Doing computational science better

Some sources of inspiration
Some tools
Getting help

A vous
Some sources of inspiration
A Quick Guide to Organizing Computational Biology Projects

William Stafford Noble¹,²*
A Quick Guide to Organizing Computational Biology Projects

William Stafford Noble

The core guiding principle is simple: the purpose of this article is to describe a logical organization of files and directories. This organization will make it easy for others to find the data and results you have generated. As you develop your project, you will need to decide upon some organization of files and directories. The directory structure will typically be generated automatically. Someone unfamiliar with your project will probably have to do over again. Murphy's Law: Everything you do, you will probably have to do over again. Therefore, it makes sense to organize your work carefully. In practice, the principles associated with performing computational experiments are applied in practice, let's begin by considering the two principles below.

The first of these two principles is that your work should be able to look at your computer source code automatically generates the three subdirectories: results, src, and bin. The src directory contains the source code and the code to build the executable. The bin directory contains the executable. The results directory contains your output. In addition to the primary text describing the organization, it is often valuable to transcribe notes from conversations as well.
In each results folder:

- **script**: `getResults.rb` or `WHATIDIDID.txt`
- intermediates
- output
Best Practices for Scientific Computing

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.
Best Practices for Scientific Computing


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.

1. Write programs for people, not computers.
Best Practices for Scientific Computing


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.

1. Write programs for people, not computers.
2. Automate repetitive tasks.
Best Practices for Scientific Computing


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.

1. Write programs for people, not computers.
2. Automate repetitive tasks.
3. Use the computer to record history.
Best Practices for Scientific Computing


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.

1. Write programs for people, not computers.
2. Automate repetitive tasks.
3. Use the computer to record history.
4. Make incremental changes.
Best Practices for Scientific Computing


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.

1. Write programs for people, not computers.
2. Automate repetitive tasks.
3. Use the computer to record history.
4. Make incremental changes.
5. Use version control.
Best Practices for Scientific Computing


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.

1. Write programs for people, not computers.
2. Automate repetitive tasks.
3. Use the computer to record history.
4. Make incremental changes.
5. Use version control.
6. Don’t repeat yourself (or others).
Best Practices for Scientific Computing


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.

1. Write programs for people, not computers.
2. Automate repetitive tasks.
3. Use the computer to record history.
4. Make incremental changes.
5. Use version control.
6. Don’t repeat yourself (or others).
7. Plan for mistakes.
Best Practices for Scientific Computing


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.

1. Write programs for people, not computers.
2. Automate repetitive tasks.
3. Use the computer to record history.
4. Make incremental changes.
5. Use version control.
6. Don’t repeat yourself (or others).
7. Plan for mistakes.
8. Optimize software only after it works correctly.
Best Practices for Scientific Computing


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.

1. Write programs for people, not computers.
2. Automate repetitive tasks.
3. Use the computer to record history.
4. Make incremental changes.
5. Use version control.
6. Don’t repeat yourself (or others).
7. Plan for mistakes.
8. Optimize software only after it works correctly.
9. Document the design and purpose of code rather than its mechanics.
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.

1. Write programs for people, not computers.
2. Automate repetitive tasks.
3. Use the computer to record history.
4. Make incremental changes.
5. Use version control.
6. Don’t repeat yourself (or others).
7. Plan for mistakes.
8. Optimize software only after it works correctly.
9. Document the design and purpose of code rather than its mechanics.
10. Conduct code reviews.
Ruby.

(or maybe python)
Ruby.

(or maybe python)

“Friends don’t let friends do Perl” - reddit user
Programming better

• “being able to use understand and improve your code in 6 months & in 60 years” - approximate Damian Conway
Programming better

• “being able to use understand and improve your code in 6 months & in 60 years” - approximate Damian Conway

• variable naming
Programming better

• “being able to use understand and improve your code in 6 months & in 60 years” - approximate Damian Conway

• variable naming

• coding width: 100 characters
Programming better

• “being able to use understand and improve your code in 6 months & in 60 years” - approximate Damian Conway

• variable naming

• coding width: 100 characters

• indenting

Confusing mess...

Nice and clean. mmmmmmmmm...
Programming better

• “being able to use understand and improve your code in 6 months & in 60 years” - approximate Damian Conway
• variable naming
• coding width: 100 characters
• indenting

• Follow conventions -e.g “Google R Style” or https://github.com/hadley/devtools/wiki/Style

Confusing mess...

Nice and clean. mmmmmmmmm...
Programming better

• “being able to use understand and improve your code in 6 months & in 60 years” - approximate Damian Conway

• variable naming

• coding width: 100 characters

• indenting

• Follow conventions -eg “Google R Style” or https://github.com/hadley/devtools/wiki/Style

• Versioning: DropBox & http://github.com/
Programming better

• “being able to use understand and improve your code in 6 months & in 60 years” - approximate Damian Conway

• variable naming

• coding width: 100 characters

• indenting

• Follow conventions -eg “Google R Style” or https://github.com/hadley/devtools/wiki/Style

• Versioning: Dropbox & http://github.com/

• Automated testing. e.g.
Programming better

• “being able to use understand and improve your code in 6 months & in 60 years” - approximate Damian Conway

• variable naming

• coding width: 100 characters

• indenting

• Follow conventions -eg “Google R Style” or https://github.com/hadley/devtools/wiki/Style

• Versioning: DropBox & http://github.com/

• Automated testing. e.g.: if (testing) {
    # run a bunch of tests of extreme situations.
    # quit if a test gives a weird result.
}

    # real part of function.

```r
preprocess_snps <- function(snp_table, testing=FALSE) {
    # run a bunch of tests of extreme situations.
    # quit if a test gives a weird result.
    }
    # real part of function.
```

Confusing mess...

Nice and clean. mmmmmmmm...
A few tools
# This is my project intro

Yes oh yes ants are the best

# Results

Lorem ipsum **dolor sit amet**, consectetur adipiscing elit. Morbi a quam et urna fringilla... a facilisis. Sed commodo, turpis et luctus pellentesque, nisl nunc luctus mauris, ut sollicit... tudi nian massa eu dolor. Phasellus interdum neque porta lorem vehicula auctor. Etiam j... usto magna, aliquam at tempus non, adipiscing vitae nibh. Integer pharetra laoreet eros, a t ultrices leo gravida vel. Integer sollicitudin nibh eros, ut ullamcorper tellus. *Nulla ac tor... tor sed massa bibendum accumsan et fringilla ligula*. Etiam at metus lorem, vitae euismo... d metus. Maecenas sollicitudin elit eget nulla consequat fermentum tincidunt ipsum adipi... scing. Donec ut fringilla turpis. Nunc augue purus, elementum id imperdiet et, volutpat v... el magna. Donec euismod libero non augue varius sed venenatis magna tempor. Suspendi... sse rhoncus felis velit, et scelerisque risus.

## They really are

Uh-huh

```
./this_script_shows_what_happens > output
```

## They really really are

Ok good job because:

* bla
* blabla
* blablabla

# Conclusion

You win: Ants are cool. I want to look at them and crush them and sequence them and ge... notype them.
# This is my project intro

Yes oh yes ants are the best

# Results


Nulla ac tor
tor sed massa bibendum accumsan et fringilla ligula.* Etiam at metus lorem, vitae eusimo
d metus. Maecenas sollicitudin elit eget nulla consequat fermentum tincidunt ipsum adipiscing. Donec ut fringilla turpis. Nunc augue purus, elementum id imperdiet et, volutpat v
el magna. Donec eusimod libero non augue varius sed venenatis magna tempor. Suspendisse rhoncus felis velit, et scelerisque risus.

## They really are

Uh-huh

... ./this_script_shows_what_happens > output

## They really really are

Ok good job because:

* bla
* blabla
* blablabl

# Conclusion

You win: Ants are cool. I want to look at them and crush them and sequence them and get...
knitr (sweave) Analyzing & Reporting in a single file.

MyFile.Rnw
knitr (sweave) Analyzing & Reporting in a single file.

MyFile.Rnw

\documentclass{article}
\usepackage[sc]{mathpazo}
\usepackage[T1]{fontenc}
\usepackage[url]{url}
\begin{document}

<<setup, include=FALSE, cache=FALSE, echo=FALSE>>=
# this is equivalent to \SweaveOpts{...}
opts_chunk$set(fig.path='figure/minimal-', fig.align='center', fig.show='hold')
options(replace.assign=TRUE,width=90)
@

\title{A Minimal Demo of knitr}
\author{Yihui Xie}
\maketitle

You can test if \textbf{knitr} works with this minimal demo. OK, let's get started with some boring random numbers:

<<boring-random,echo=TRUE,cache=TRUE>>=
set.seed(1121)
(x=rnorm(20))
mean(x);var(x)
@

The first element of \texttt{x} is \Sexpr{x[1]}. Boring boxplots and histograms recorded by the PDF device:

<<boring-plots,cache=TRUE,echo=TRUE>>=
## two plots side by side
par(mar=c(4,4,.1,.1),cex.lab=.95,cex.axis=.9,mgp=c(2,.7,0),tcl=-.3,las=1)
boxplot(x)
hist(x,main='')
@

Do the above chunks work? You should be able to compile the \TeX{}
You can test if \textbf{knitr} works with this minimal demo. OK, let's get started with some boring random numbers:

\begin{verbatim}
<<boring-random,echo=TRUE,cache=TRUE>>=
set.seed(1121)
(x=rnorm(20))
mean(x);var(x)
\end{verbatim}

The first element of \texttt{x} is \Sexpr{x[1]}. Boring boxplots and histograms recorded by the PDF device:

\begin{verbatim}
<<boring-plots,cache=TRUE,echo=TRUE>>=
## two plots side by side
par(mar=c(4,4,.1,.1),cex.lab=.95,cex.axis=.9,mgp=c(2,.7,0),tcl=-.3,las=1)
boxplot(x)
hist(x,main='')
\end{verbatim}

Do the above chunks work? You should be able to compile the \TeX{} document and get a PDF file like this one: \url{https://github.com/downloads/yihui/knitr/knitr-minimal.pdf}.

The Rnw source of this document is at \url{https://github.com/yihui/knitr/blob/master/inst/examples/knitr-minimal.Rnw}.
## A Minimal Demo of knitr

You can test if \textbf{knitr} works with this minimal demo. OK, let's get started with some boring random numbers:

### in R:
```r
library(knitr)
knit("MyFile.Rnw")
# --> creates MyFile.tex
```

### in shell:
```bash
pdflatex MyFile.tex
# --> creates MyFile.pdf
```

---

You can test if \textbf{knitr} works with this minimal demo. OK, let's get started with some boring random numbers:

```r
set.seed(1121)
x <- rnorm(20)
mean(x); var(x)
```

The first element of `x` is `0.14496`. Boring boxplots and histograms recorded by the PDF device:

```r
## two plots side by side
par(mar=c(4,4,.1,.1), cex.lab=.95,cex.axis=.9,mgp=c(2,7,0),tcl=-.3,las=1)
boxplot(x)
hist(x,main='')
```

Do the above chunks work? You should be able to compile the \TeX{} document and get a PDF file like this one: [https://github.com/downloads/yihui/knitr/knitr-minimal.pdf](https://github.com/downloads/yihui/knitr/knitr-minimal.pdf). The Rnw source of this document is at [https://github.com/yihui/knitr/blob/master/inst/examples/knitr-minimal.Rnw](https://github.com/yihui/knitr/blob/master/inst/examples/knitr-minimal.Rnw).

---

Yihui Xie

February 26, 2012
A Minimal Demo of knitr

Yihui Xie
February 26, 2012

You can test if \textbf{knitr} works with this minimal demo. OK, let’s get started with some boring random numbers:

\begin{verbatim}
set.seed(1121)
(x <- rnorm(20))
\end{verbatim}

\begin{verbatim}
## [1] 0.14496 0.43832 0.15319 1.08494 1.99954 -0.81188 0.16027 0.58589 0.36009
## [10] -0.02531 0.15088 0.11008 1.35968 -0.32699 -0.71638 1.80977 0.50840 -0.52746
## [19] 0.13272 -0.15594
\end{verbatim}

\begin{verbatim}
mean(x)
\end{verbatim}

\begin{verbatim}
## [1] 0.3217
\end{verbatim}

\begin{verbatim}
var(x)
\end{verbatim}

\begin{verbatim}
## [1] 0.5715
\end{verbatim}

The first element of \texttt{x} is 0.145. Boring boxplots and histograms recorded by the PDF device:

\begin{verbatim}
boxplot(x)
hist(x, main = "")
\end{verbatim}

Do the above chunks work? You should be able to compile the \TeX{} document and get a PDF file like this one: https://github.com/downloads/yihui/knitr/knitr-minimal.pdf. The Rnw source of this document is at https://github.com/yihui/knitr/blob/master/inst/examples/knitr-minimal.Rnw.
Plotting in R
Plotting in R

• R’s graphs suck:
  • embarassingly ugly
  • require tweaking in Illustrator --> hard to automate.
  • counterintuitive & inconsistent API --> hard to switch between e.g. histogram and density plot.
  • hard to customize.

• --> Need for something beautiful, easy & effortless.
ggplot2: beautiful & (almost) effortless R plots

```r
> library(ggplot2)
> mtcars

<table>
<thead>
<tr>
<th></th>
<th>mpg</th>
<th>cyl</th>
<th>disp</th>
<th>hp</th>
<th>drat</th>
<th>wt</th>
<th>qsec</th>
<th>vs</th>
<th>am</th>
<th>gear</th>
<th>carb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mazda RX4</td>
<td>21.0</td>
<td>6</td>
<td>160.0</td>
<td>110</td>
<td>3.90</td>
<td>2.620</td>
<td>16.46</td>
<td>8</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Mazda RX4 Wag</td>
<td>21.0</td>
<td>6</td>
<td>160.0</td>
<td>110</td>
<td>3.90</td>
<td>2.875</td>
<td>17.02</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Datsun 710</td>
<td>22.8</td>
<td>4</td>
<td>108.0</td>
<td>93</td>
<td>3.85</td>
<td>2.320</td>
<td>18.61</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Hornet 4 Drive</td>
<td>21.4</td>
<td>6</td>
<td>258.0</td>
<td>110</td>
<td>3.08</td>
<td>3.215</td>
<td>19.44</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Hornet Sportabout</td>
<td>18.7</td>
<td>8</td>
<td>360.0</td>
<td>175</td>
<td>3.15</td>
<td>3.440</td>
<td>17.02</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Valiant</td>
<td>18.1</td>
<td>6</td>
<td>225.0</td>
<td>105</td>
<td>2.76</td>
<td>3.460</td>
<td>20.22</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Duster 360</td>
<td>14.3</td>
<td>8</td>
<td>360.0</td>
<td>245</td>
<td>3.21</td>
<td>3.570</td>
<td>15.84</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Merc 240D</td>
<td>24.4</td>
<td>4</td>
<td>146.7</td>
<td>62</td>
<td>3.69</td>
<td>3.190</td>
<td>20.00</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Merc 230</td>
<td>22.8</td>
<td>4</td>
<td>140.8</td>
<td>95</td>
<td>3.92</td>
<td>3.150</td>
<td>22.90</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Merc 280</td>
<td>19.2</td>
<td>6</td>
<td>167.6</td>
<td>123</td>
<td>3.92</td>
<td>3.440</td>
<td>18.30</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Merc 280C</td>
<td>17.8</td>
<td>6</td>
<td>167.6</td>
<td>123</td>
<td>3.92</td>
<td>3.440</td>
<td>18.90</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Merc 440</td>
<td>16.4</td>
<td>6</td>
<td>275.8</td>
<td>180</td>
<td>3.07</td>
<td>4.070</td>
<td>17.40</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Merc 450SE</td>
<td>17.3</td>
<td>8</td>
<td>275.8</td>
<td>180</td>
<td>3.07</td>
<td>3.730</td>
<td>17.60</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Merc 450SL</td>
<td>15.2</td>
<td>8</td>
<td>275.8</td>
<td>180</td>
<td>3.07</td>
<td>3.780</td>
<td>18.00</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Cadillac Fleetwood</td>
<td>18.4</td>
<td>8</td>
<td>472.0</td>
<td>205</td>
<td>2.93</td>
<td>5.250</td>
<td>17.98</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Lincoln Continental</td>
<td>18.4</td>
<td>8</td>
<td>460.0</td>
<td>215</td>
<td>3.00</td>
<td>4.424</td>
<td>17.82</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Chrysler Imperial</td>
<td>14.7</td>
<td>8</td>
<td>440.0</td>
<td>230</td>
<td>3.23</td>
<td>5.345</td>
<td>17.42</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Fiat 128</td>
<td>32.4</td>
<td>4</td>
<td>78.7</td>
<td>66</td>
<td>4.08</td>
<td>2.200</td>
<td>19.47</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Honda Civic</td>
<td>30.4</td>
<td>4</td>
<td>75.7</td>
<td>62</td>
<td>4.93</td>
<td>1.615</td>
<td>18.52</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Toyota Corolla</td>
<td>33.9</td>
<td>4</td>
<td>71.1</td>
<td>65</td>
<td>4.22</td>
<td>1.835</td>
<td>19.90</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
```
ggplot2: beautiful & (almost) effortless R plots

```r
> library(ggplot2)
> mtcars

<table>
<thead>
<tr>
<th></th>
<th>mpg</th>
<th>cyl</th>
<th>disp</th>
<th>hp</th>
<th>drat</th>
<th>wt</th>
<th>qsec</th>
<th>vs</th>
<th>am</th>
<th>gear</th>
<th>carb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mazda RX4</td>
<td>21.0</td>
<td>6</td>
<td>160</td>
<td>110</td>
<td>3.90</td>
<td>2.620</td>
<td>16.46</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Mazda RX4 Wag</td>
<td>21.0</td>
<td>6</td>
<td>160</td>
<td>110</td>
<td>3.90</td>
<td>2.875</td>
<td>17.02</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Datsun 710</td>
<td>22.8</td>
<td>4</td>
<td>108</td>
<td>93</td>
<td>3.85</td>
<td>2.320</td>
<td>18.61</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Hornet 4 Drive</td>
<td>21.4</td>
<td>6</td>
<td>258</td>
<td>110</td>
<td>3.08</td>
<td>3.215</td>
<td>19.44</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Hornet Sportabout</td>
<td>18.7</td>
<td>8</td>
<td>360</td>
<td>175</td>
<td>3.15</td>
<td>3.440</td>
<td>17.02</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Valiant</td>
<td>18.1</td>
<td>6</td>
<td>225</td>
<td>105</td>
<td>2.76</td>
<td>3.460</td>
<td>20.22</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Duster 360</td>
<td>14.3</td>
<td>8</td>
<td>360</td>
<td>245</td>
<td>3.21</td>
<td>3.570</td>
<td>15.84</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Merc 240D</td>
<td>24.4</td>
<td>4</td>
<td>146.7</td>
<td>240</td>
<td>3.69</td>
<td>3.190</td>
<td>20.00</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Merc 230</td>
<td>22.8</td>
<td>4</td>
<td>140.8</td>
<td>97</td>
<td>3.92</td>
<td>3.150</td>
<td>22.90</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Merc 280</td>
<td>19.2</td>
<td>6</td>
<td>167.6</td>
<td>123</td>
<td>3.92</td>
<td>3.440</td>
<td>18.33</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Merc 280C</td>
<td>17.8</td>
<td>6</td>
<td>167.6</td>
<td>123</td>
<td>3.92</td>
<td>3.440</td>
<td>18.90</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Merc 450SE</td>
<td>16.4</td>
<td>8</td>
<td>275.8</td>
<td>180</td>
<td>3.07</td>
<td>4.070</td>
<td>17.44</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Merc 450SL</td>
<td>17.3</td>
<td>8</td>
<td>275.8</td>
<td>180</td>
<td>3.07</td>
<td>3.700</td>
<td>17.60</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Merc 450SLC</td>
<td>15.2</td>
<td>8</td>
<td>275.8</td>
<td>180</td>
<td>3.07</td>
<td>3.700</td>
<td>18.00</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Cadillac Fleetwood</td>
<td>18.4</td>
<td>8</td>
<td>472.0</td>
<td>205</td>
<td>2.93</td>
<td>5.250</td>
<td>17.98</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Lincoln Continental</td>
<td>10.4</td>
<td>8</td>
<td>460</td>
<td>215</td>
<td>3.00</td>
<td>5.424</td>
<td>17.82</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Chrysler Imperial</td>
<td>14.7</td>
<td>8</td>
<td>440</td>
<td>230</td>
<td>3.23</td>
<td>3.455</td>
<td>17.42</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Fiat 128</td>
<td>32.4</td>
<td>4</td>
<td>70.7</td>
<td>66</td>
<td>4.08</td>
<td>2.200</td>
<td>19.47</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Honda Civic</td>
<td>30.4</td>
<td>4</td>
<td>75.7</td>
<td>52</td>
<td>4.93</td>
<td>1.615</td>
<td>18.52</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Toyota Corolla</td>
<td>33.9</td>
<td>4</td>
<td>71.1</td>
<td>65</td>
<td>4.22</td>
<td>1.835</td>
<td>19.90</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

ggplot(mtcars, aes(factor(cyl))) + geom_bar()
ggplot2: beautiful & (almost) effortless R plots

```r
library(ggplot2)
mtcars

<table>
<thead>
<tr>
<th>model</th>
<th>mpg</th>
<th>cyl</th>
<th>disp</th>
<th>hp</th>
<th>drat</th>
<th>wt</th>
<th>qsec</th>
<th>vs</th>
<th>am</th>
<th>gear</th>
<th>carb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mazda RX4</td>
<td>21.0</td>
<td>6</td>
<td>160.0</td>
<td>110</td>
<td>3.90</td>
<td>2.620</td>
<td>16.46</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Mazda RX4 Wag</td>
<td>21.0</td>
<td>6</td>
<td>160.0</td>
<td>110</td>
<td>3.90</td>
<td>2.875</td>
<td>17.02</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Datsun 710</td>
<td>22.8</td>
<td>4</td>
<td>108.0</td>
<td>93</td>
<td>3.85</td>
<td>2.320</td>
<td>18.61</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Hornet 4 Drive</td>
<td>21.4</td>
<td>6</td>
<td>258.0</td>
<td>110</td>
<td>3.08</td>
<td>3.215</td>
<td>19.44</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Hornet Sportabout</td>
<td>18.7</td>
<td>8</td>
<td>360.0</td>
<td>175</td>
<td>3.15</td>
<td>3.440</td>
<td>17.02</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Valiant</td>
<td>18.1</td>
<td>6</td>
<td>225.0</td>
<td>105</td>
<td>3.76</td>
<td>3.600</td>
<td>20.22</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Duster 360</td>
<td>14.3</td>
<td>8</td>
<td>369.0</td>
<td>245</td>
<td>3.44</td>
<td>3.950</td>
<td>19.70</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Merc 240D</td>
<td>24.4</td>
<td>4</td>
<td>146.7</td>
<td>62</td>
<td>3.92</td>
<td>3.090</td>
<td>20.60</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Merc 230</td>
<td>22.8</td>
<td>4</td>
<td>140.8</td>
<td>95</td>
<td>3.92</td>
<td>3.150</td>
<td>22.90</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Merc 280</td>
<td>19.2</td>
<td>6</td>
<td>167.6</td>
<td>123</td>
<td>3.16</td>
<td>3.440</td>
<td>18.30</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Merc 280C</td>
<td>17.8</td>
<td>6</td>
<td>167.6</td>
<td>123</td>
<td>3.19</td>
<td>3.440</td>
<td>18.30</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Merc 450SE</td>
<td>16.4</td>
<td>8</td>
<td>277.8</td>
<td>180</td>
<td>3.07</td>
<td>4.070</td>
<td>15.80</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Merc 450SL</td>
<td>17.3</td>
<td>8</td>
<td>275.8</td>
<td>180</td>
<td>3.07</td>
<td>3.730</td>
<td>17.40</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Merc 450SLC</td>
<td>15.2</td>
<td>8</td>
<td>275.8</td>
<td>180</td>
<td>3.07</td>
<td>3.780</td>
<td>17.50</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Cadillac Fleetwood</td>
<td>18.4</td>
<td>8</td>
<td>472.0</td>
<td>205</td>
<td>2.93</td>
<td>5.600</td>
<td>18.90</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Lincoln Continental</td>
<td>18.0</td>
<td>8</td>
<td>460.0</td>
<td>215</td>
<td>3.00</td>
<td>5.240</td>
<td>17.22</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Chrysler Imperial</td>
<td>14.7</td>
<td>8</td>
<td>440.0</td>
<td>230</td>
<td>3.23</td>
<td>3.545</td>
<td>15.84</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Fiat 128</td>
<td>32.4</td>
<td>4</td>
<td>78.7</td>
<td>66</td>
<td>4.08</td>
<td>2.200</td>
<td>19.47</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Honda Civic</td>
<td>30.4</td>
<td>4</td>
<td>75.7</td>
<td>52</td>
<td>4.93</td>
<td>1.645</td>
<td>18.90</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Toyota Corolla</td>
<td>33.9</td>
<td>4</td>
<td>71.1</td>
<td>55</td>
<td>4.22</td>
<td>1.835</td>
<td>18.90</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

ggplot(mtcars, aes(factor(cyl))) + geom_bar()
ggplot(mtcars, aes(factor(cyl), fill=factor(gear))) + geom_bar()
ggbio: an R package for extending the grammar of graphics for genomic data
Getting help.
Getting help.
Getting help.

- In real life: Make friends with people. Talk to them.
Getting help.

- In real life: Make friends with people. Talk to them.

- Online:
Getting help.

• In real life: Make friends with people. Talk to them.

• Online:
  • Specific discussion mailing lists (e.g.: R, Stacks, bioruby, MAKER... )
Getting help.

• In real life: Make friends with people. Talk to them.

• Online:
  • Specific discussion mailing lists (e.g.: R, Stacks, bioruby, MAKER...)
  • Programming: http://stackoverflow.com
Getting help.

• In real life: Make friends with people. Talk to them.

• Online:
  • Specific discussion mailing lists (e.g.: R, Stacks, bioruby, MAKER...)
  • Programming: http://stackoverflow.com
  • Bioinformatics: http://www.biostars.org
Getting help.

• In real life: Make friends with people. Talk to them.

• Online:
  • Specific discussion mailing lists (e.g.: R, Stacks, bioruby, MAKER...)
  • Programming: http://stackoverflow.com
  • Bioinformatics: http://www.biostars.org
  • Sequencing-related: http://seqanswers.com
Where to advertise or find bioinformatics jobs

Data Selection With/Without Databases (Large Data Sets, ORMs, and Speed)

Counting n's within fasta

Transcription Factor Enrichment

How to detect and query poly-allelic SNPs?


Extract according to row

Forum: How to be helpful as a BioStar moderator/editor?

Find nearest gene upstream using mysql and perl program
Question: extracting sequence from a 3GB fasta file

Hi,
How to extract fasta sequence from an huge 3gb fasta file by giving sequence id as input using perl, Thanks in advance.

12
sequence retrieval fasta perl

similar posts • permalink • comment • revisions

↑ 1 I've modified your original question, as it was not very clear. You should put an example of your input file and an example of your output. Is your input file a fasta file?

reply • written 2.1 years ago by Giovanni M Dall'Olio • 1094 • 1 • 15 • 35

11 answers

If I read your question correctly, you have many sequences in a large file and you want to retrieve certain sequences by some ID.

One way to do this using Perl is first to index the file. If you install Bioperl and its accessory scripts, you can do this using bp_index.pl:

This example assumes that your fasta sequences are in myfile.fa in the current directory and you want to create the index file, myIndex, also in the current directory:

```bash
bp_index.pl -dir . -fmt fasta myIndex myfile.fa
```

You can then retrieve by ID using bp_fetch.pl. Assuming that you are in the same directory and you want the sequence with ID myID, something like:

```bash
bp_fetch.pl -dir . -fmt fasta myIndex:myID
```

It's been some time since I used these tools, so you should check the syntax and read up on them at the Bioperl

15

created 2.1 years ago by Neillw • 2884 • 1 • 20 • 49

11 answers
• Once I wanted to set up a BLAST server.
• Once I wanted to set up a BLAST server.

Anurag Priyam, Mechanical engineering student, Kharagpur
• Once I wanted to set up a BLAST server.

Anurag Priyam, Mechanical engineering student, Kharagpur

Aim: An open source idiot-proof web-interface for custom BLAST
http://www.sequenceserver.com/

1. Installing

```
gem install sequenceserver
```
1. Installing

```
$ gem install sequenceserver
```

# .sequenceserver.conf

```bash
bin: ~/ncbi-blast-2.2.25+/bin/
database: /Users/me/blast_databases/
```
http://www.sequenceserver.com/

1. Installing

```bash
$ gem install sequenceserver
```

# .sequenceserver.conf

```bash
bin: ~/ncbi-blast-2.2.25+/bin/
database: /Users/me/blast_databases/
```

2. Configure.

```bash
# .sequenceserver.conf
bin: ~/ncbi-blast-2.2.25+/bin/
database: /Users/me/blast_databases/
```

3. Launch.

```bash
$ sequenceserver
### Launched SequenceServer at: http://0.0.0.0:4567
```
http://www.sequenceserver.com/

1. Installing

```bash

gem install sequenceserver
```

(requires a BLAST+ install)

Do you have BLAST-formatted databases? If not:

```bash
sequenceserver format-databases /path/to/fastas
```

2. Configure.

```bash
# .sequenceserver.conf
bin: ~/ncbi-blast-2.2.25+/bin/
database: /Users/me/blast_databases/
```

3. Launch.

```bash
sequenceserver
### Launched SequenceServer at: http://0.0.0.0:4567
```
>mysequence
ACCCACCCACGATAGAGAGAG

>MyOTHERSequence
acaccacaggtagatagagagagagagagagacacagtacagtagtagagatta

Detected: nucleotide sequence(s).

Nucleotide databases
- Acromyrmex echinatior genome 2.0
- Acromyrmex echinatior predicted transcripts 3.8
- Atta cephalotes genome
- Atta cephalotes predicted transcripts 1.2
- Camponotus floridanus genome 3.3
- Camponotus floridanus predicted transcripts 3.3
- Camponotus floridanus transcriptome (assembled from RNA)
- Harpegnathos saltator genome 3.3
- Harpegnathos saltator predicted transcripts 3.3
- Harpegnathos saltator transcriptome (assembled from RNA)
- Linepithema humile genome 4
- Linepithema humile predicted transcripts 1.2
- Nylanderia pubens transcriptome (assembled from RNA)
- Pogonomyrmex barbatus genome 03
- Pogonomyrmex barbatus predicted transcripts 1.2
- Solenopsis invicta genome Sl_gnF
- Solenopsis invicta predicted transcripts 2.2.3
- [Other ants] Genbank download 2011-09-06
- [Outgroup] Apis mellifera genome 4.5
- [Outgroup] Apis mellifera predicted transcripts prerelease-2
- [Outgroup] Bombus terrestris genome 1.1
- [Outgroup] Nasonia vitripennis genome 2.0
- [Raw unassembled reads] Linepithema humile genome
- [Raw unassembled reads] Nasonia vitripennis predicted transcripts 1.2
- [Raw unassembled reads] Pogonomyrmex barbatus genome
- [Raw unassembled reads] Pogonomyrmex barbatus transcriptome
- [Raw unassembled reads] Solenopsis invicta genome

Protein databases
- Acromyrmex echinatior proteins 3.8
- Atta cephalotes proteins 1.2
- Camponotus floridanus proteins 3.3
- Harpegnathos saltator proteins 3.3
- Linepithema humile proteins 1.2
- Pogonomyrmex barbatus proteins 1.2
- Solenopsis invicta proteins 2.2.3
- [Outgroup] Apis mellifera proteins prerelease-2
- [Outgroup] Nasonia vitripennis proteins 1.2

Advanced Parameters: eg: -evalue 1.0e-5 -num_alignments 100

BLASTX
Let's try something

- Code review
- Examine a style guide
- Set up SequenceServer BLAST server
- take notes in Markdown & convert them to pdf
- perform analysis in R/knitr report and make pretty output
- make graphs in ggplot
Best Practices for Scientific Computing


* University of Toronto (guy@cs.utoronto.ca),
† Mary University of London (mark.plumbley@eecs.qmul.ac.uk),
‡ Michigan State University (ctb@msu.edu),
§ University of British Columbia (mitchell@cs.ubc.ca),
¶ Monterey Bay Aquarium Research Institute (mrdavis@stsci.edu),
¶¶ Space Telescope Science Institute (N.ChueHong@epcc.ed.ac.uk),
¶¶¶ Weizmann Institute of Science (ianmitchell@weizmann.ac.il),
‡‡‡ ETH Zurich (dhavide.aruliah@inf.ethz.ch),
‡‡§§ QMUL (ethan@weecology.org), and
‡‡¶¶ University of Ontario Institute of Technology (Dhavide.Aruniah@uoit.ca), and
‡¶¶* University of California, Irvine (plumbley@uci.edu), and
∥∥∥* University of California, Irvine (wkguy@uci.edu), and
∥∥∥∥* University of California, Irvine (mrdavis@stsci.edu), and
∥∥∥∥∥* University of California, Irvine (plumbley@uci.edu), and
Best Practices for Scientific Computing


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.
Best Practices for Scientific Computing


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.

1. Write programs for people, not computers.
Best Practices for Scientific Computing


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.

1. Write programs for people, not computers.
2. Automate repetitive tasks.
Best Practices for Scientific Computing


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.

1. Write programs for people, not computers.
2. Automate repetitive tasks.
3. Use the computer to record history.
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.

1. Write programs for people, not computers.
2. Automate repetitive tasks.
3. Use the computer to record history.
4. Make incremental changes.
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.

1. Write programs for people, not computers.
2. Automate repetitive tasks.
3. Use the computer to record history.
4. Make incremental changes.
5. Use version control.
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.

1. Write programs for people, not computers.
2. Automate repetitive tasks.
3. Use the computer to record history.
4. Make incremental changes.
5. Use version control.
6. Don’t repeat yourself (or others).
Best Practices for Scientific Computing


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.

1. Write programs for people, not computers.
2. Automate repetitive tasks.
3. Use the computer to record history.
4. Make incremental changes.
5. Use version control.
6. Don’t repeat yourself (or others).
7. Plan for mistakes.
Best Practices for Scientific Computing


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.

1. Write programs for people, not computers.
2. Automate repetitive tasks.
3. Use the computer to record history.
4. Make incremental changes.
5. Use version control.
6. Don’t repeat yourself (or others).
7. Plan for mistakes.
8. Optimize software only after it works correctly.
Best Practices for Scientific Computing


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.

1. Write programs for people, not computers.
2. Automate repetitive tasks.
3. Use the computer to record history.
4. Make incremental changes.
5. Use version control.
6. Don’t repeat yourself (or others).
7. Plan for mistakes.
8. Optimize software only after it works correctly.
9. Document the design and purpose of code rather than its mechanics.
Best Practices for Scientific Computing


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.

1. Write programs for people, not computers.
2. Automate repetitive tasks.
3. Use the computer to record history.
4. Make incremental changes.
5. Use version control.
6. Don’t repeat yourself (or others).
7. Plan for mistakes.
8. Optimize software only after it works correctly.
9. Document the design and purpose of code rather than its mechanics.
10. Conduct code reviews.