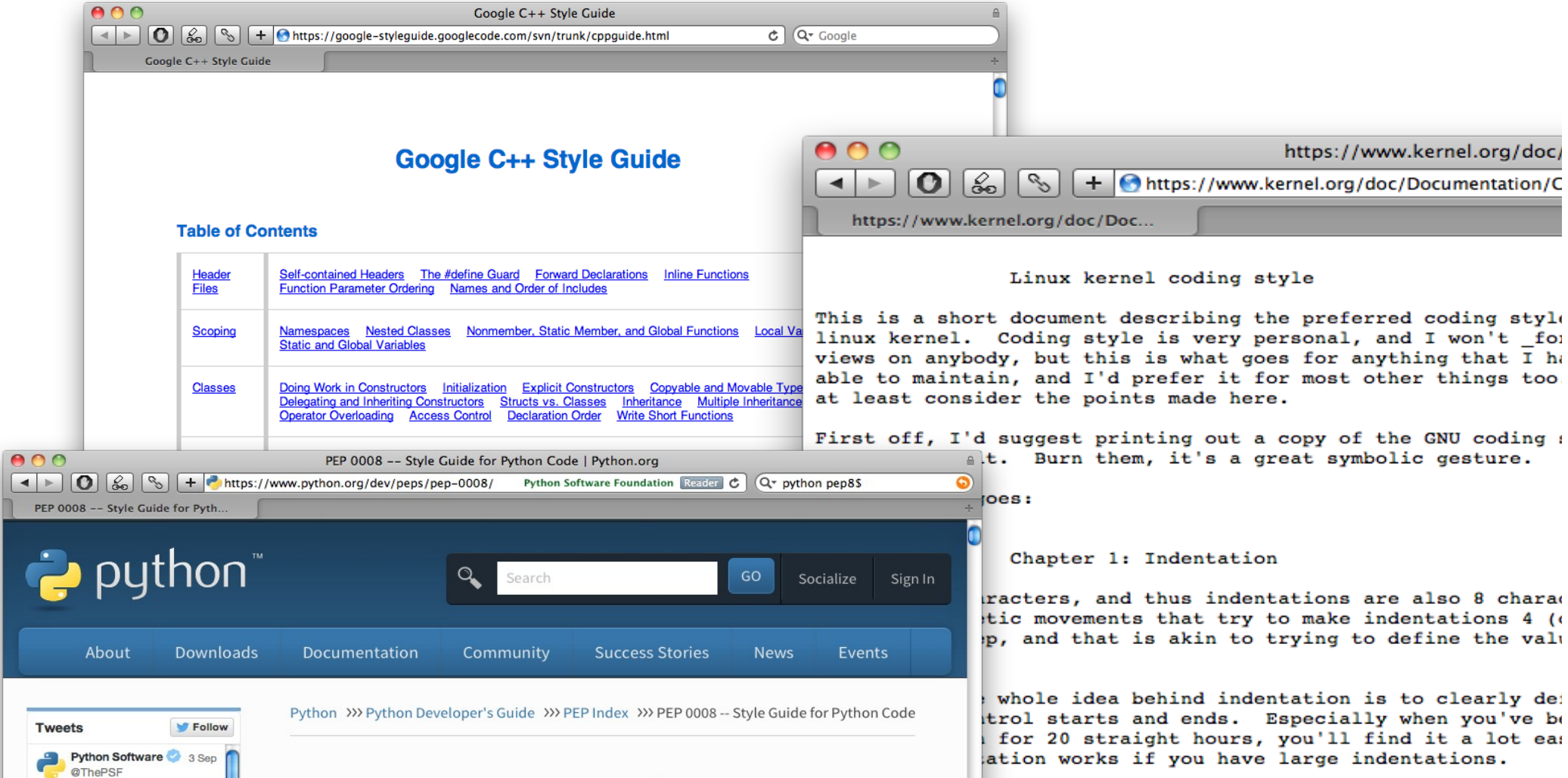
I'd assume this is C, since it works in GCC as well. Where is this defined in the standard, and where has it come from?

https://stackoverflow.com/questions/1642028/what-is-the-operator-in-c

4. Follow good coding principles and style



4. Follow good coding principles and style



4. Follow good coding principles : gracefully handle user errors

Use error handling techniques:

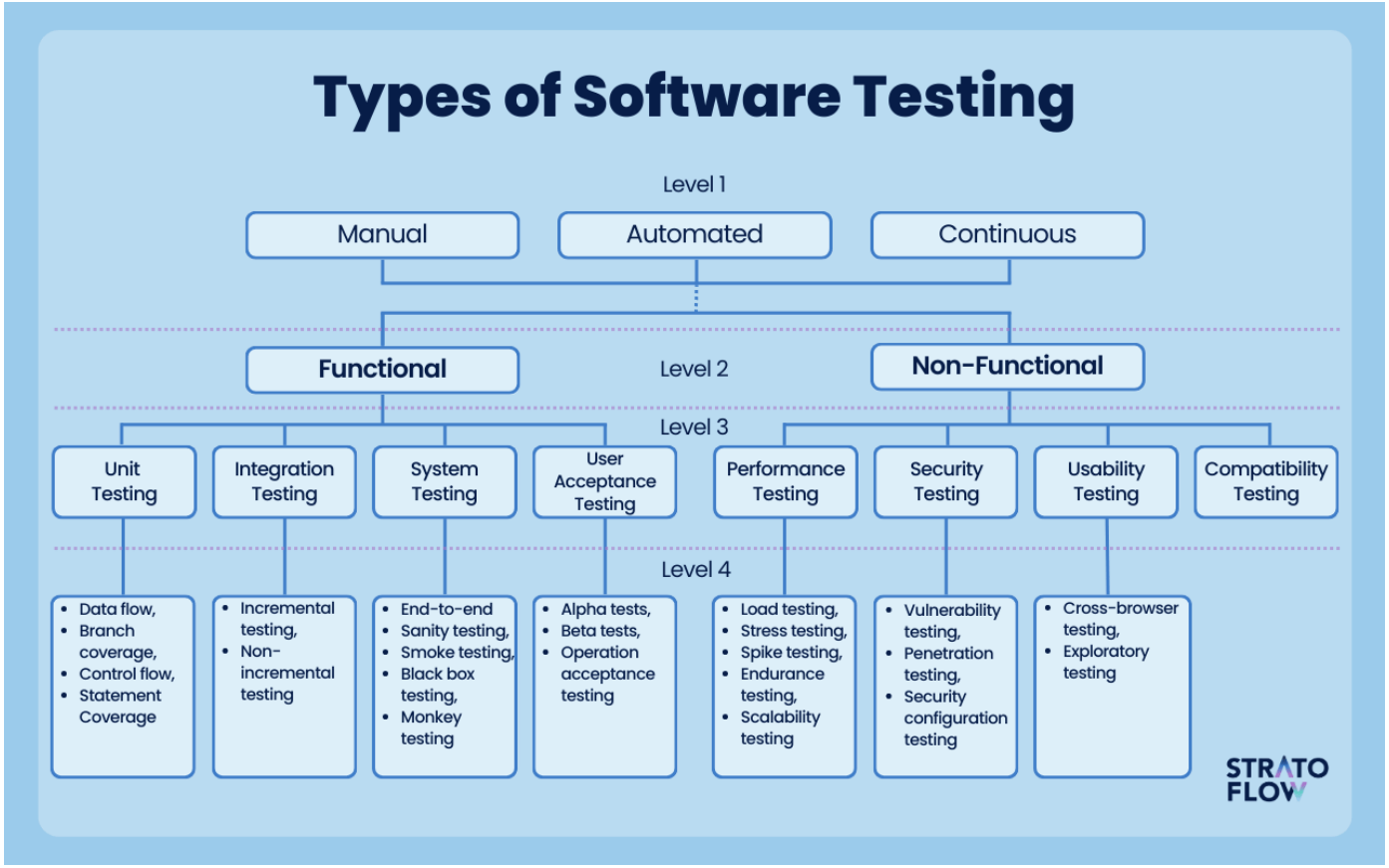
- Return codes
- Exceptions
- Callbacks
- “Result” structs

Be informative in the error messages.

- “`r no good`” ***vs*** “`Error: Input argument Rate (r) must be positive`”
- Grade errors: “Warning”, “Error”, “Fatal”

https://andreabergia.com/blog/2023/05/error-handling-patterns/

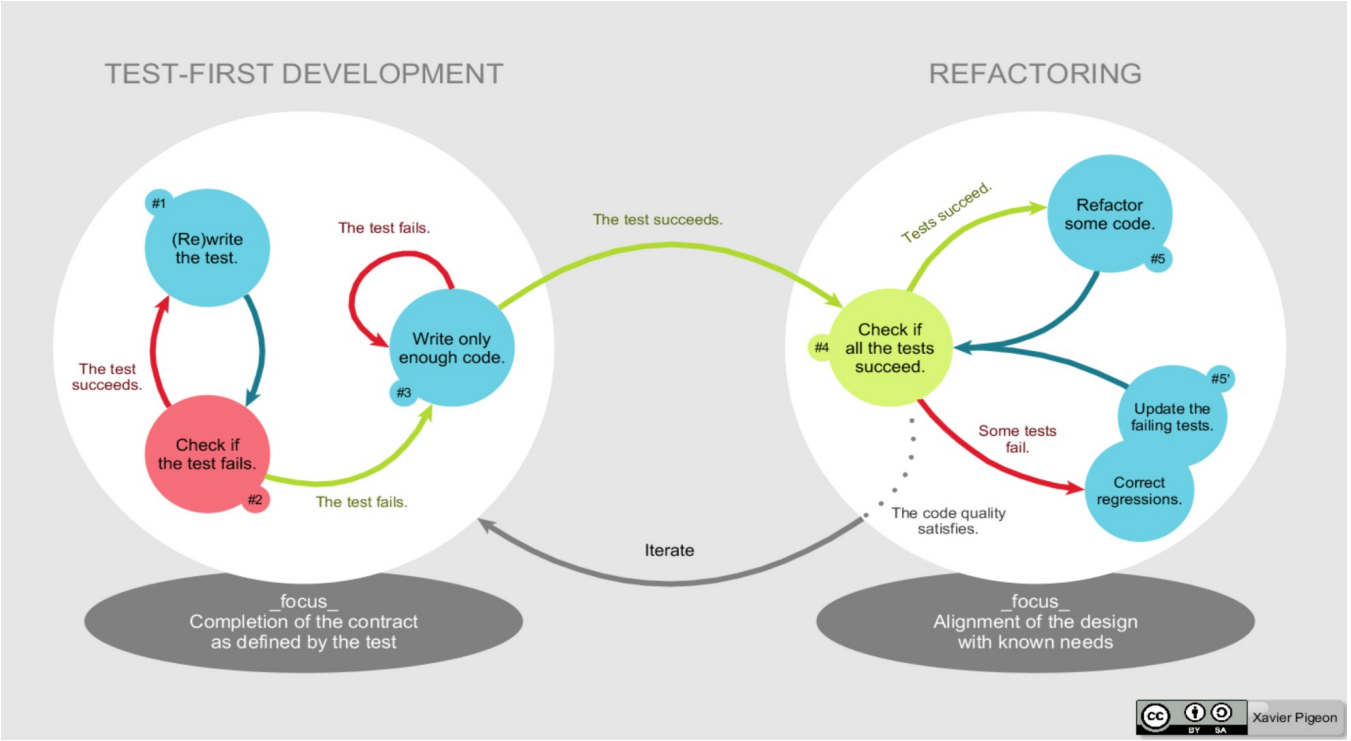
5. Plan for mistakes, automate testing; Test-driven development



https://stratoflow.com/types-of-software-testing/

5. Plan for mistakes, automate testing; Test-driven development

Write the tests before you even write the code



https://en.wikipedia.org/wiki/Test-driven_development

6. Use modern source-code management system

for your code, papers, thesis, etc.

2021

Introduction to code versioning

by Dr Olivier Mattelaer (UCLouvain/CISM)

Wednesday 27 Oct 2021, 09:00 → 12:00 Europe/Brussels

comodal (louvain-la-neuve or remote)

Description

Code versioning is very important to master, even for non programmers. It allows tracking the changes made to a submission script, a piece of code, a configuration file, or event a dataset and propagate the changes in a consistent and systematic way to all clusters.

Contents:

- Notions of code versioning
- Working as a team with code versioning
- Using git to access code from others
- Publishing code

Prerequisite:

- Being able to use SSH with private keys
- Being familiar with a text editor
- Mastering the Linux command line and the GNU utilities (mkdir, cp, scp, etc.)

Type: Hands-on
Target audience: Rookie programmer
Must: This session is a must-have for anyone not familiar with code versioning or git.

Organized by UCLouvain/CISM

Registration Participants 11 / 60 Register

Contact egs-cism@listes.uclouvain.be
0494424767

7. Document the purpose and design, not the implementation


```
function res = f(base, num)
% Assign base to res
res = base
% loop from 2 to num
for i=2:num
    % multiply current res by base
    res=base*res;
end
```

vs

```
function res = pow(base, num)
% compute base^num by iterative multiply for baseline check
res = base
for i=2:num
    res=res*base;
end
```

https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1006561

7. Document the purpose and design, not the implementation

Learn Markdown

```
Super software
============

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
enim ad minim veniam, ...

Subtitle
----------

Here is a list:

  - item 1
  - item 2

And a [link](http://www.google.com) as well.

Some code:

    #!/bin/bash
```

Super software

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, ...

Subtitle

Here is a list:

- item 1
- item 2

And a link as well.

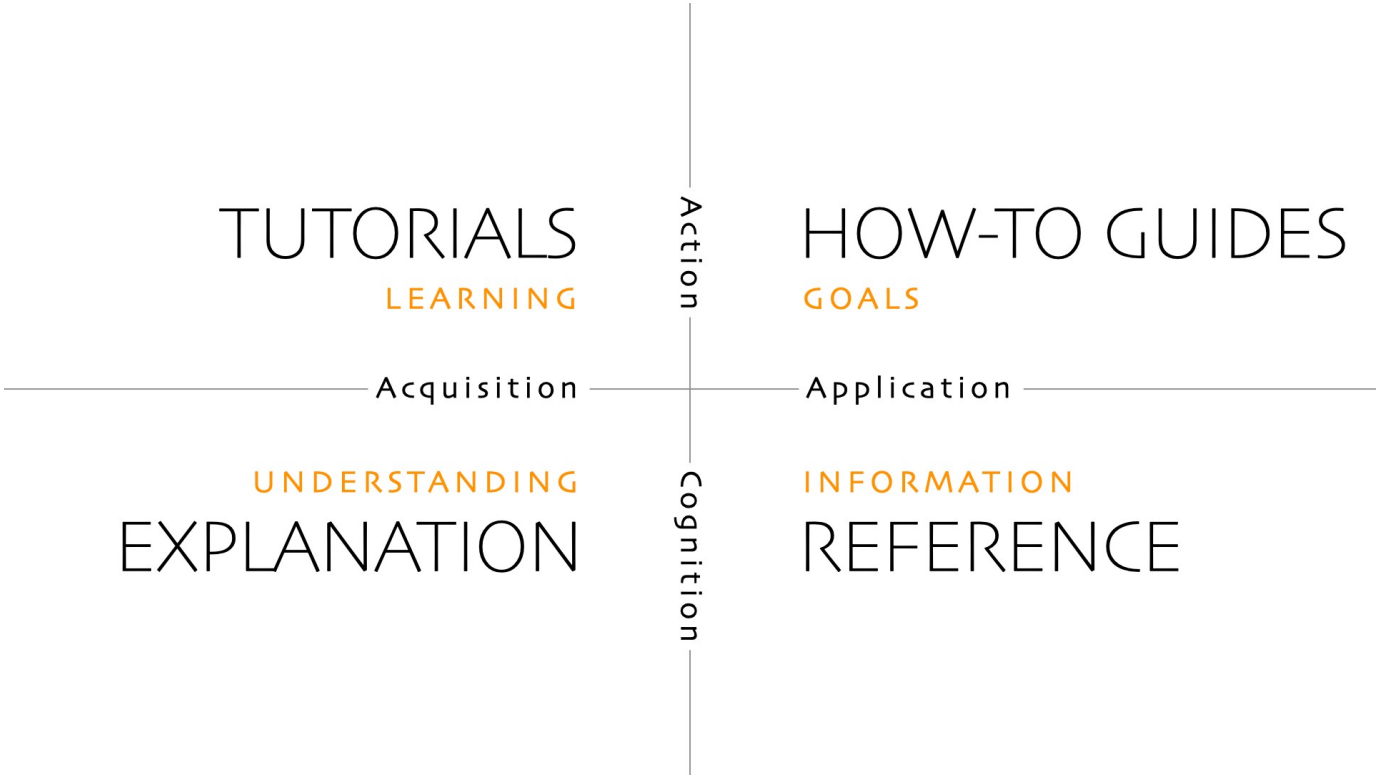
Some code:

```
#!/bin/bash
echo OK
```

https://daringfireball.net/projects/markdown/

7. Document the purpose and design, not the implementation

Try not to mix documentation types



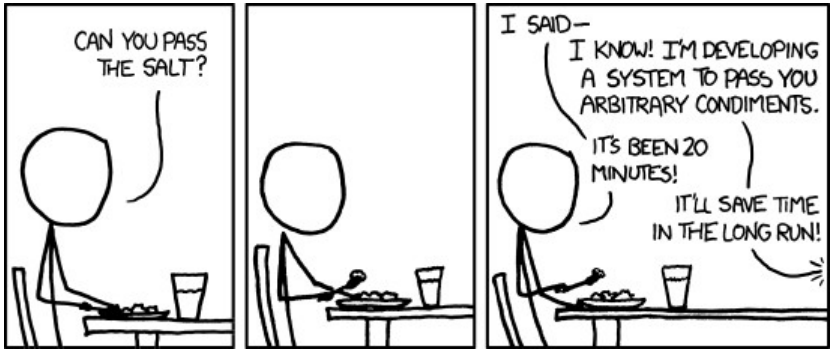
https://diataxis.fr

8. Optimize only when it works already

- Do not try to make it fast when it is not working yet

  (focus on data structures, organization, etc. rather than on micro-optimizations)

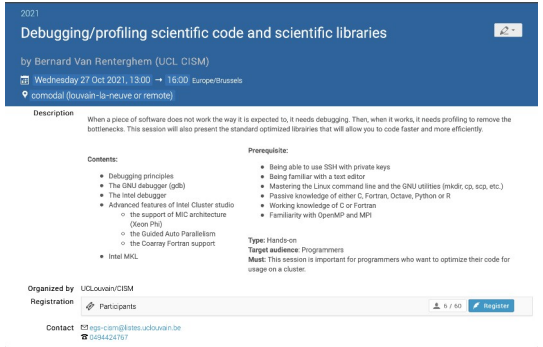
- Do not try to make it universal for all possible future needs at the beginning “YAGNI” (do not close doors either)



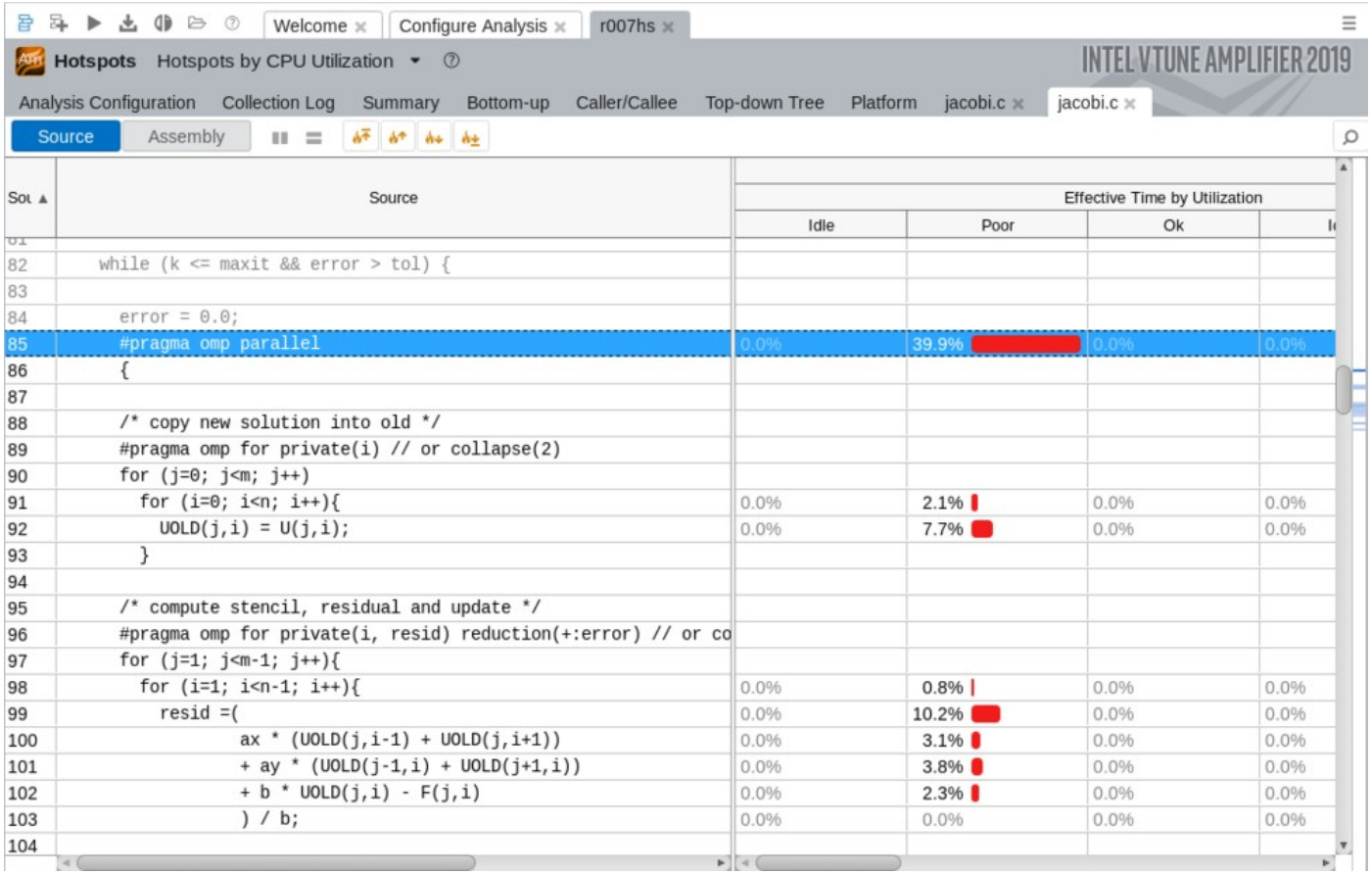
https://xkcd.com/974/

8. Optimize only when it works already

Use a profiler

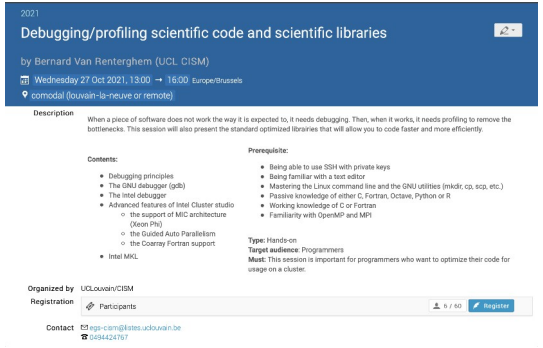


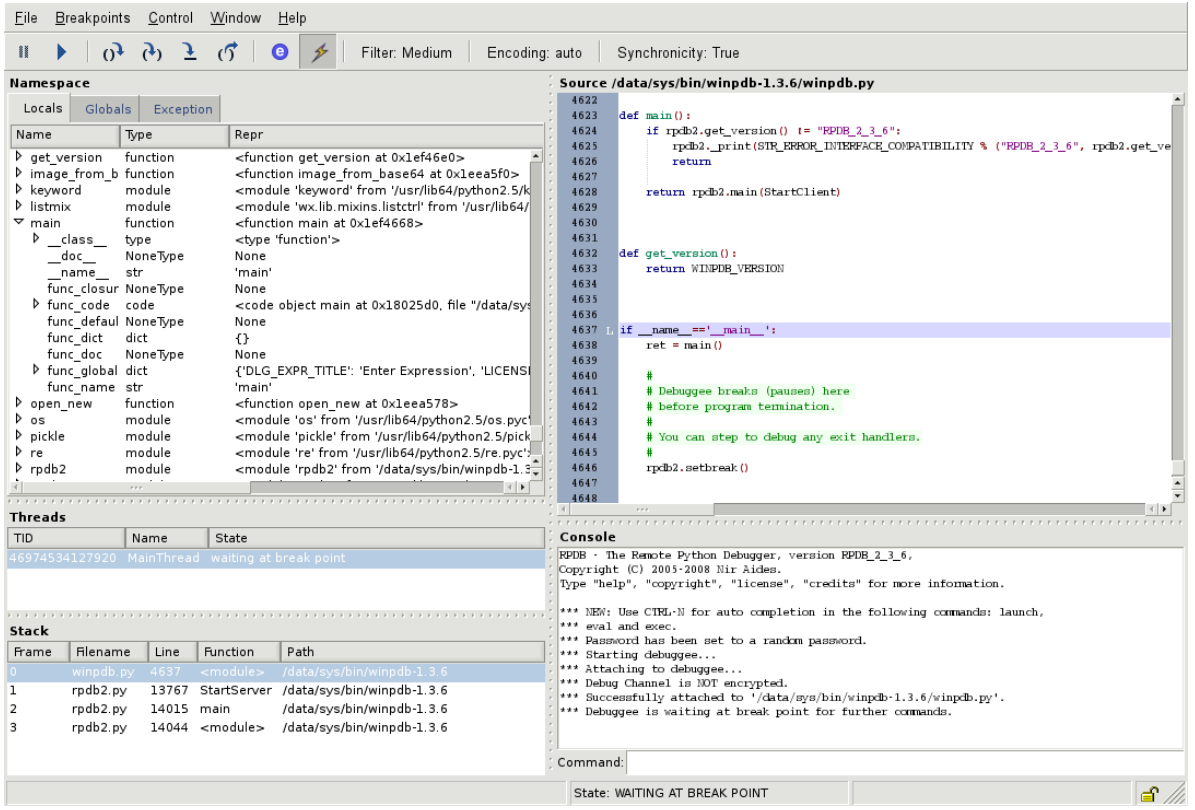
Incorporate benchmarks in your tests



9. Debug cleverly

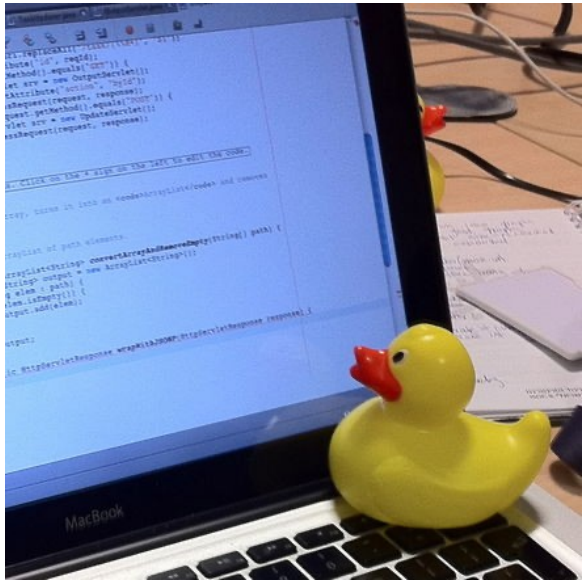
Use a debugger





9. Debug cleverly

Use a method



Describe out loud to an imaginary rubber duck (or a willing colleague) each line in your code in simple terms and why it is obviously correct.

At some point, if you get hesitant, that is probably where the bug is!

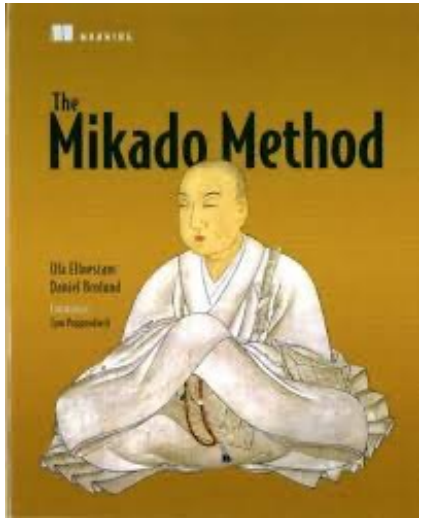
Dig that part of the code until you are confident again that it works.

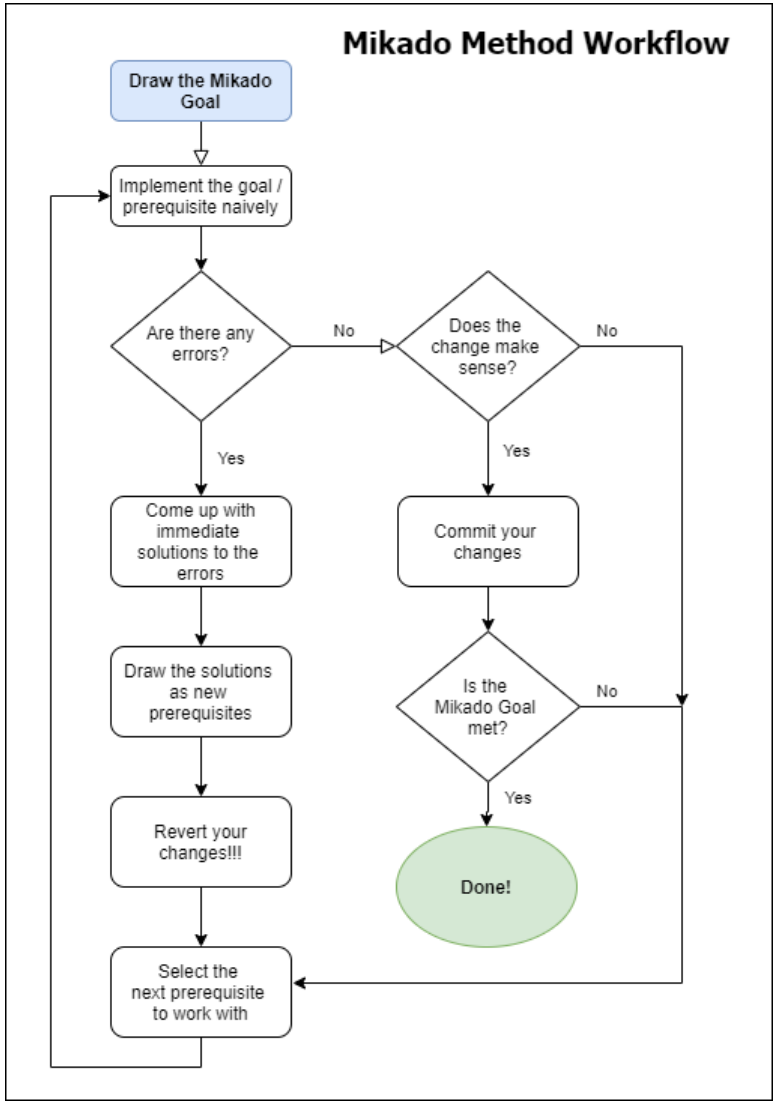
Or discover that it does actually not work as expected ...

https://en.wikipedia.org/wiki/Rubber_duck_debugging

9. Debug cleverly

Use a method





The Mikado Method, O. Ellnestam & D. Brolund, Manning, 2014

“… to make them … **shareable** …, *efficiently*”

1. Automate the compiling process
2. Learn about containers
3. License your code

1. Automate the compiling process

CMake

About Resources Developer Resources Download

CMake 3.3.2 Released

Learn more

CMake News & Blogs

10.06.2015 CMake 3.4.0-rc1 is now ready!

10.02.2015 Automated Tests on GitHub for your ITK-dependent project with Cir...

09.24.2015 Kitware at SciPy 2015

Making sure it compiles on your laptop is not enough

Article Talk Read Edit View history Search

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)*

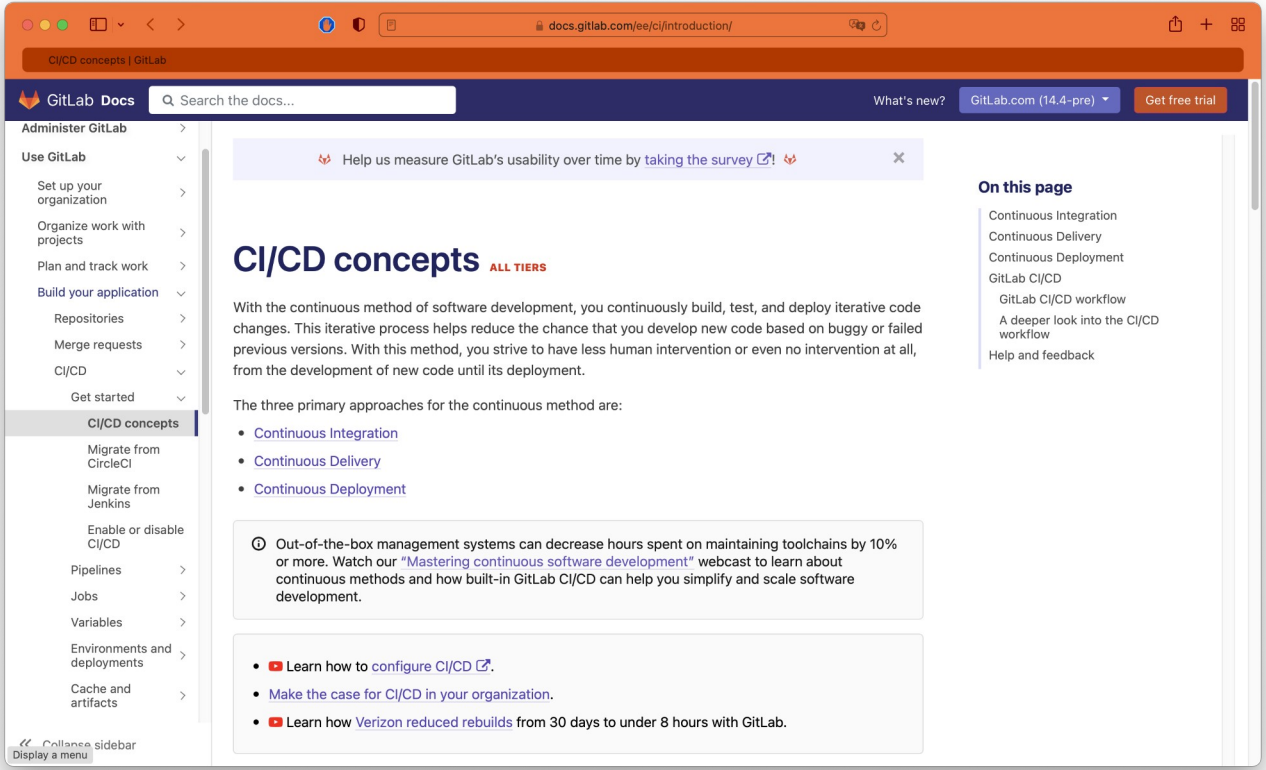
The **GNU build system**, also known as the **Autotools**, is a suite of programming tools designed to assist in making source code packages portable to many Unix-like systems.

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 manageably.

Autotools is part of the GNU toolchain and is widely used in many free software and open

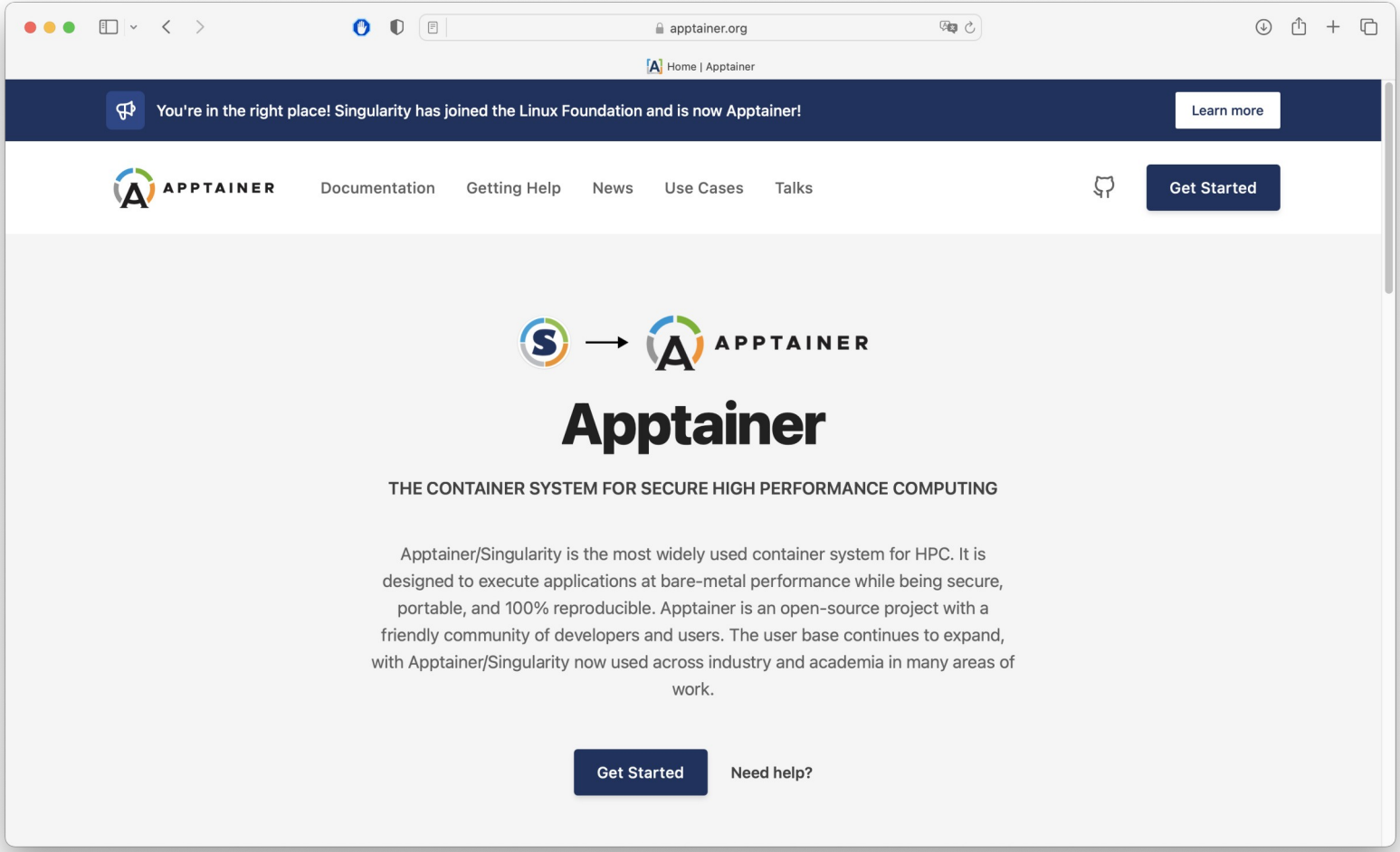
It has to compile on all the clusters...

1. Automate the compiling process



https://docs.gitlab.com/ee/ci/introduction/

2. Learn about containers



3. License your code: Why?

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

- **Scientific 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

3. License your code: e.g. MIT

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 Softwarre.

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.

Can

- Commercial Use
- Modify
- Distribute
- Sublicense
- Private Use

Cannot

- Hold Liable

Must

- Include Copyright
- Include License

3. License your code: e.g. BSD, GPL

BSD

Can	Cannot	Must
Commercial Use	Use Trademark	Include Copyright
Modify	Hold Liable	Include License
Distribute		
Place Warranty		

GPL

Can	Cannot	Must
Commercial Use	Sublicense	Include Original
Modify	Hold Liable	State Changes
Distribute		Disclose Source
Place Warranty		Include License
Use Patent Claims		Include Copyright
		Include Install Instructions

3. License your code: finding help

LIEU – Network of Knowledge Transfer offices

Coordinator

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
TECHNOLOGY TRANSFER OFFICE (LTTO)

Sébastien ADAM
✆ +32 10 47 24 43
sebastien.adam@uclouvain.be

UNIVERSITÉ DE NAMUR
TECHNOLOGY TRANSFER OFFICE

Nathalie FABRY
✆ +32 81 72 53 36
nathalie.fabry@unamur.be

UNIVERSITÉ DE LIÈGE
TECHNOLOGY TRANSFER OFFICE

Jérémie FAYS
✆ + 32 4 349 85 21
j.fays@uliege.be

UNIVERSITÉ DE LIÈGE
TECHNOLOGY TRANSFER OFFICE

Thi Tuyet Minh NGUYEN
✆ +32 349 85 10
ttm.nguyen@uliege.be

UNIVERSITÉ LIBRE DE BRUXELLES
TECHNOLOGY TRANSFER OFFICE (ULB-TTO)

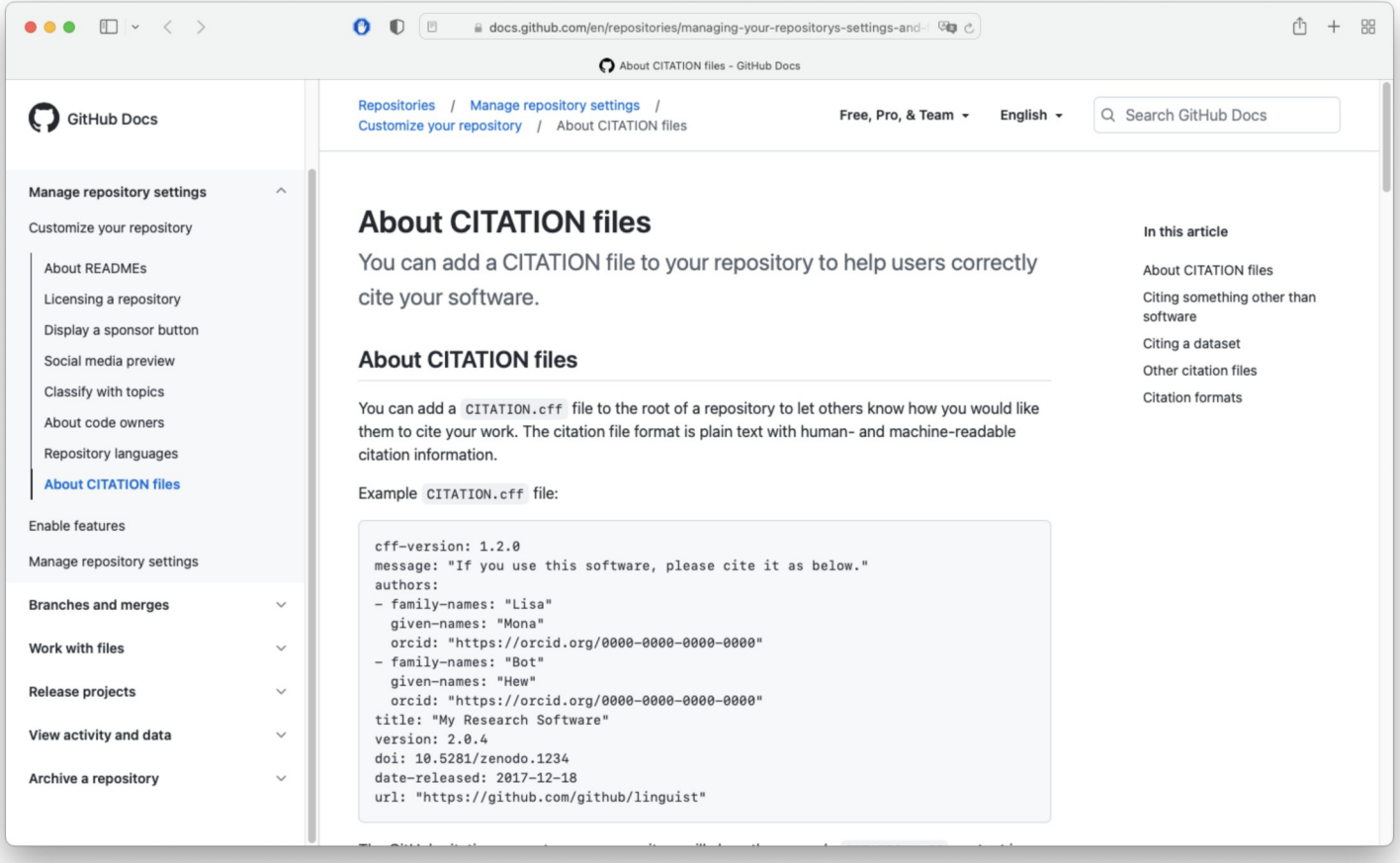
Kévin DEPLUS
✆ +32 650 23 15
kevin.deplus@ulb.ac.be

UNIVERSITÉ DE MONS
TECHNOLOGY TRANSFER OFFICE

Sandrine BROGNAUX
✆ +32 65 37 47 97
sandrine.brognaux@umons.ac.be

Slide credit: Sébastien ADAM

3 ½. GitHub CITATION files

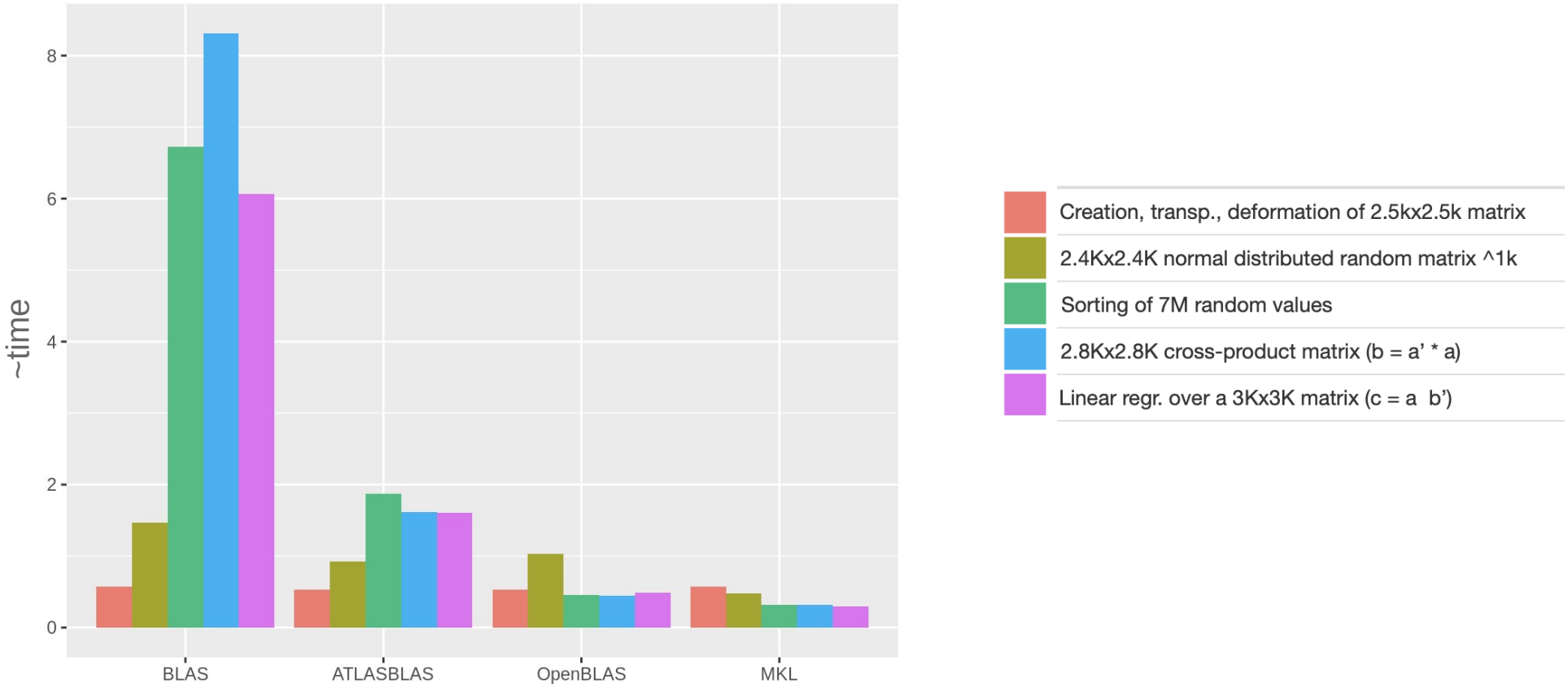


https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/about-citation-files

“… to make them … **fast** …, *efficiently*”

1. Use optimized libraries
2. Choose the right storage system
3. Think parallel from the start
4. Integrate checkpoint/restart from the start

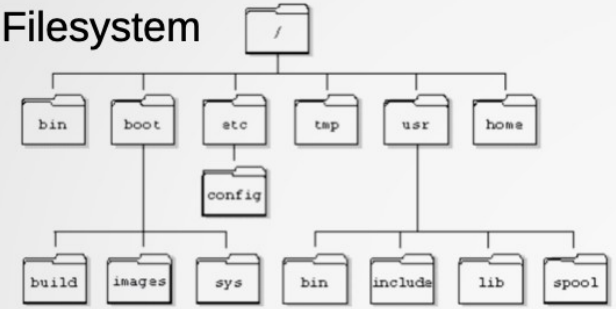
1. Use optimized libraries



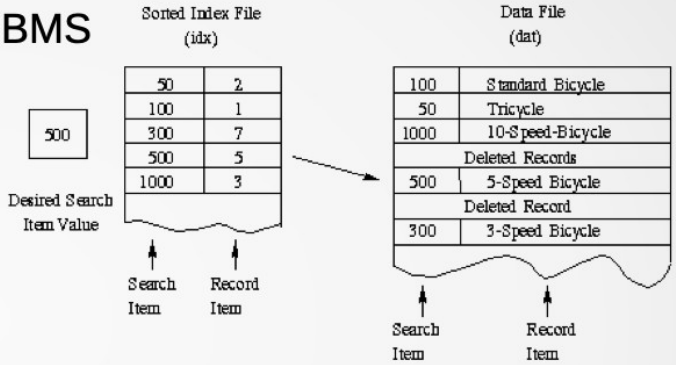
https://csantill.github.io/RPerformanceWBLAS/

2. Choose the right storage system

Filesystem

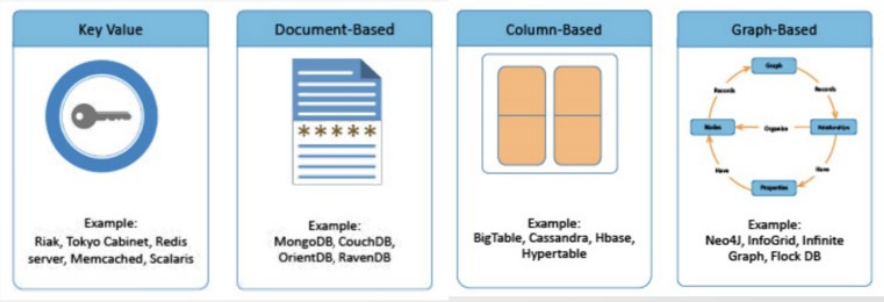


RDBMS



Objects store

NoSQL



3. Think parallel from the start

1. Identify data flows and independent tasks
2. Make data decomposition easy
3. Make work decomposition easy

```
data = [(x1,y1), (x2,y2), .., (x10,y10)]
for i = 1 .. 10
  s[i] = ( data[xi]  + data[yi] )
  ss[i] = ( data[xi]^2  + data[yi]^2 )
end
```

```
begin=0, end=10
data = [(x1,y1), (x2,y2), .., (x10,y10)]
for i = begin .. end
  s[i] = ( data[xi]  + data[yi] )
end
for i = begin .. end
  ss[i] = ( data[xi]^2  + data[yi]^2 )
end
```

4. Integrate checkpoint/restart from the start

1. Allow starting from a non-initial state
2. Save variables to disk frequently

```
data = [(x1,y1), (x2,y2), .., (x10,y10)]
for i = 1 .. 10
  res[i] = ( data[xi]^2  + data[yi]^2 )
end
```

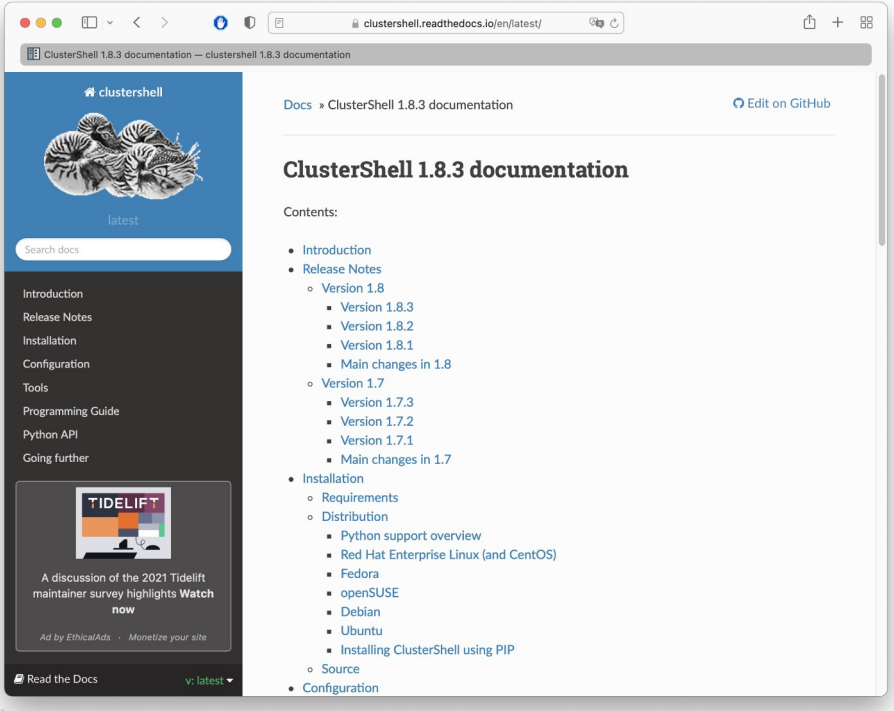
```
if exists(i) and exists(res)
    begin=load(i)
    res=load(res)
else
    begin = 1
end=10
data = [(x1,y1), (x2,y2), .., (x10,y10)]
for i = begin .. end
  res[i] = ( data[xi]^2  + data[yi]^2 )
  save(res, i)
end
```

“ …, *efficiently*”

1. Perform “multi-host” SSH
2. Master configuration management
3. Use terminal multiplexing
4. Install software like a boss
5. Avoid the boilerplate
6. BACKUPS!

1. Perform “multi-host” SSH

```
❯ clush -Bw lemaitre3,hercules,nic5,dragon2 "emacs --version"
---------------
dragon2,hercules,lemaitre3 (3)
---------------
GNU Emacs 24.3.1
Copyright (C) 2013 Free Software Foundation, Inc.
GNU Emacs comes with ABSOLUTELY NO WARRANTY.
You may redistribute copies of Emacs
under the terms of the GNU General Public License.
For more information about these matters, see the file named COPYING.
---------------
nic5
---------------
GNU Emacs 26.1
Copyright (C) 2018 Free Software Foundation, Inc.
GNU Emacs comes with ABSOLUTELY NO WARRANTY.
You may redistribute copies of GNU Emacs
under the terms of the GNU General Public License.
For more information about these matters, see the file named COPYING.
❯ clush -Bw lemaitre3,hercules,nic5,dragon2 "scontrol version"
---------------
dragon2,lemaitre3 (2)
---------------
slurm 20.02.7
---------------
hercules
---------------
slurm 20.02.6
---------------
nic5
---------------
slurm 20.02.3
❯ clush -w lemaitre3,hercules,nic5,dragon2 "squeue -tPD | wc -l"
nic5: 1420
lemaitre3: 288
dragon2: 145
hercules: 102
❯
```



https://clustershell.readthedocs.io/en/latest/

2. Master configuration management

```
❯ ansible -i lemaitre3,nic5 'all' -m lineinfile -a "dest=myfile line='Contents' create=true"
nic5 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "backup": "",
    "changed": true,
    "msg": "line added"
}
lemaitre3 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "backup": "",
    "changed": false,
    "msg": ""
}
```

2. Master configuration management

```
❯ cat inventory playbook.yml myfile
```

File: **inventory**

```
[all]

lemaitre3 short_name="lm3"
nic5      short_name="nic5"
```

File: **playbook.yml**

```
---
- hosts:
    - lemaitre3
    - nic5
  tasks:
    - name: Upload templated file
      template: src=myfile dest=. mode=700
```

File: **myfile**

```
This cluster's short name is {{ short_name }}
```

https://

2. Master configuration management

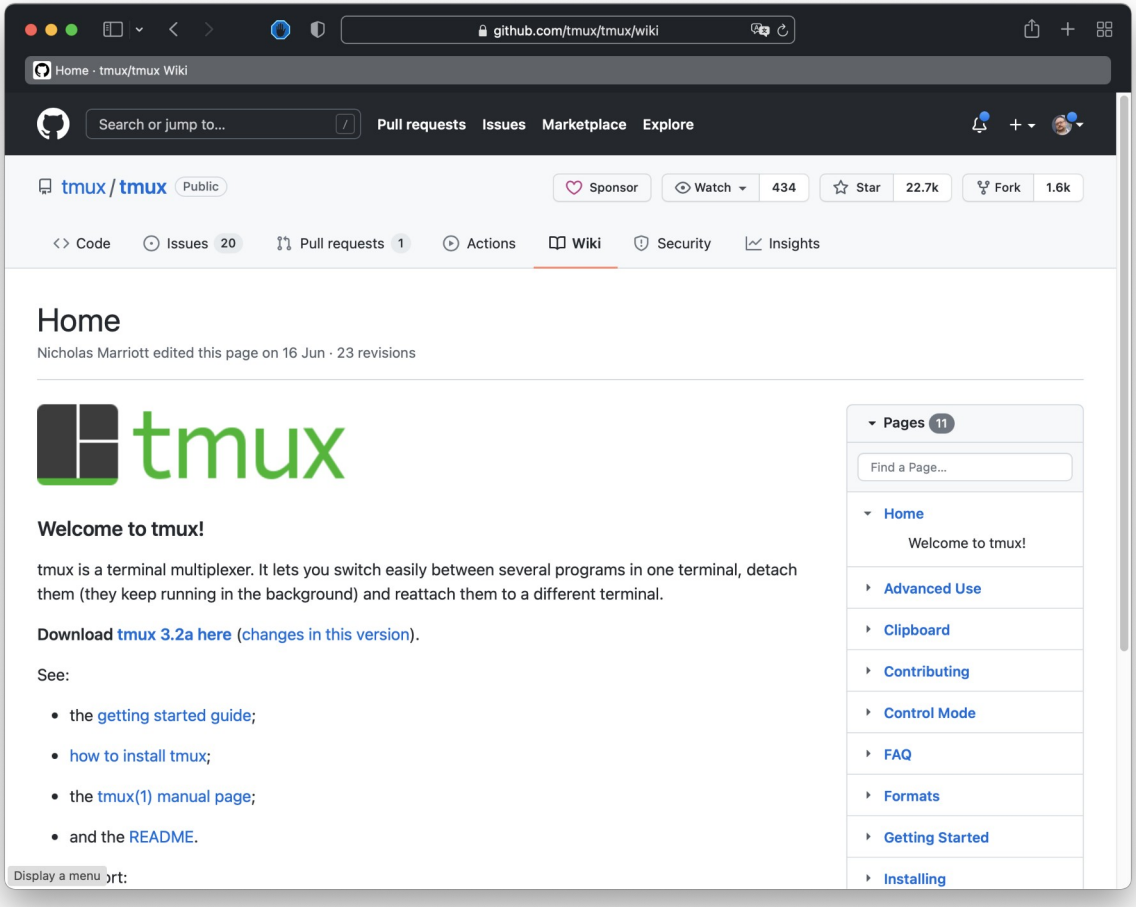
```
❯ ansible-playbook -i inventory playbook.yml --diff

PLAY [lemaitre3,nic5] ***********************************************************************************

TASK [Gathering Facts] **********************************************************************************
ok: [nic5]
ok: [lemaitre3]

TASK [Upload templated file] ****************************************************************************
--- before: ./myfile
+++ after: /Users/dfrancois/.ansible/tmp/ansible-local-40594k4ng0q9q/tmpd689k7ap/myfile
@@ -1 +1 @@
-Contents
+This cluster's short name is nic5

changed: [nic5]
--- before: ./myfile
+++ after: /Users/dfrancois/.ansible/tmp/ansible-local-40594k4ng0q9q/tmpa5narr9m/myfile
@@ -1 +1 @@
-Contents
+This cluster's short name is lm3

changed: [lemaitre3]

PLAY RECAP ******************************************************************************************************
lemaitre3                  : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
nic5                       : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

❯ clush -w lemaitre3,nic5 "cat myfile"
nic5: This cluster's short name is nic5
lemaitre3: This cluster's short name is lm3
```

https://

3. Use terminal multiplexing

Do not let SSH disconnections harm your workflow

(and much more)



https://github.com/tmux/tmux/wiki

4. Install software like a boss

```
[dfr@lemaitre3 ~]$ eb --search emacs
== found valid index for /usr/easybuild/easyconfigs, so using it...
== found valid index for /usr/easybuild/easyconfigs, so using it...
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-24.3-GCC-4.8.3-bare.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-24.3-GCC-4.8.3.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-24.4-GCC-4.9.2.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-24.5-GCC-4.9.3-2.25.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-25.1-foss-2016a.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-25.3-GCCcore-6.3.0.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-25.3-GCCcore-6.4.0.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-25.3-GCCcore-7.3.0.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-26.3-GCCcore-8.3.0.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-27.1-GCCcore-9.3.0.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-27.1-GCCcore-10.2.0.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-24.3-GCC-4.8.3-bare.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-24.3-GCC-4.8.3.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-24.4-GCC-4.9.2.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-24.5-GCC-4.9.3-2.25.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-25.1-foss-2016a.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-25.3-GCCcore-6.3.0.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-25.3-GCCcore-6.4.0.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-25.3-GCCcore-7.3.0.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-26.3-GCCcore-8.3.0.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-27.1-GCCcore-9.3.0.eb
 * /usr/easybuild/easyconfigs/e/Emacs/Emacs-27.1-GCCcore-10.2.0.eb
```

easybuild

EasyBuild @PyPi | EasyBuild docs | EasyBuild @GitHub

EasyBuild: building software with ease.

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/.

Getting started

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

https://easybuild.readthedocs.io/en/latest/

4. Install software like a boss (module tips)

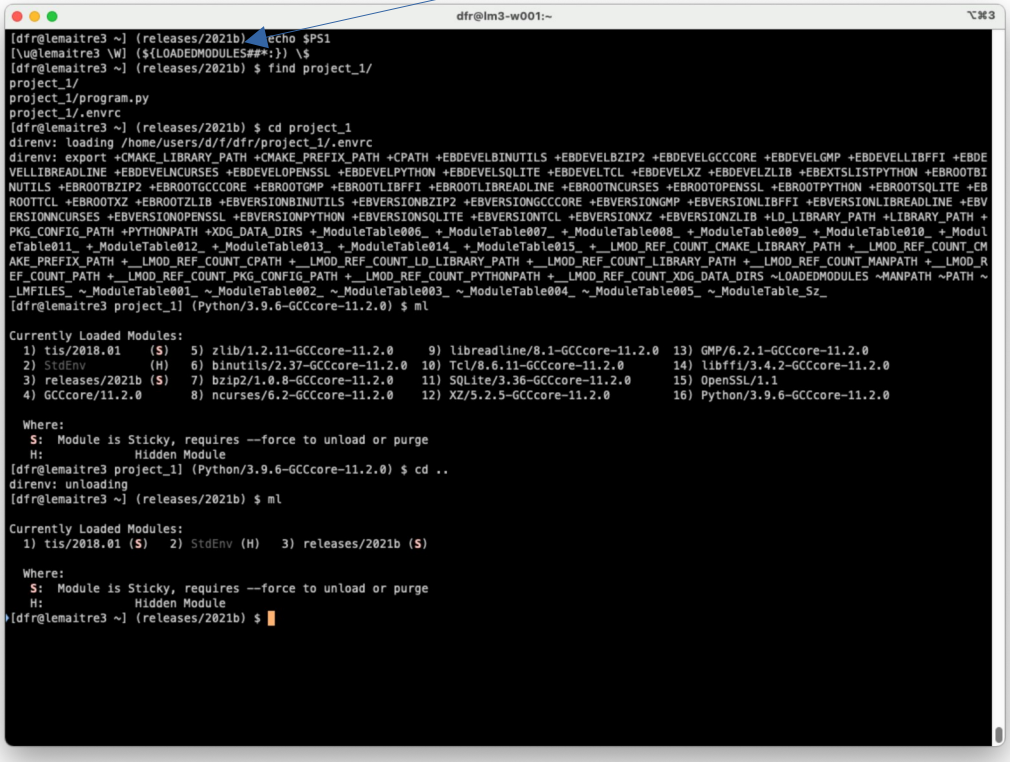
Setup your `$PS1` as `[\u@\h \W] (${LOADEDMODULES##*:}) \$` to see latest loaded module

Use `direnv` to automatically load modules based on the current working directory

Create module collections with `module save`

Write and use your own modules with `module use PATH`

```
help([[Setup  env so that Slurm commands
operate on Debug partition]])


local partition='debug'


setenv("SQUEUE_PARTITION", partition)
setenv("SINFO_PARTITION", partition)
setenv("SBATCH_PARTITION", partition)
setenv("SRUN_PARTITION", partition)
```



https://github.com/direnv/direnv/

5. Avoid the boilerplate

COOKIECUTTER

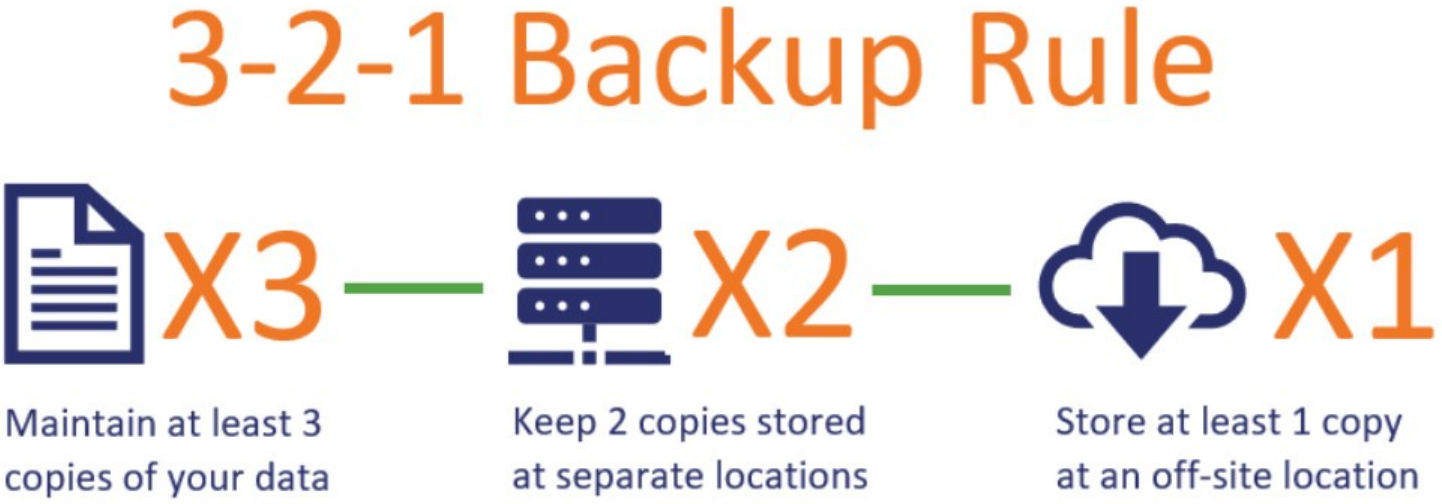
```
{
    "project_name": "project_name",
    "repo_name": "{{ cookiecutter.project_name.lower().replace(' ', '_') }}",
    "author_name": "Your name (or your organization/company/team)",
    "description": "A short description of the project.",
    "open_source_license": ["MIT", "BSD-3-Clause", "No license file"],
    "s3_bucket": "[OPTIONAL] your-bucket-for-syncing-data (do not include 's3://')",
    "aws_profile": "default",
    "python_interpreter": ["python3", "python"]
}
```

→

```
├── LICENSE
├── Makefile           <- Makefile with commands like `make data` or `make train`
├── README.md          <- The top-level README for developers using this project.
├── data
│   ├── external       <- Data from third party sources.
│   ├── interim        <- Intermediate data that has been transformed.
│   ├── processed      <- The final, canonical data sets for modeling.
│   └── raw            <- The original, immutable data dump.
│
├── docs               <- A default Sphinx project; see sphinx-doc.org for details
│
├── models             <- Trained and serialized models, model predictions, or model summaries
│
├── notebooks          <- Jupyter notebooks. Naming convention is a number (for ordering),
│                         the creator's initials, and a short `-` delimited description, e.g.
│                         `1.0-jqp-initial-data-exploration`.
│
├── references         <- Data dictionaries, manuals, and all other explanatory materials.
│
├── reports            <- Generated analysis as HTML, PDF, LaTeX, etc.
│   └── figures        <- Generated graphics and figures to be used in reporting
│
├── requirements.txt   <- The requirements file for reproducing the analysis environment, e.g.
│                         generated with `pip freeze > requirements.txt`
│
├── setup.py           <- makes project pip installable (pip install -e .) so src can be imported
├── src                <- Source code for use in this project.
│   ├── __init__.py    <- Makes src a Python module
│   │
│   ├── data           <- Scripts to download or generate data
│   │   └── make_dataset.py
│   │
│   ├── features       <- Scripts to turn raw data into features for modeling
│   │   └── build_features.py
│   │
│   ├── models         <- Scripts to train models and then use trained models to make
│   │   │                 predictions
│   │   ├── predict_model.py
│   │   └── train_model.py
│   │
│   └── visualization  <- Scripts to create exploratory and results oriented visualizations
│       └── visualize.py
│
└── tox.ini            <- tox file with settings for running tox; see tox.readthedocs.io
```

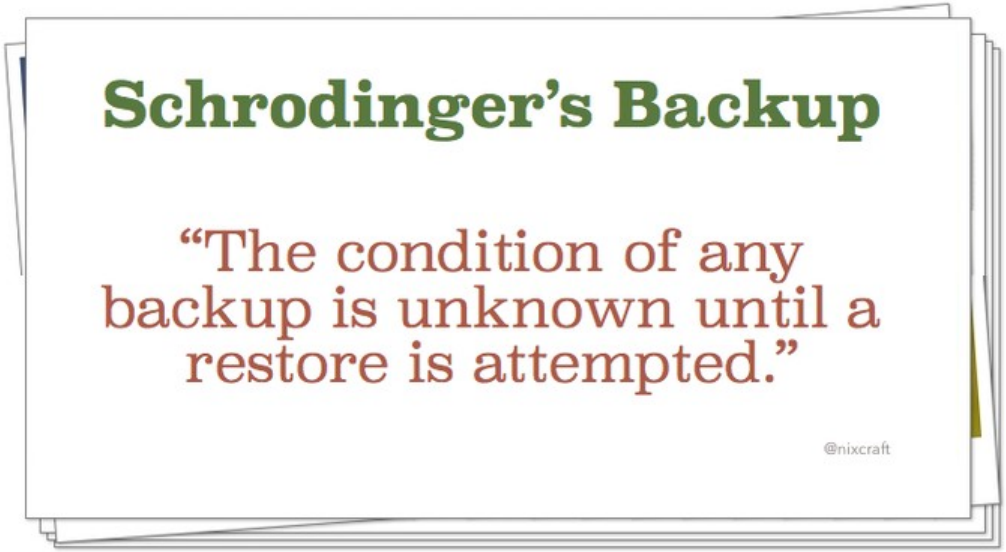
https://cookiecutter.readthedocs.io/en/stable/
https://github.com/search?q=cookiecutter&type=Repositories

6. BACKUPS!!!



https://securityboulevard.com/2020/05/3-2-1-backup-rule-the-rule-of-thumb-to-solve-your-data-loss-problems/

6. BACKUPS!!!


Schrodinger's Backup
"The condition of any backup is unknown until a restore is attempted."
@nixcraft

6. BACKUPS!!!

rsync

RdiffBACKUP

Borg

Kopia

restic

Backups done right!

https://www.borgbackup.org | https://rdiff-backup.net | https://kopia.io | https://restic.net

This was:

“A short catalog of *tools*
the professionals are using for
developing and **deploying** programs,
to make them **correct**, **maintainable**, **shareable**, and **fast**,
efficiently.”

We discussed:

- good practices
- important choices
- useful tools
- practical references

The “Phillip test” (by Philip Guo)

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 got hit by a bus, can one of your lab-mates resume your research where you left off with less than a week of delay?

Work faster & more reliably