Boas Práticas para Computação Científica

Raniere Gaia Costa da Silva[1]

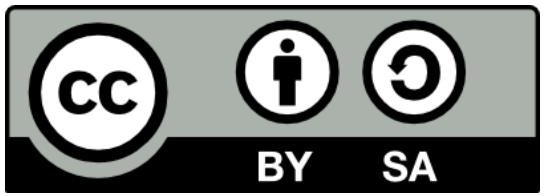
30/11/2012

[1] r.gaia.cs@gmail.com

Esta apresentação é baseada no artigo "Best Practices for Scientific Computing" de D. A. Aruliah *et al.* que encontra-se disponível em `http://arxiv.org/abs/1210.0530v2`.

Os arquivos desta apresentação encontram-se disponíveis em `https://github.com/r-gaia-cs/presentations`.

Licença



Scientists spend an increasing amount of time building and using software. However, most scientists are never taught how to do this efficiently. As a result, many are unaware of tools and practices that would allow them to write more reliable and maintainable code with less effort. We describe a set of best practices for scientific software development that have solid foundations in research and experience, and that improve scientists' productivity and the reliability of their software. (Retirado de [ABH+12].)

***Scientists spend an increasing amount of time building and using software**. However, most scientists are never taught how to do this efficiently. As a result, many are unaware of tools and practices that would allow them to write more reliable and maintainable code with less effort. We describe a set of best practices for scientific software development that have solid foundations in research and experience, and that improve scientists' productivity and the reliability of their software. (Retirado de [ABH+12].)*

None of these practices will guarantee efficient, error-free, software development, but used in concert they will reduce the number of errors in scientific software, make it easier to reuse, and save the authors of the software time and effort that can used for focusing on the underlying scientific questions. (Retirado de [ABH+12].)

***None of these practices will guarantee efficient, error-free, software development**, but used in concert they will reduce the number of errors in scientific software, make it easier to reuse, and save the authors of the software time and effort that can used for focusing on the underlying scientific questions. (Retirado de [ABH+12].)*

O Código

Um código deve:

- executar corretamente e
- ser fácil de ler.

O Código

Um código deve:

- executar corretamente e
- ser fácil de ler.

Para que o código seja fácil de ler deve-se considerar os seguintes aspectos quando este for ser escrito:

- o leitor possui memória e atenção bastante limitada e
- uso involuntário de reconhecimento de padrões.

Nomes

Nomes devem ser consistente, distintos e significativos. Para isso evite nomes não-descritivos e muito similares.

Nomes

Nomes devem ser consistente, distintos e significativos. Para isso evite nomes não-descritivos e muito similares.

Exemplo (Nomes não-descritivos)

`a`, `foo`, `bar`, ...

Nomes

Nomes devem ser consistente, distintos e significativos. Para isso evite nomes não-descritivos e muito similares.

Exemplo (Nomes não-descritivos)

`a`, `foo`, `bar`, ...

Exemplo (Nomes similares)

`resultado`, `resultado1`, `resultados`, ...

Estilos

Deve-se seguir um único estilo de indentação e preferencialmente não misturar `CamelCaseNaming` com `pothole_case_naming`.
Exemplos de estilos podem ser encontrados em:

C++ `http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml`.

Python `http://google-styleguide.googlecode.com/svn/trunk/pyguide.html`.

R `http://google-styleguide.googlecode.com/svn/trunk/google-r-style.html`.

Parâmetros (1)

A ordem dos parâmetros é bastante importante para evitar erros por parte dos usuários.

Parâmetros (1)

A ordem dos parâmetros é bastante importante para evitar erros por parte dos usuários.

Exemplo

Considere a subrotina em Fortran abaixo:

```
subroutine rect_area(x1, x2, y1, y2, a)
    ! This function attribute to `a` the area of the rectangle
    ! with opposite vertices in (x1, x2) and (y1, y2).
    double precision x1, x2, y1, y2, a
    a = abs((x1 - y1) * (x2 - y2))
end subroutine rect_area
```

As seguintes chamadas seriam erradas:

```
call rect_area(x_1, y_1, x_2, y_2, a)
call rect_area(a, x_1, x_2, y_1, y_2)
```

Parâmetros (2)

Via de regra o uso de estruturas/objetos é recomendada.

Parâmetros (2)

Via de regra o uso de estruturas/objetos é recomendada.

Exemplo

Considere a subrotina para o GNU Octave abaixo:

```
function y = sum_vect(a, b)
    % This function return the sum of the vectors a and b.
    n = size(a, 1);
    for i = 1:n
        y(i) = a(i) + b(i);
    end
end
```

A seguinte chamada seria errada:

```
a = [0 0];
b = [1 1];
x = sum_vect(a, b);
```

Evite repetir comandos e clicks (1)

Deve-se evitar repetir comandos e clicks utilizados com muita frequência. Em relação aos comandos, recomenda-se utilizar o GNU Make e/ou alguma linguagem de script, e.g.,

- GNU Bash,
- Python,
- Ruby,
- GNU Octave, ...

E em relação aos clicks, recomenda-se atribuí-los a algum novo botão ou atalho.

Evite repetir comandos e clicks (2)

Exemplo

Considere o código abaixo em C:

```
#include <stdio.h>
int main(){
    printf("Hello.\n");
    return 0;
}
```

Para a tarefa de compilá-lo e executá-lo podemos utilizar o GNU Make por meio do arquivo abaixo:

```
hello: hello.c
    gcc hello.c -o hello
run: hello
    ./hello
```

Evite repetir comandos e clicks (3)

Exemplo (Continuação)

E por fim, procedemos com

```
$ make run
gcc hello.c -o hello
./hello
Hello.
```

Evite repetir comandos e clicks (3)

Exemplo (Continuação)

E por fim, procedemos com

```
$ make run
gcc hello.c -o hello
./hello
Hello.
$ make run
./hello
Hello.
```

Use o histórico

[. . .] most command-line tools have a "history" option that lets users display and re-execute recent commands, with minor edits to filenames or parameters. This is often cited as ***one reason command-line interfaces remain popular****. (Retirado de [ABH+12].)*

Permita a reprodução

In order to maximize reproducibility, everything needed to re-create the output should be recorded [. . .]. Retirado de [ABH+12].)

Desafios

Two of the biggest challenges scientists and other programmers face when working with code and data are keeping track of changes (and being able to revert them if things go wrong), and working collaboratively on a program or dataset. (Retirado de [ABH+12].)

Desafios

*Two of the **biggest challenges** scientists and other programmers face when working with code and data are **keeping track of changes** (and being able to revert them if things go wrong), and **working collaboratively** on a program or dataset. (Retirado de [ABH+12].)*

Comparativos

email ***shared folder*** **DVCS**

shared folder uso de disco compartilhado por rede local ou serviço nas nuvens, e.g., Dropbox.

DCSV *Distributed Concurrent Versions System*, e.g., git, mercurial (hg).

Comparativos

	email	***shared folder***	**DVCS**
Notificação	Fácil	Difícil	Média

shared folder uso de disco compartilhado por rede local ou serviço nas nuvens, e.g., Dropbox.

DCSV *Distributed Concurrent Versions System*, e.g., git, mercurial (hg).

Comparativos

	email	***shared folder***	**DVCS**
Notificação	Fácil	Difícil	Média
Identificação	Difícil	Fácil	Fácil

shared folder uso de disco compartilhado por rede local ou serviço nas nuvens, e.g., Dropbox.

DCSV *Distributed Concurrent Versions System*, e.g., git, mercurial (hg).

Comparativos

	email	***shared folder***	**DVCS**
Notificação	Fácil	Difícil	Média
Identificação	Difícil	Fácil	Fácil
Desfazer	Difícil	Difícil	Fácil

shared folder uso de disco compartilhado por rede local ou serviço nas nuvens, e.g., Dropbox.

DCSV *Distributed Concurrent Versions System*, e.g., git, mercurial (hg).

Comparativos

	email	***shared folder***	**DVCS**
Notificação	Fácil	Difícil	Média
Identificação	Difícil	Fácil	Fácil
Desfazer	Difícil	Difícil	Fácil
Procurar	Difícil	Difícil	Fácil

shared folder uso de disco compartilhado por rede local ou serviço nas nuvens, e.g., Dropbox.

DCSV *Distributed Concurrent Versions System*, e.g., git, mercurial (hg).

Git

Exemplo

```
git init -q
```

Git

Exemplo

```
git init -q
echo 'Versao 0.1' >> meu_projeto.txt
```

Git

Exemplo

```
git init -q
echo 'Versao 0.1' >> meu_projeto.txt
git add meu_projeto.txt
```

Git

Exemplo

```
git init -q
echo 'Versao 0.1' >> meu_projeto.txt
git add meu_projeto.txt
git commit -q -m 'Iniciei'
```

Git

Exemplo

```
git init -q
echo 'Versao 0.1' >> meu_projeto.txt
git add meu_projeto.txt
git commit -q -m 'Iniciei'
echo 'Versao 0.2' >> meu_projeto.txt
```

Git

Exemplo

```
git init -q
echo 'Versao 0.1' >> meu_projeto.txt
git add meu_projeto.txt
git commit -q -m 'Iniciei'
echo 'Versao 0.2' >> meu_projeto.txt
git add -u
```

Git

Exemplo

```
git init -q
echo 'Versao 0.1' >> meu_projeto.txt
git add meu_projeto.txt
git commit -q -m 'Iniciei'
echo 'Versao 0.2' >> meu_projeto.txt
git add -u
git commit -q -m 'Mudei'
```

Git

Exemplo

```
git init -q
echo 'Versao 0.1' >> meu_projeto.txt
git add meu_projeto.txt
git commit -q -m 'Iniciei'
echo 'Versao 0.2' >> meu_projeto.txt
git add -u
git commit -q -m 'Mudei'
cat meu_projeto.txt
Versao 0.1
Versao 0.2
```

Git

Exemplo

```
git init -q
echo 'Versao 0.1' >> meu_projeto.txt
git add meu_projeto.txt
git commit -q -m 'Iniciei'
echo 'Versao 0.2' >> meu_projeto.txt
git add -u
git commit -q -m 'Mudei'
cat meu_projeto.txt
Versao 0.1
Versao 0.2
git checkout HEAD^ -q
```

Git

Exemplo

```
git init -q
echo 'Versao 0.1' >> meu_projeto.txt
git add meu_projeto.txt
git commit -q -m 'Iniciei'
echo 'Versao 0.2' >> meu_projeto.txt
git add -u
git commit -q -m 'Mudei'
cat meu_projeto.txt
Versao 0.1
Versao 0.2
git checkout HEAD^ -q
cat meu_projeto.txt
Versao 0.1
```

Git

Exemplo

```
git init -q
echo 'Versao 0.1' >> meu_projeto.txt
git add meu_projeto.txt
git commit -q -m 'Iniciei'
echo 'Versao 0.2' >> meu_projeto.txt
git add -u
git commit -q -m 'Mudei'
cat meu_projeto.txt
Versao 0.1
Versao 0.2
git checkout HEAD^ -q
cat meu_projeto.txt
Versao 0.1
git checkout master -q
```

Git

Exemplo

```
git init -q
echo 'Versao 0.1' >> meu_projeto.txt
git add meu_projeto.txt
git commit -q -m 'Iniciei'
echo 'Versao 0.2' >> meu_projeto.txt
git add -u
git commit -q -m 'Mudei'
cat meu_projeto.txt
Versao 0.1
Versao 0.2
git checkout HEAD^ -q
cat meu_projeto.txt
Versao 0.1
git checkout master -q
cat meu_projeto.txt
Versao 0.1
Versao 0.2
```

O que versionar?

[. . .] everything that has been created manually should be put in version control, including programs, original field observations, and the source files for papers. [. . .] (Retirado de [ABH+12].)

O que versionar?

[. . .] ***everything*** *that has been created manually should be put in version control, including programs, original field observations, and the source files for papers. [. . .] (Retirado de [ABH+12].)*

Teste

Realizar testes para verificar a execução correta do código é essencial.
Fazê-los de maneira eficiêncie é uma arte.

Teste

Realizar testes para verificar a execução correta do código é essencial.
Fazê-los de maneira eficiêncie é uma arte.
Os dois tipos principais de testes são:

- White-Box testing e
- Black-box testing.

Teste

Realizar testes para verificar a execução correta do código é essencial.
Fazê-los de maneira eficiêncie é uma arte.
Os dois tipos principais de testes são:

- White-Box testing e
- Black-box testing.

Uma de lista *frameworks* para teste encontra-se em `http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks`.

Python Doctest (1)

Exemplo

```
def sum_vect(a, b):
    """
    This function return the sum of the vectors a and b.

        >>> a = [0, 0, 0]
        >>> b = [1, 1, 1]
        >>> sum_vect(a, b)
        [1, 1, 1]
        >>> b = ['a', 'b', 'c']
        >>> sum_vect(a, b)
        [98, 99, 100]
    """
    n = len(a)
    y = [0 for i in range(n)]
    for i in range(n):
        y[i] = a[i] + b[i]
    return y
```

Python Doctest (2)

Exemplo (Continuação)

```
python -m doctest sum_vect.py
**********************************************************************
File "sum_vect.py", line 10, in sum_vect.sum_vect
Failed example:
    sum_vect(a, b)
Exception raised:
    Traceback (most recent call last):
      File "/usr/lib/python2.7/doctest.py", line 1289, in __run
        compileflags, 1) in test.globs
      File "<doctest␣sum_vect.sum_vect[4]>", line 1, in <module>
        sum_vect(a, b)
      File "sum_vect.py", line 16, in sum_vect
        y[i] = a[i] + b[i]
    TypeError: unsupported operand type(s) for +: 'int' and 'str'
**********************************************************************
1 items had failures:
   1 of   5 in sum_vect.sum_vect
***Test Failed*** 1 failures.
```

Aos poucos ir tornando cada uma das boas práticas um hábito.

Atenção

Não deve-se seguir a risca as soluções aqui propostas.

Pretende-se apresentar o Git com maiores detalhes entre Janeiro e Fevereiro de 2013.
O material para a apresentação já começou a ser desenvolvido e a versão beta encontra-se em `https://github.com/downloads/r-gaia-cs/presentations/bgt.pdf`.

Obrigado!

r.gaia.cs@gmail.com


DA Aruliah, C.T. Brown, N.P.C. Hong, M. Davis, R.T. Guy, S.H.D. Haddock, K. Huff, I. Mitchell, M. Plumbley, B. Waugh, et al.
Best practices for scientific computing.
arXiv preprint arXiv:1210.0530, 2012.