Doing computational science better

Some sources of inspiration
Some tools
Getting help

A vous

Some sources of inspiration

OPEN ACCESS Freely available online

PLoS COMPUTATIONAL BIOLOGY

Education

A Quick Guide to Organizing Computational Biology Projects

William Stafford Noble[1,2]*

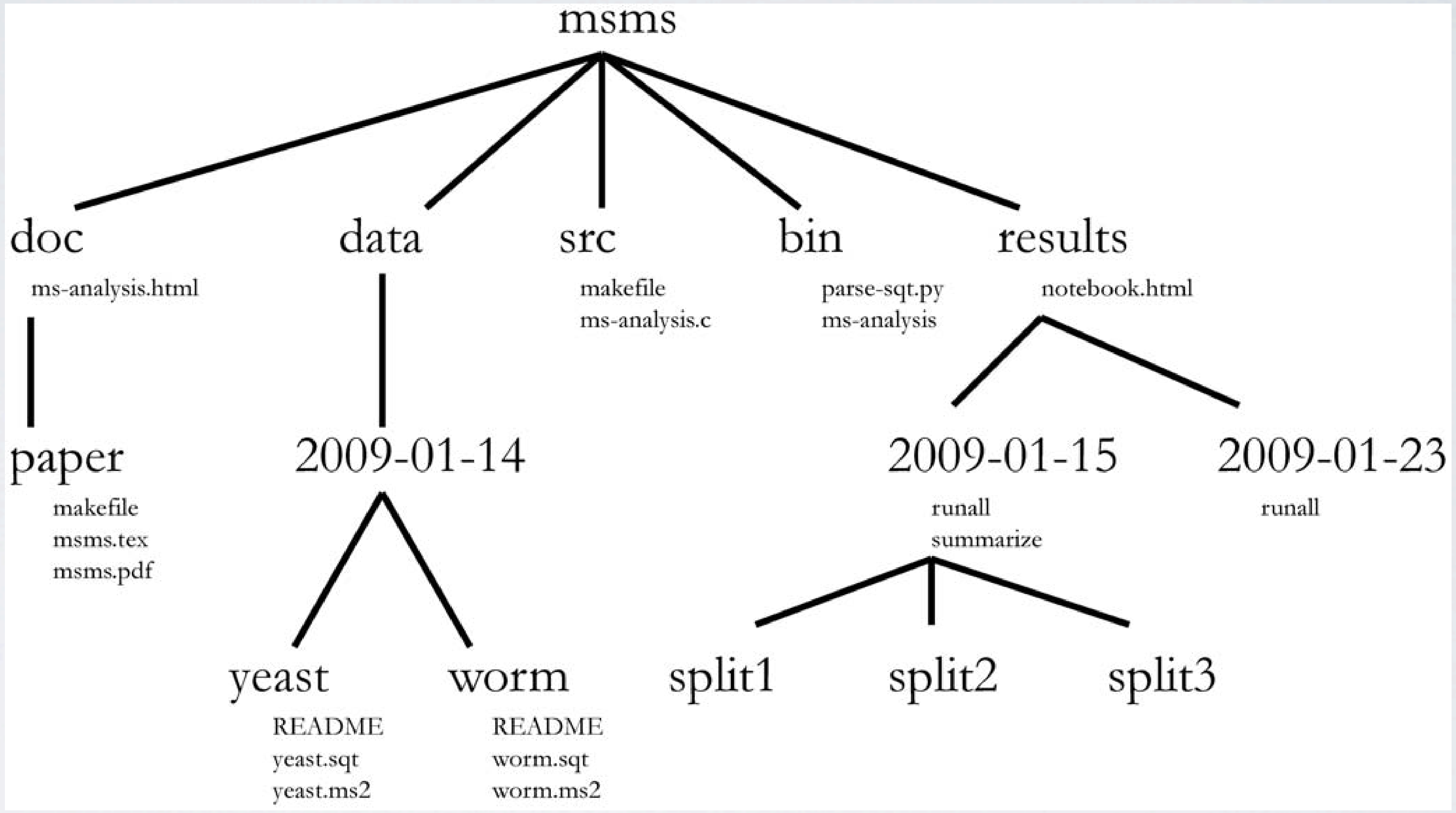
OPEN ACCESS Freely available online

PLoS COMPUTATIONAL BIOLOGY

Education

A Quick Guide to Organizing Computational Biology Projects

William Stafford Noble[1,2]*



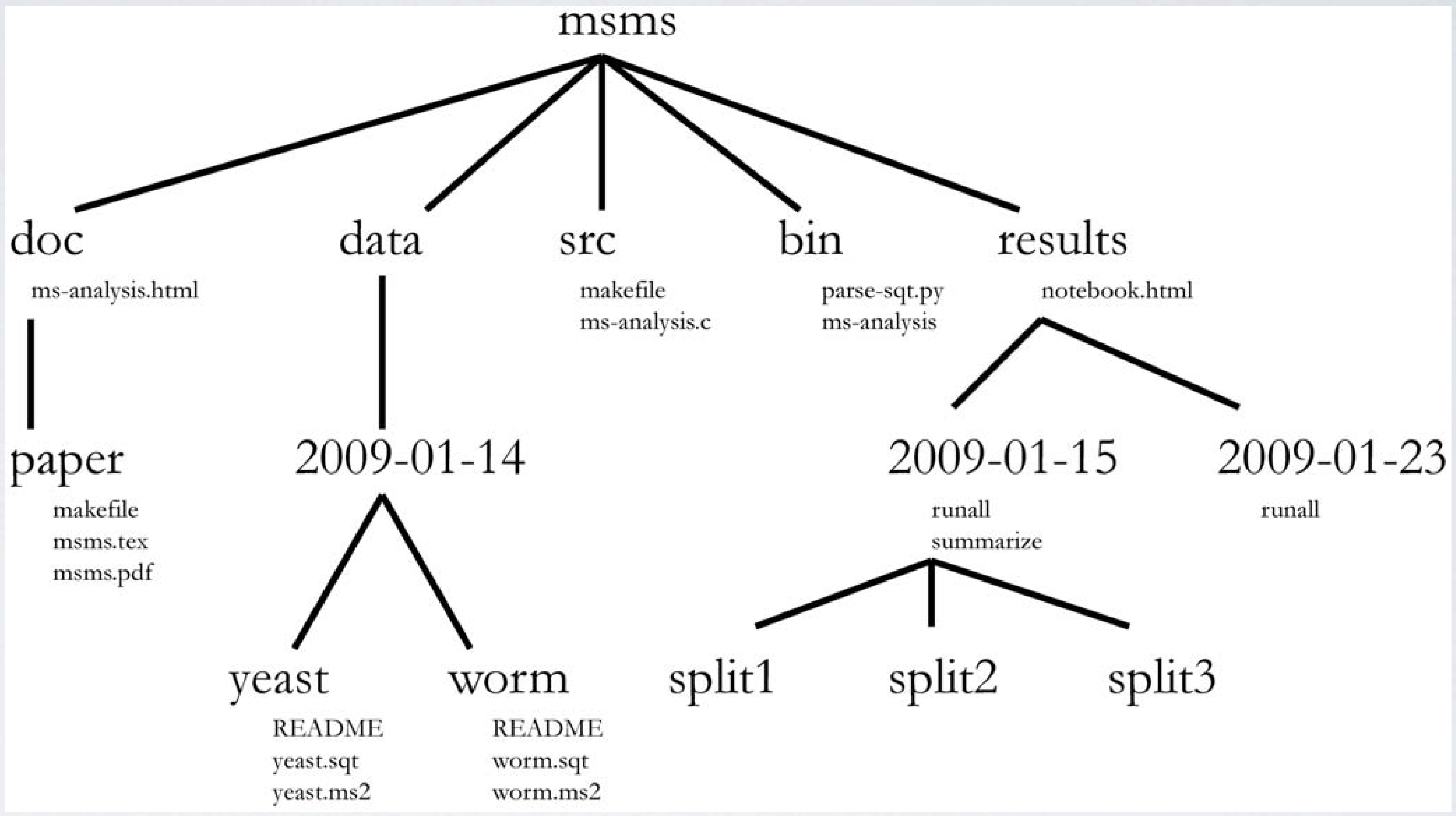
OPEN ACCESS Freely available online

PLoS COMPUTATIONAL BIOLOGY

Education

A Quick Guide to Organizing Computational Biology Projects

William Stafford Noble[1,2]*



In each results folder:

- script: **`getResults.rb`** or **`WHATIDID.txt`**
- intermediates
- output

Best Practices for Scientific Computing

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

Best Practices for Scientific Computing

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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.

arXiv:1210.0530v3 [cs.MS] 29 Nov 2012

Best Practices for Scientific Computing

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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.

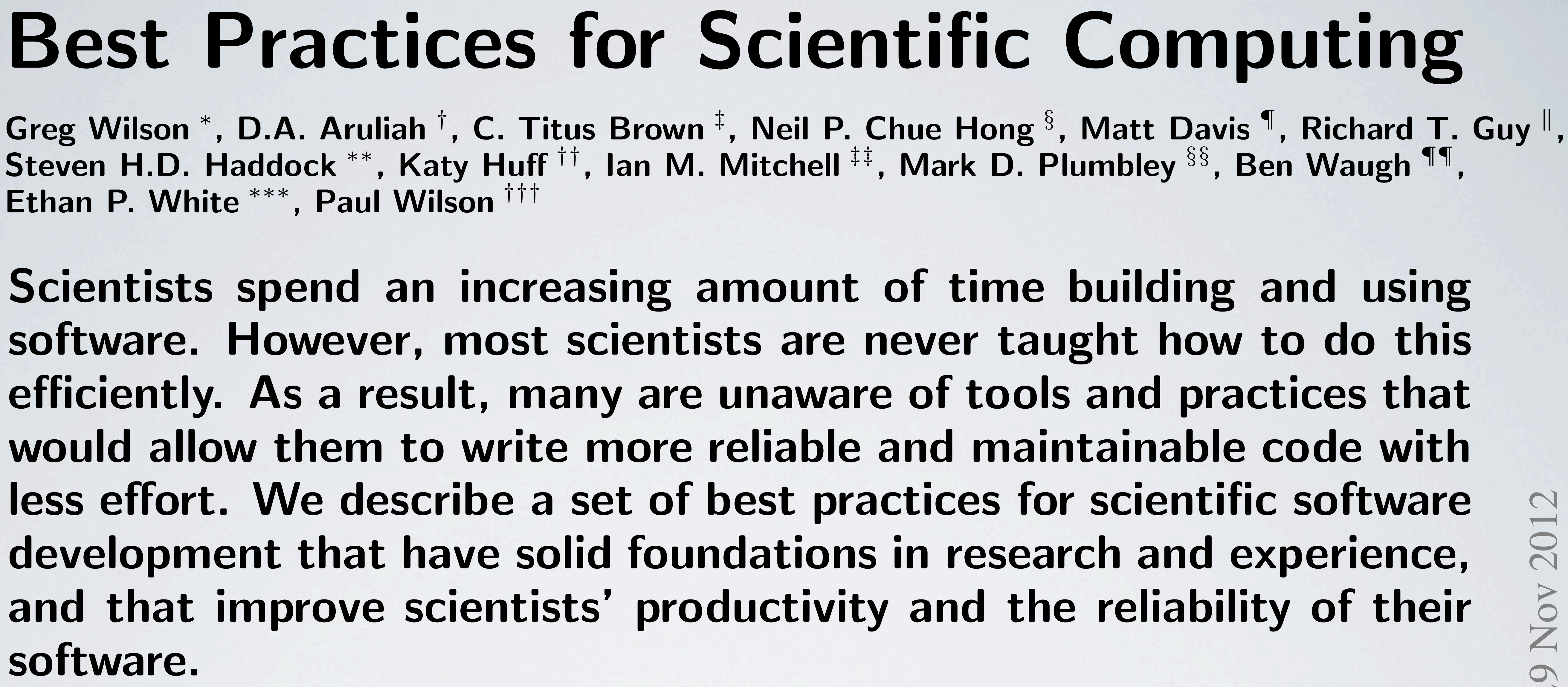
1. Write programs for people, not computers.
2. Automate repetitive tasks.
3. Use the computer to record history.

Best Practices for Scientific Computing

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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.

arXiv:1210.0530v3 [cs.MS] 29 Nov 2012

Best Practices for Scientific Computing

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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

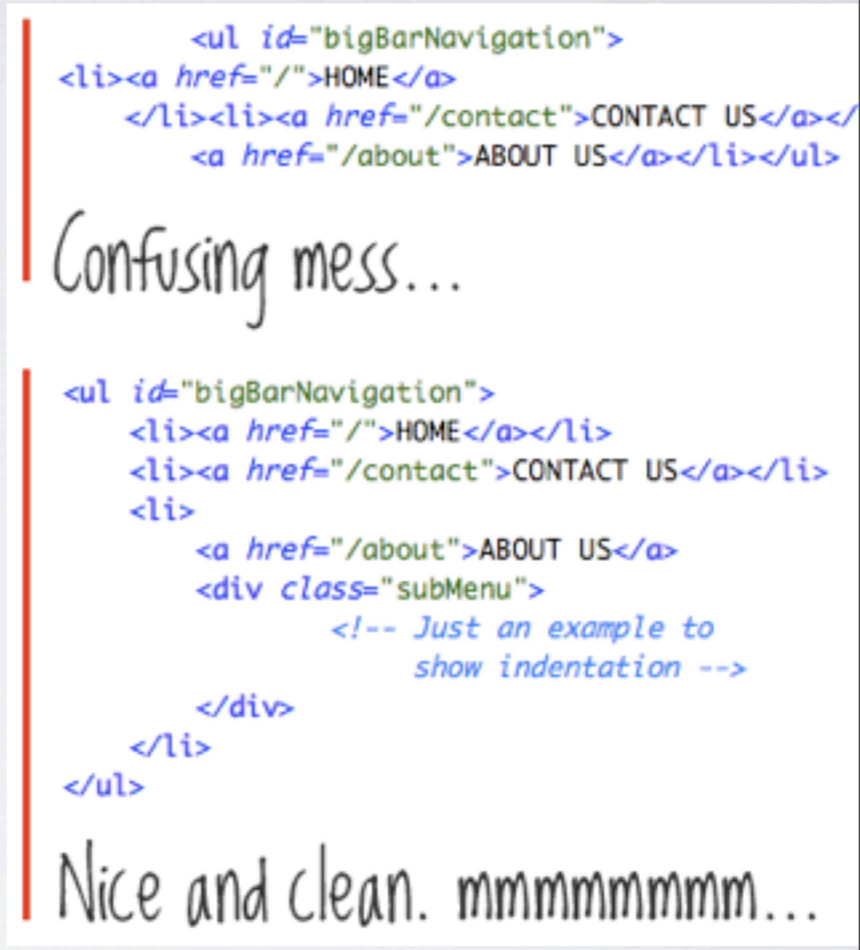
- “being able to use understand and improve your code in 6 months & in 60 years” - approximate Damian Conway
- variable naming

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



```
        <ul id="bigBarNavigation">
<li><a href="/">HOME</a>
    </li><li><a href="/contact">CONTACT US</a></
        <a href="/about">ABOUT US</a></li></ul>
```

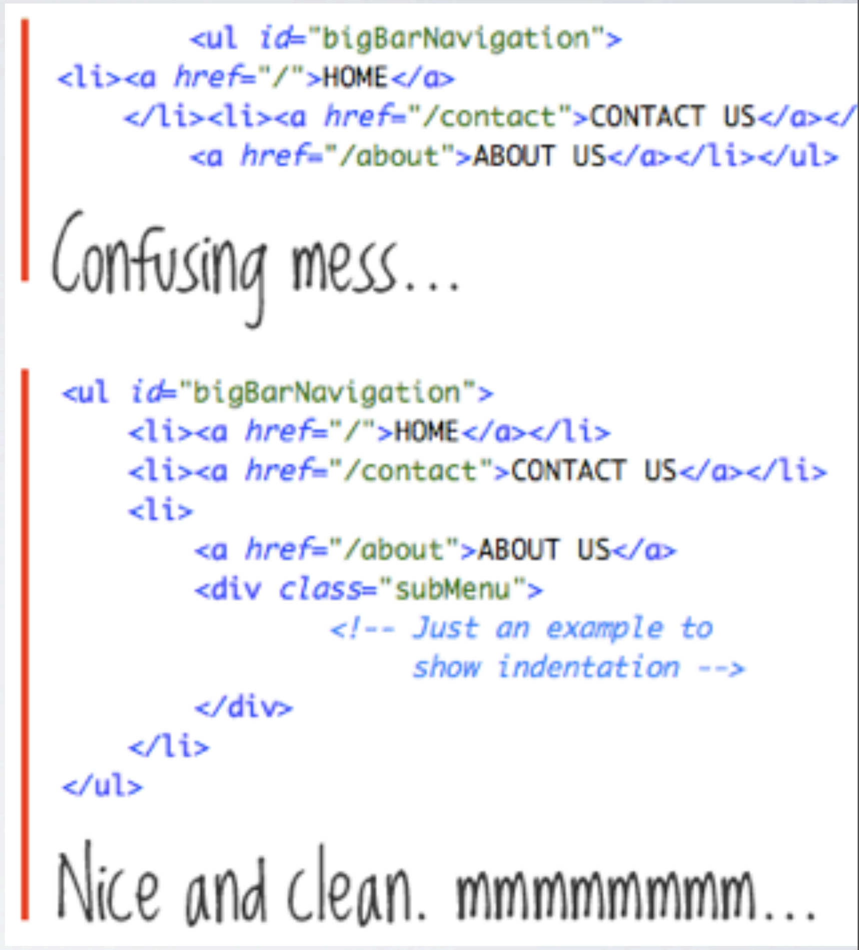
Confusing mess...

```
<ul id="bigBarNavigation">
    <li><a href="/">HOME</a></li>
    <li><a href="/contact">CONTACT US</a></li>
    <li>
        <a href="/about">ABOUT US</a>
        <div class="subMenu">
                <!-- Just an example to
                     show indentation -->
        </div>
    </li>
</ul>
```

Nice and clean. mmmmmmmm...

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



```
        <ul id="bigBarNavigation">
<li><a href="/">HOME</a>
    </li><li><a href="/contact">CONTACT US</a></
        <a href="/about">ABOUT US</a></li></ul>
```

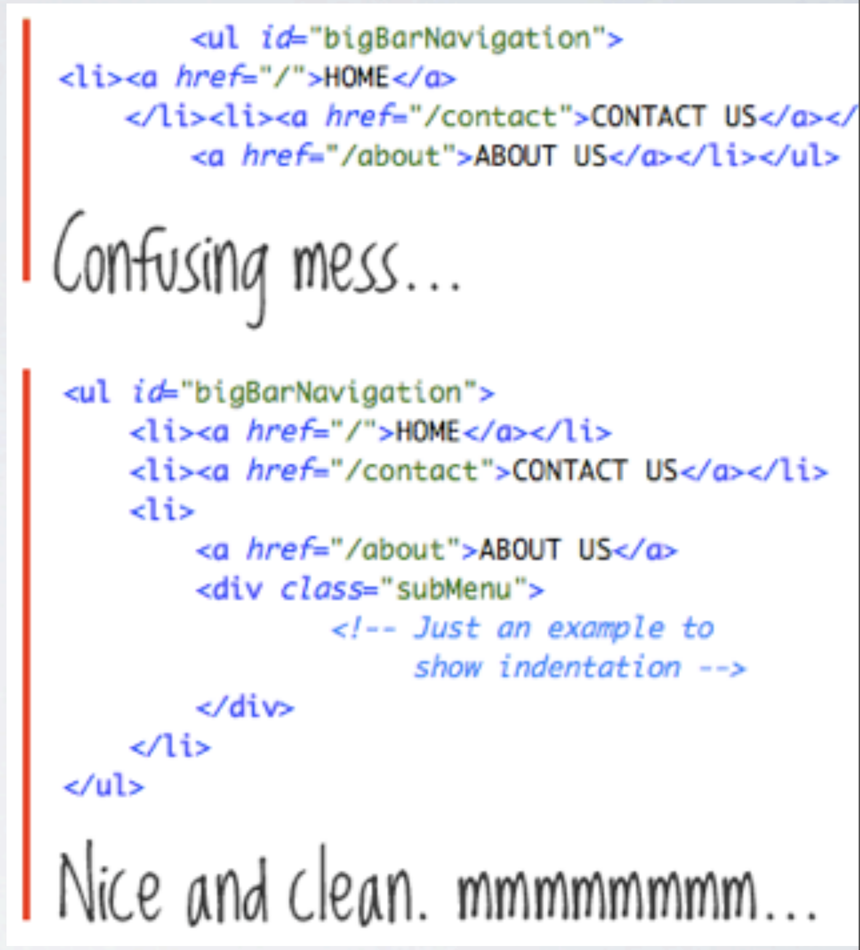
Confusing mess...

```
<ul id="bigBarNavigation">
    <li><a href="/">HOME</a></li>
    <li><a href="/contact">CONTACT US</a></li>
    <li>
        <a href="/about">ABOUT US</a>
        <div class="subMenu">
                <!-- Just an example to
                     show indentation -->
        </div>
    </li>
</ul>
```

Nice and clean. mmmmmmm...

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/



```
        <ul id="bigBarNavigation">
<li><a href="/">HOME</a>
    </li><li><a href="/contact">CONTACT US</a></
        <a href="/about">ABOUT US</a></li></ul>
```

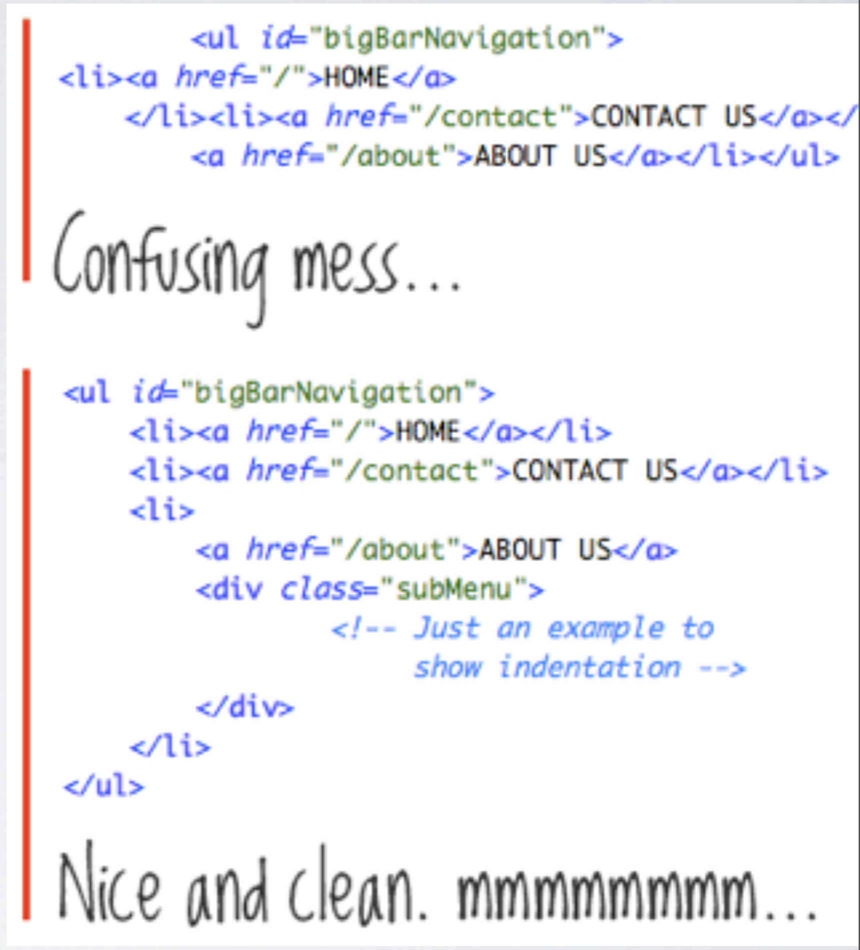
Confusing mess...

```
<ul id="bigBarNavigation">
    <li><a href="/">HOME</a></li>
    <li><a href="/contact">CONTACT US</a></li>
    <li>
        <a href="/about">ABOUT US</a>
        <div class="subMenu">
                <!-- Just an example to
                     show indentation -->
        </div>
    </li>
</ul>
```

Nice and clean. mmmmmmm...

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.:



```
        <ul id="bigBarNavigation">
<li><a href="/">HOME</a>
    </li><li><a href="/contact">CONTACT US</a></
        <a href="/about">ABOUT US</a></li></ul>
```

Confusing mess...

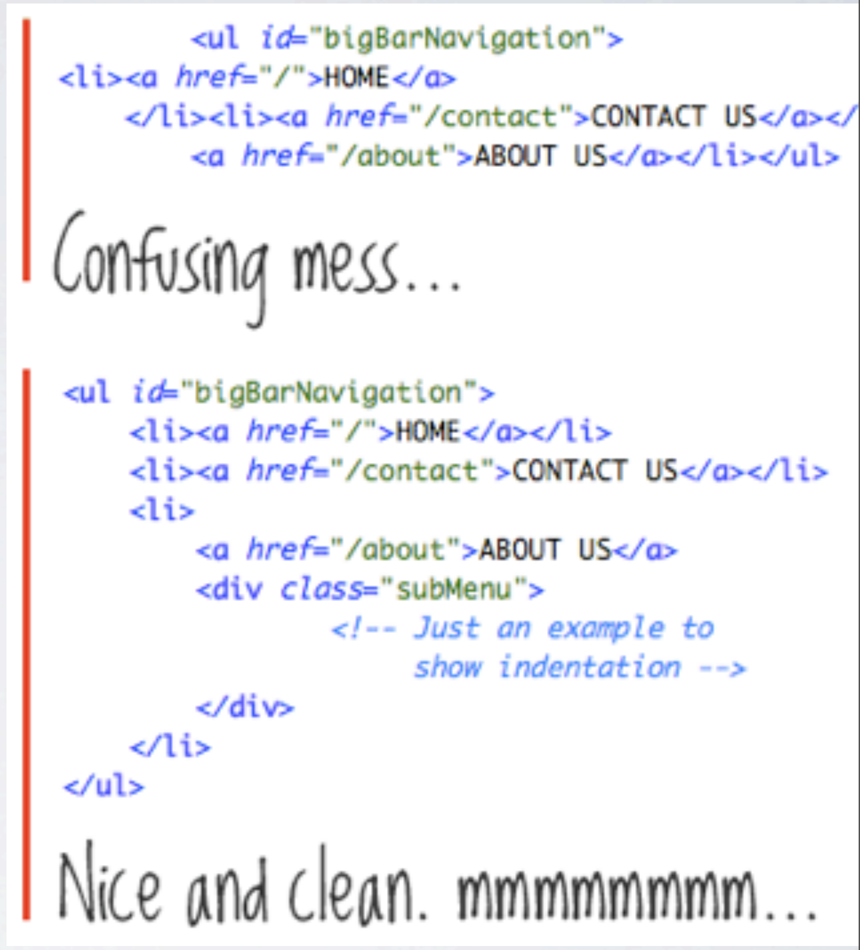
```
<ul id="bigBarNavigation">
    <li><a href="/">HOME</a></li>
    <li><a href="/contact">CONTACT US</a></li>
    <li>
        <a href="/about">ABOUT US</a>
        <div class="subMenu">
                <!-- Just an example to
                     show indentation -->
        </div>
    </li>
</ul>
```

Nice and clean. mmmmmmmm...

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.:

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



```
        <ul id="bigBarNavigation">
<li><a href="/">HOME</a>
    </li><li><a href="/contact">CONTACT US</a></
        <a href="/about">ABOUT US</a></li></ul>
```

Confusing mess...

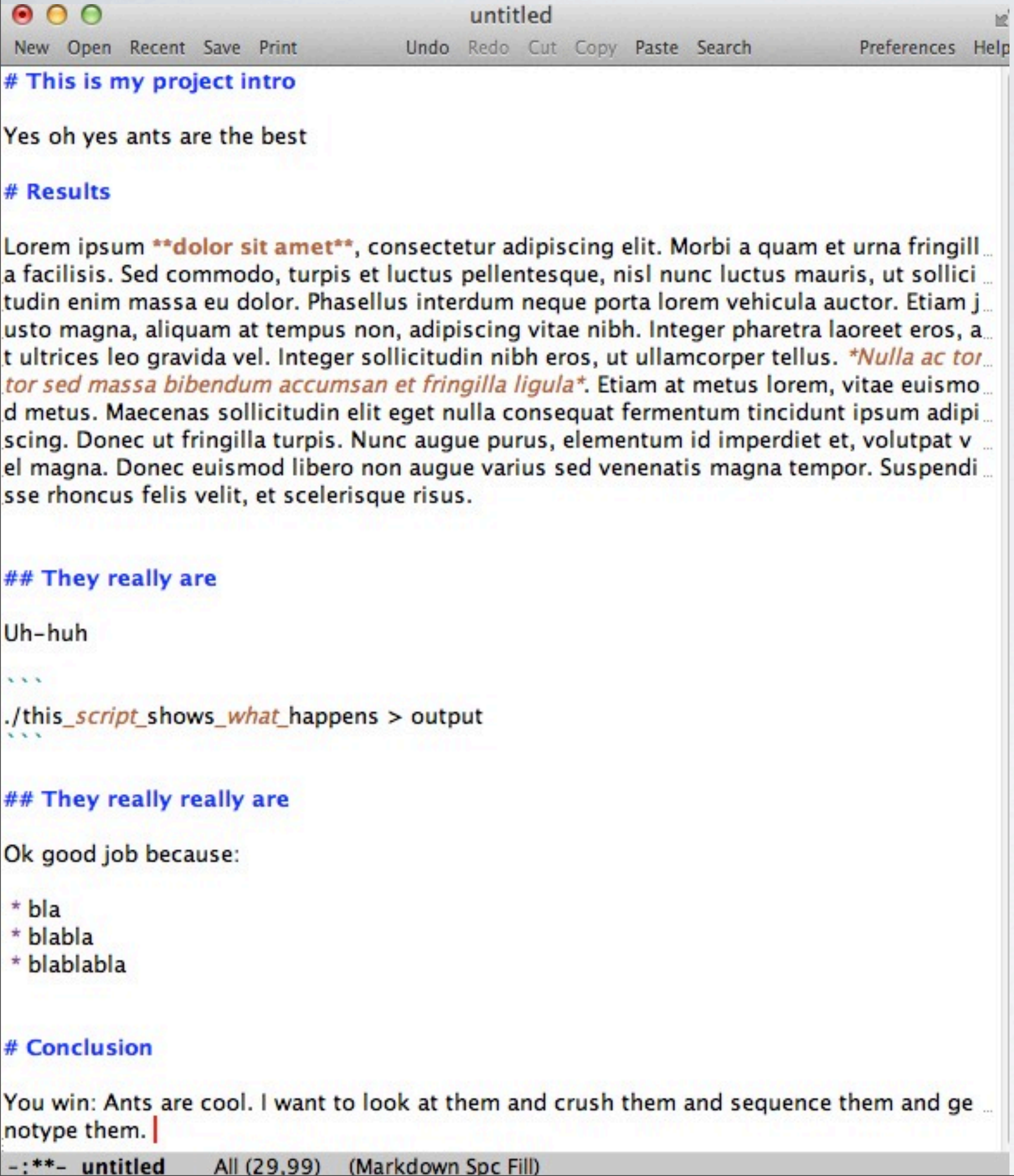
```
<ul id="bigBarNavigation">
    <li><a href="/">HOME</a></li>
    <li><a href="/contact">CONTACT US</a></li>
    <li>
        <a href="/about">ABOUT US</a>
        <div class="subMenu">
                <!-- Just an example to
                     show indentation -->
        </div>
    </li>
</ul>
```

Nice and clean. mmmmmmmm...

A few tools

Take notes in Markdown

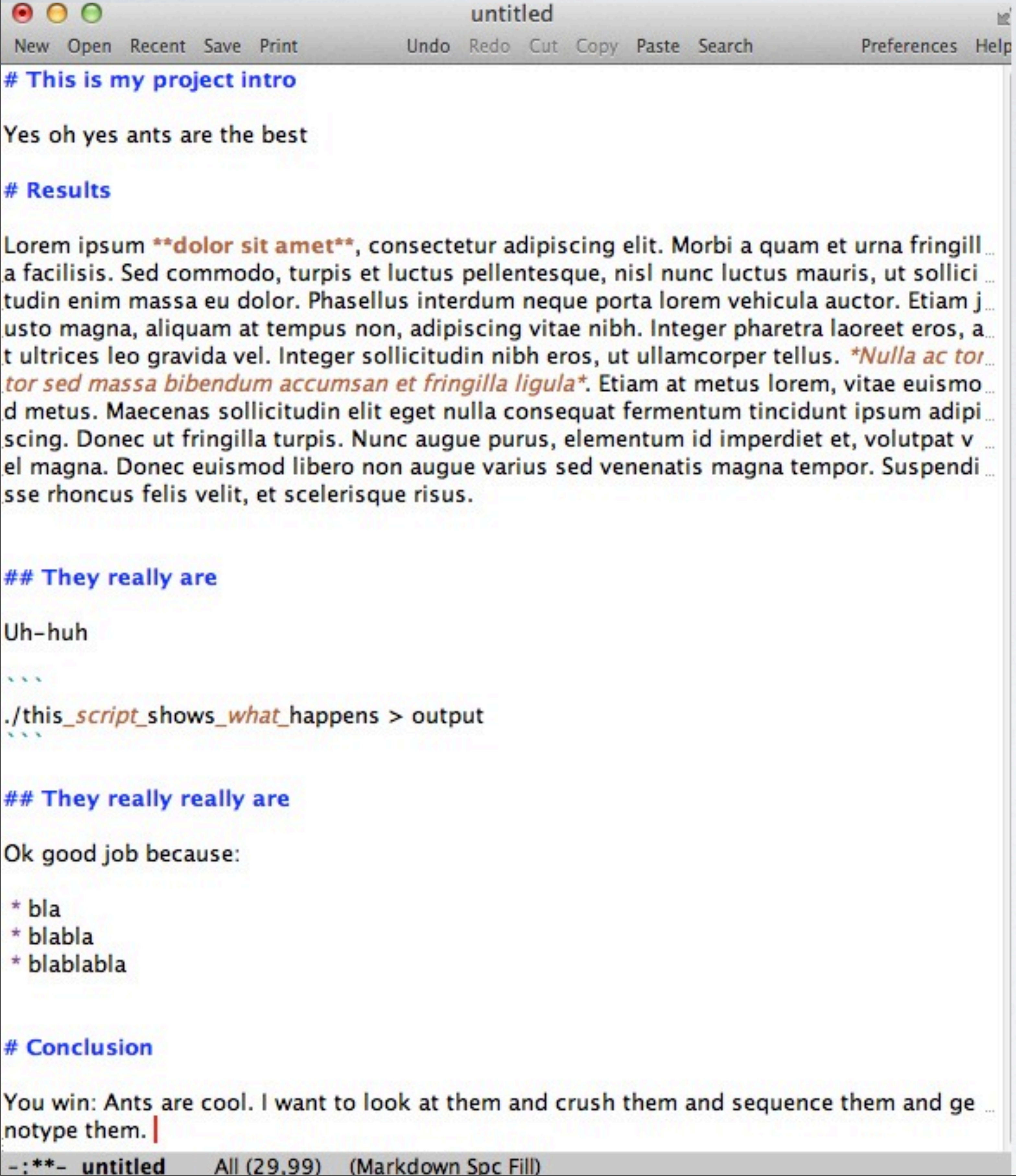
to html, pdf,


untitled
New Open Recent Save Print Undo Redo Cut Copy Paste Search Preferences Help

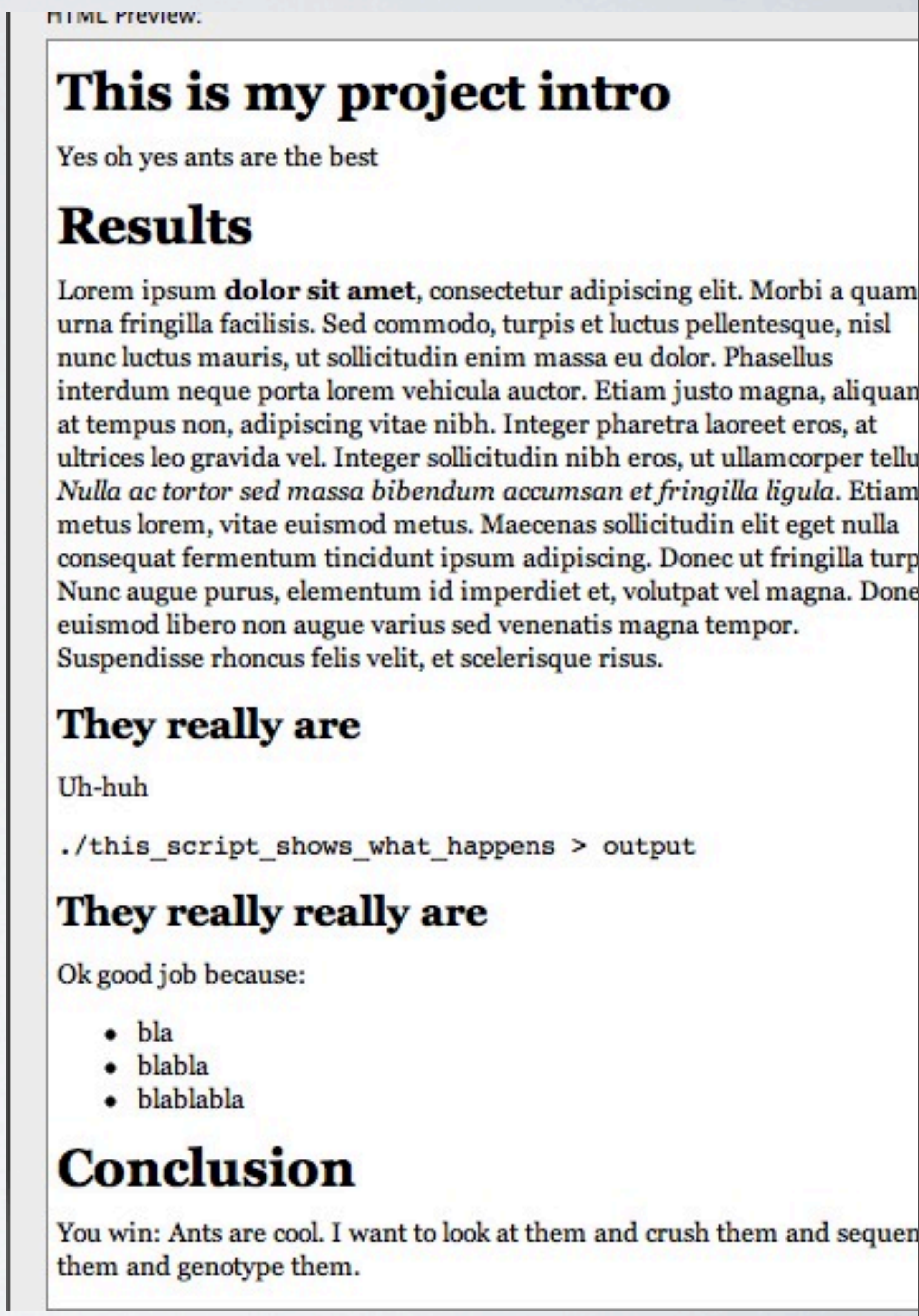
```
# This is my project intro

Yes oh yes ants are the best

# Results

Lorem ipsum **dolor sit amet**, consectetur adipiscing elit. Morbi a quam et urna fringill
a facilisis. Sed commodo, turpis et luctus pellentesque, nisl nunc luctus mauris, ut sollici
tudin enim 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.
-:**- untitled   All (29,99)   (Markdown Spc Fill)
```

Take notes in Markdown

to html, pdf,



```
# 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 fringill
a facilisis. Sed commodo, turpis et luctus pellentesque, nisl nunc luctus mauris, ut sollici
tudin enim 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.
-:**- untitled   All (29,99)   (Markdown Spc Fill)
```



This is my project intro

Yes oh yes ants are the best

Results

Lorem ipsum **dolor sit amet**, consectetur adipiscing elit. Morbi a quam
urna fringilla facilisis. Sed commodo, turpis et luctus pellentesque, nisl
nunc luctus mauris, ut sollicitudin enim massa eu dolor. Phasellus
interdum neque porta lorem vehicula auctor. Etiam justo magna, aliquan
at tempus non, adipiscing vitae nibh. Integer pharetra laoreet eros, at
ultrices leo gravida vel. Integer sollicitudin nibh eros, ut ullamcorper tellu
Nulla ac tortor sed massa bibendum accumsan et fringilla ligula. Etiam
metus lorem, vitae euismod metus. Maecenas sollicitudin elit eget nulla
consequat fermentum tincidunt ipsum adipiscing. Donec ut fringilla turp
Nunc augue purus, elementum id imperdiet et, volutpat vel magna. Done
euismod 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
- blablabla

Conclusion

You win: Ants are cool. I want to look at them and crush them and sequen
them and genotype them.

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}

\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{}
```

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

MyFile.Rnw

```
\documentclass{article}
\usepackage[sc]{mathpazo}
\usepackage[T1]{fontenc}
\usepackage{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{}
```

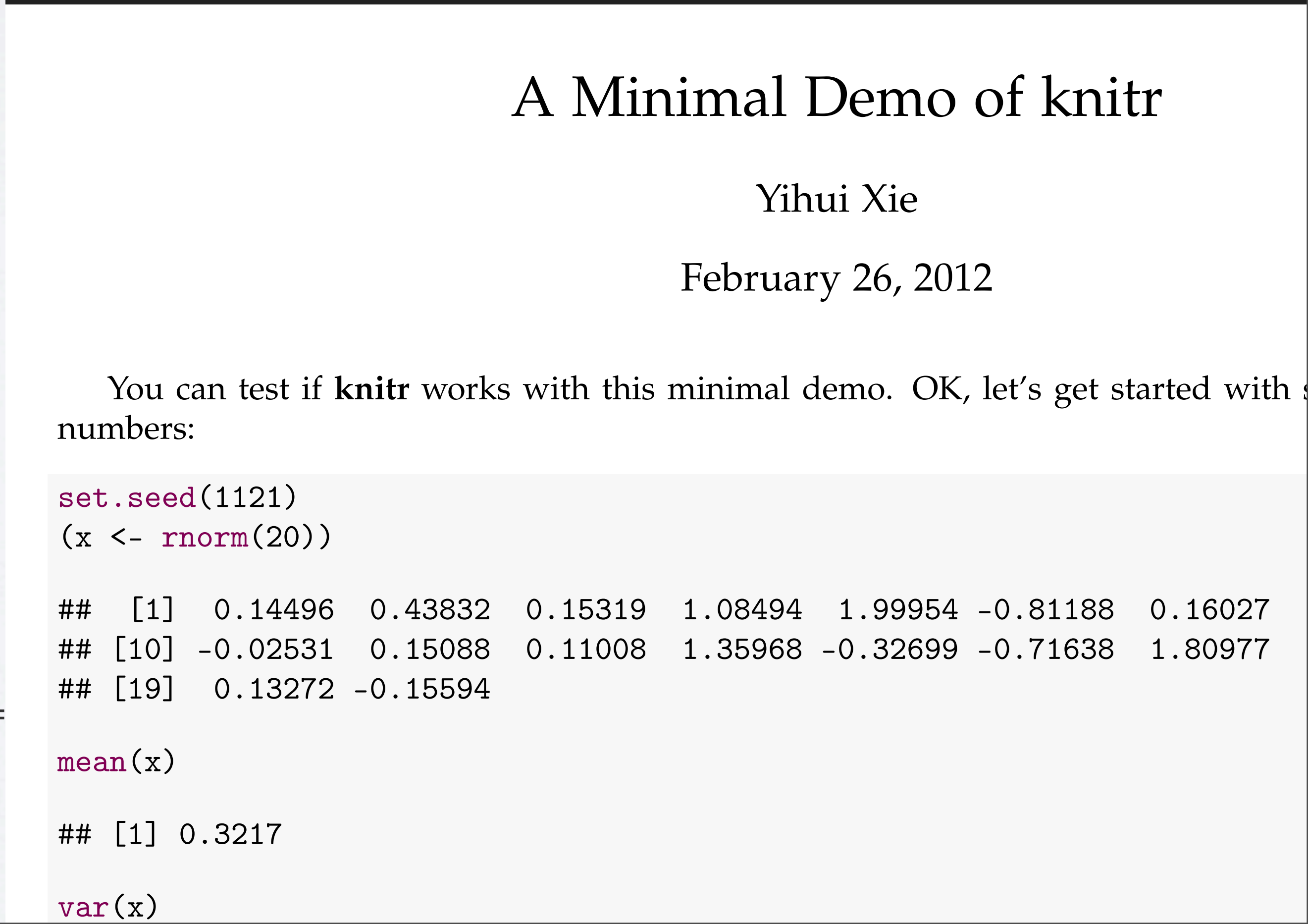
```
### in R:
library(knitr)
knit("MyFile.Rnw")
# --> creates  MyFile.tex

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

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

MyFile.Rnw

```
\documentclass{article}
\usepackage[sc]{mathpazo}
\usepackage[T1]{fontenc}
\usepackage{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=
boxplot(x)
hist(x,main='')
@

Do the above chunks work? You should be able to compile the \TeX{}
```

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

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



A Minimal Demo of knitr

Yihui Xie

February 26, 2012

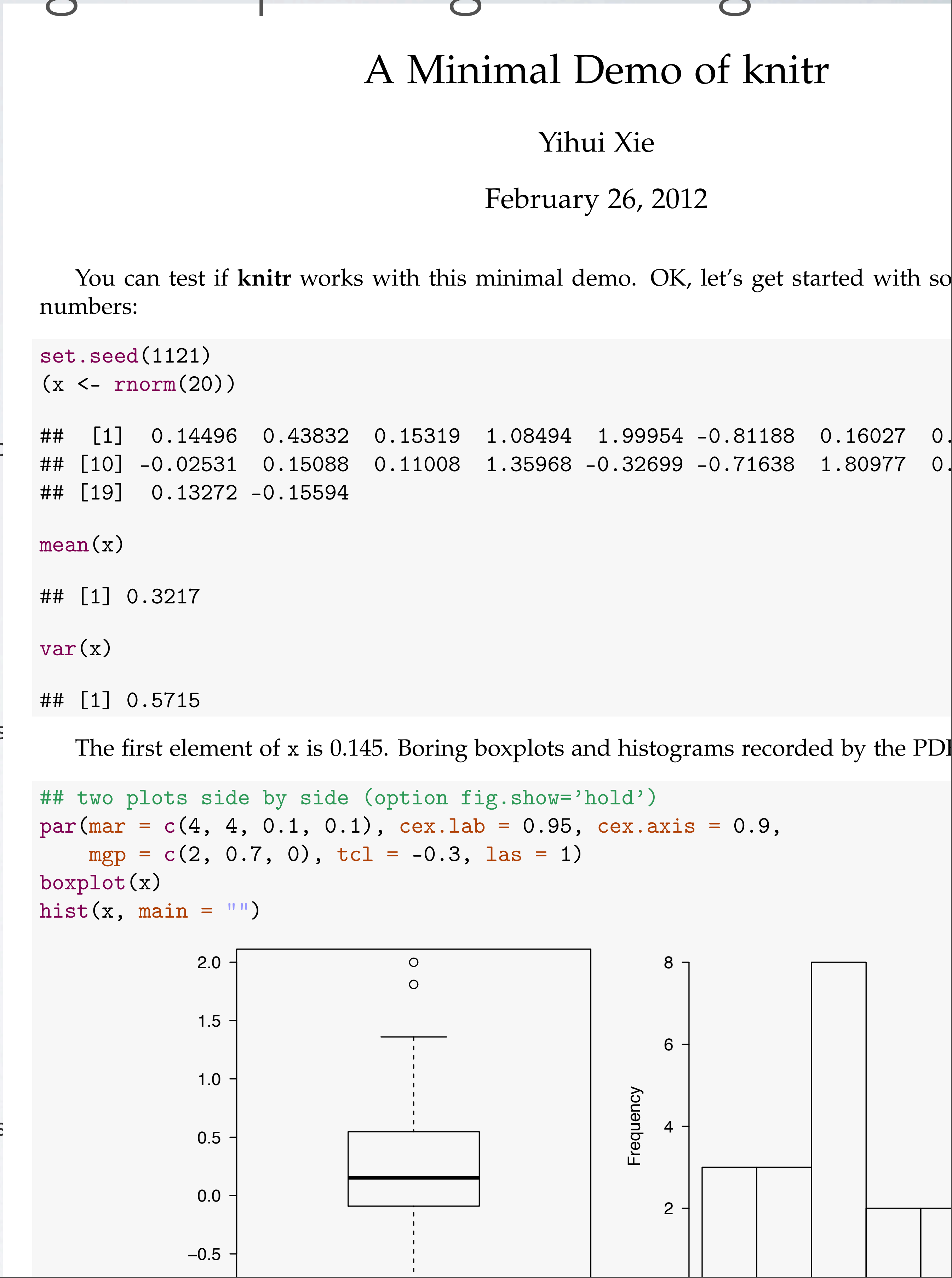
You can test if **knitr** works with this minimal demo. OK, let's get started with
numbers:

```
set.seed(1121)
(x <- rnorm(20))

##  [1]  0.14496  0.43832  0.15319  1.08494  1.99954 -0.81188  0.16027
## [10] -0.02531  0.15088  0.11008  1.35968 -0.32699 -0.71638  1.80977
## [19]  0.13272 -0.15594

mean(x)

## [1] 0.3217

var(x)
```

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

MyFile.Rnw

```
\documentclass{article}
\usepackage[sc]{mathpazo}
\usepackage[T1]{fontenc}
\usepackage{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.sho
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
boxplot(x)
hist(x,main='')
@

Do the above chunks work? You should be able to compile the \TeX{}
```



A Minimal Demo of knitr

Yihui Xie

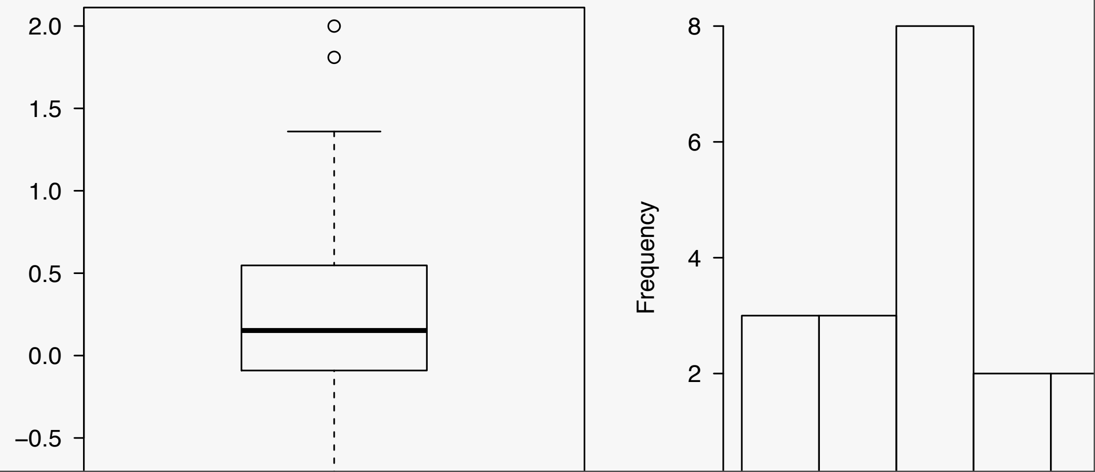
February 26, 2012

You can test if **knitr** works with this minimal demo. OK, let's get started with so
numbers:

```
set.seed(1121)
(x <- rnorm(20))

##  [1]  0.14496  0.43832  0.15319  1.08494  1.99954 -0.81188  0.16027  0.
## [10] -0.02531  0.15088  0.11008  1.35968 -0.32699 -0.71638  1.80977  0.
## [19]  0.13272 -0.15594

mean(x)

## [1] 0.3217

var(x)

## [1] 0.5715
```

The first element of x is 0.145. Boring boxplots and histograms recorded by the PDI

```
## two plots side by side (option fig.show='hold')
par(mar = c(4, 4, 0.1, 0.1), cex.lab = 0.95, cex.axis = 0.9,
    mgp = c(2, 0.7, 0), tcl = -0.3, las = 1)
boxplot(x)
hist(x, main = "")
```



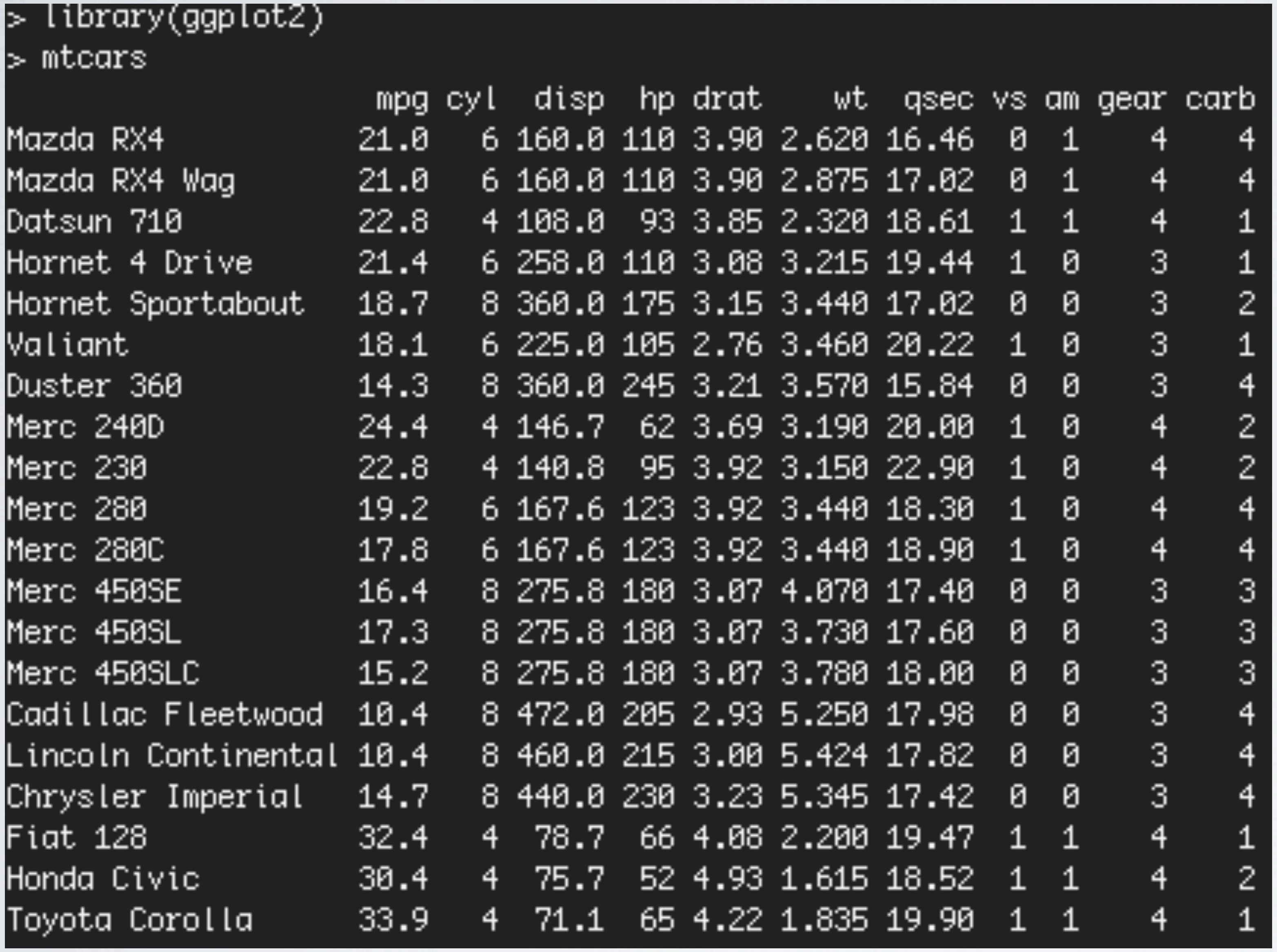
Plotting in R

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.

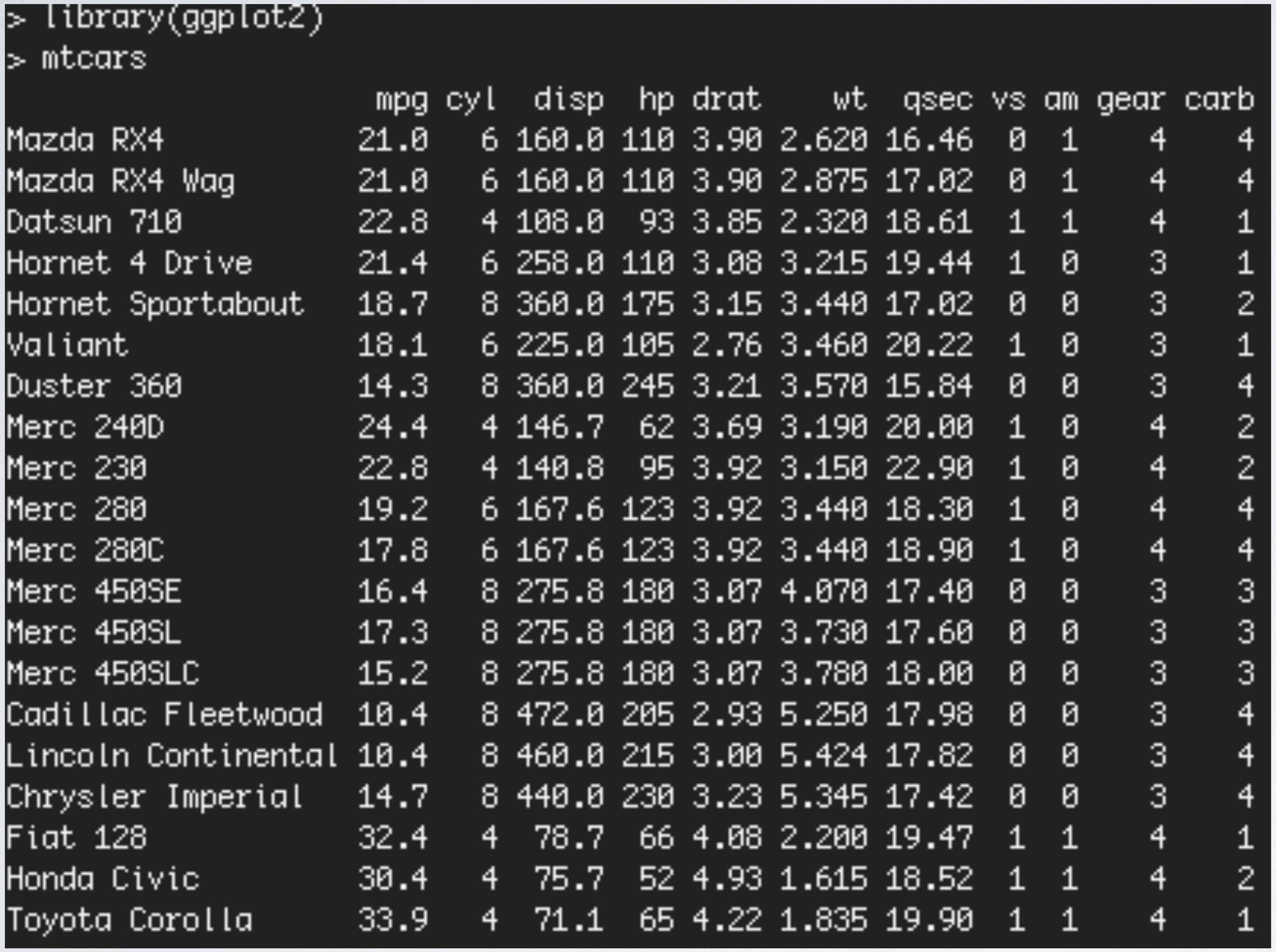
- --> Need for something beautiful, easy & effortless.

ggplot2: beautiful & (almost) effortless R plots

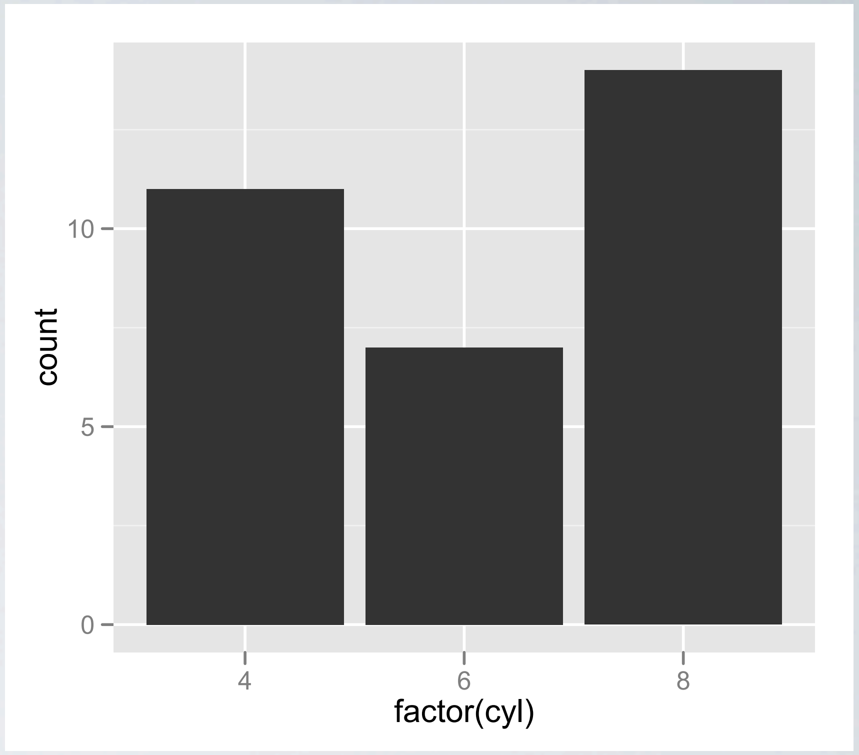


```
> library(ggplot2)
> mtcars
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
```

ggplot2: beautiful & (almost) effortless R plots

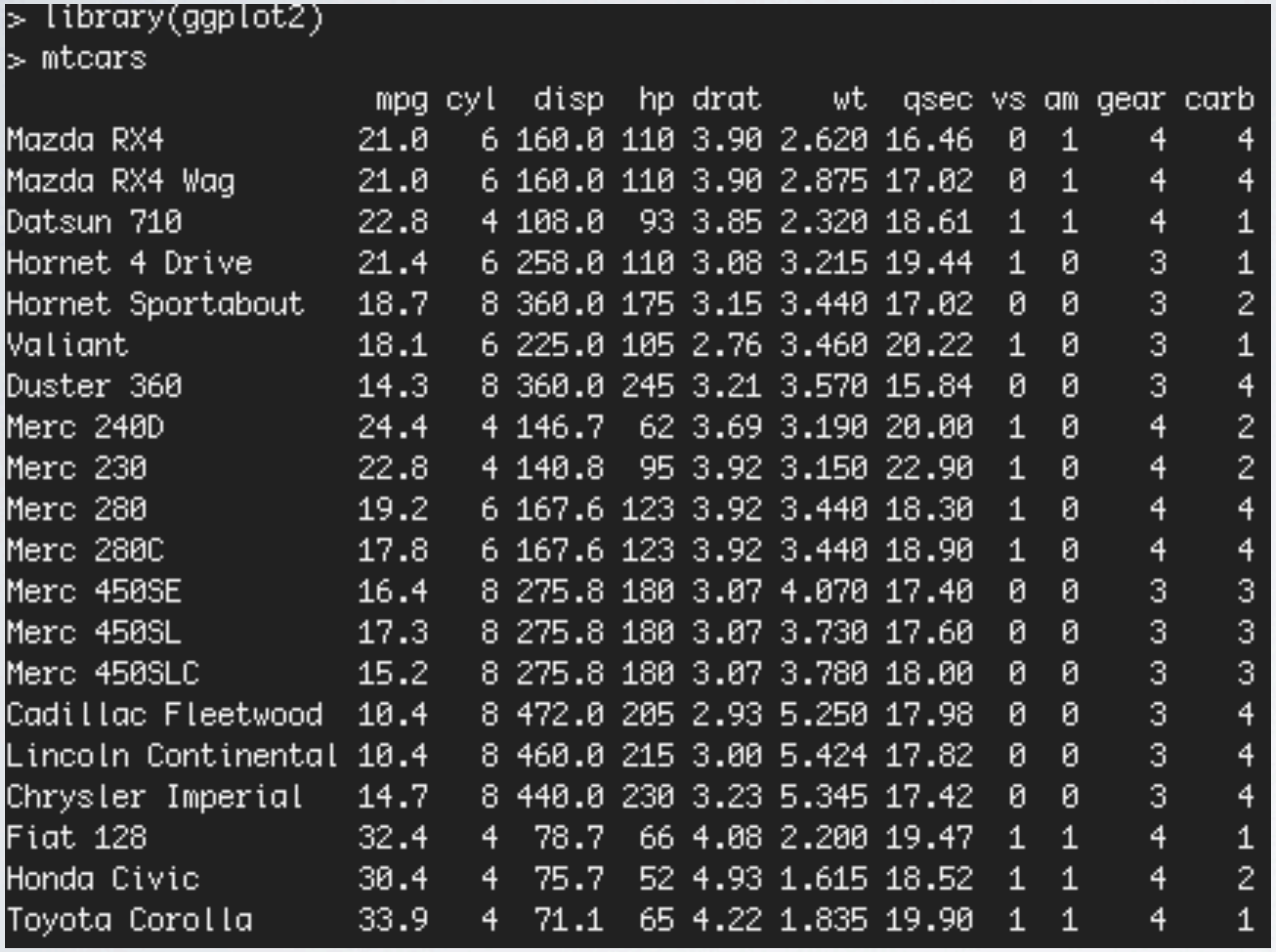


```
> library(ggplot2)
> mtcars
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
```

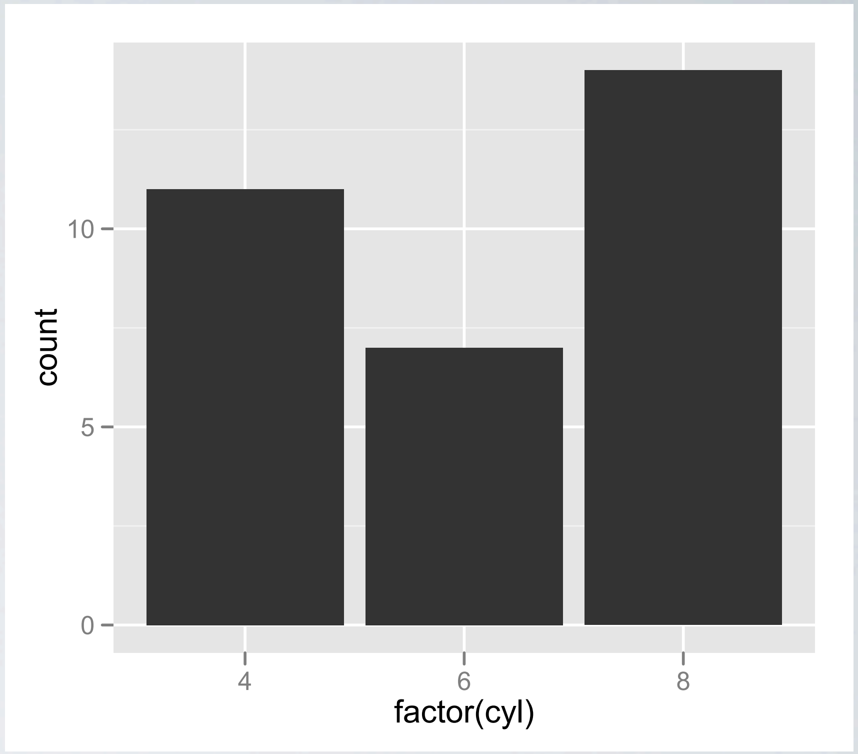


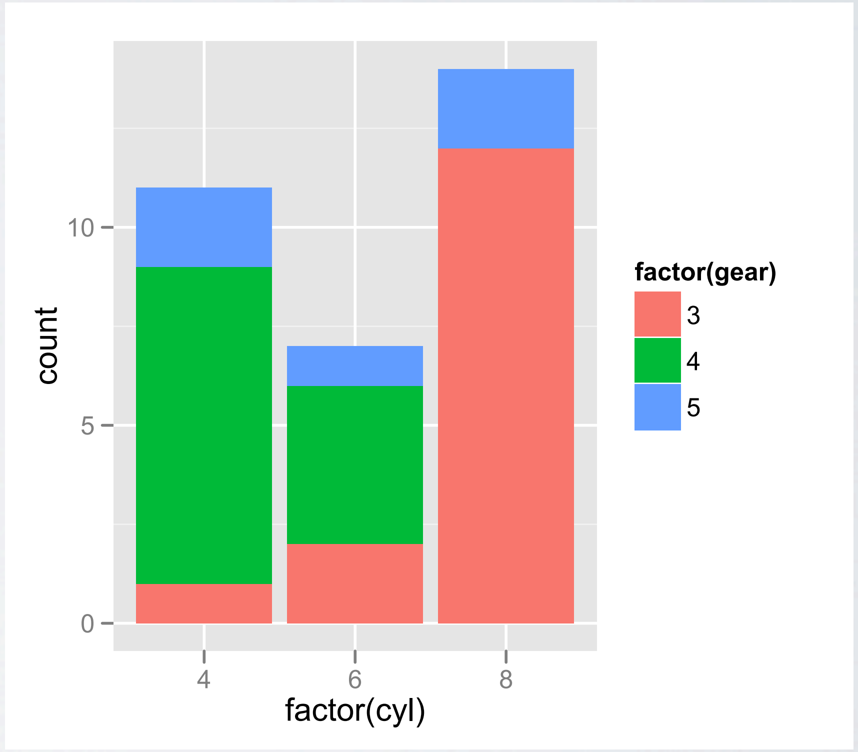
```
ggplot(mtcars, aes(factor(cyl))) + geom_bar()
```

ggplot2: beautiful & (almost) effortless R plots

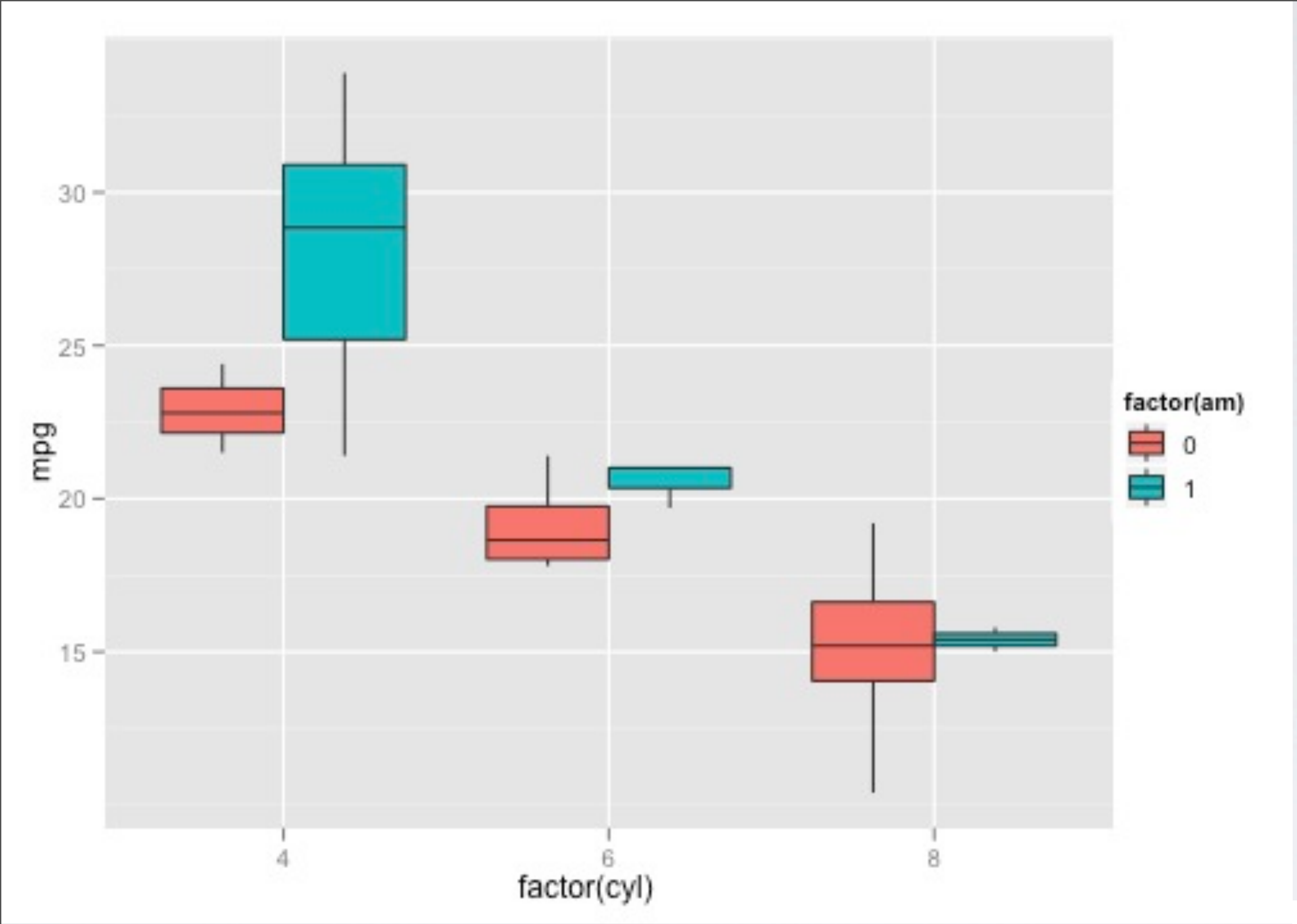


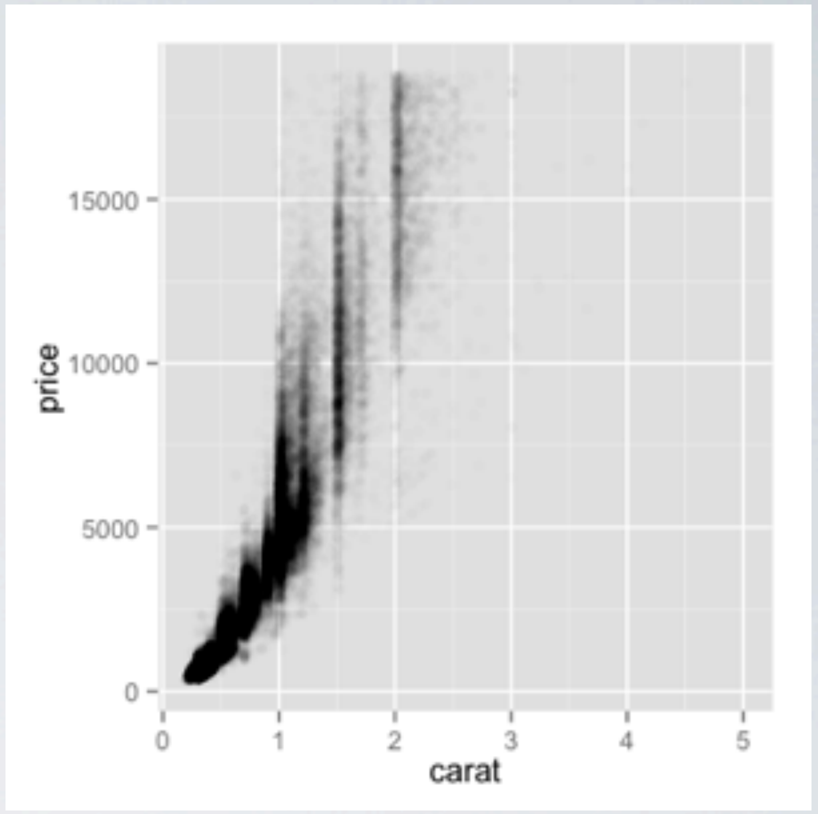
```
> library(ggplot2)
> mtcars
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
```

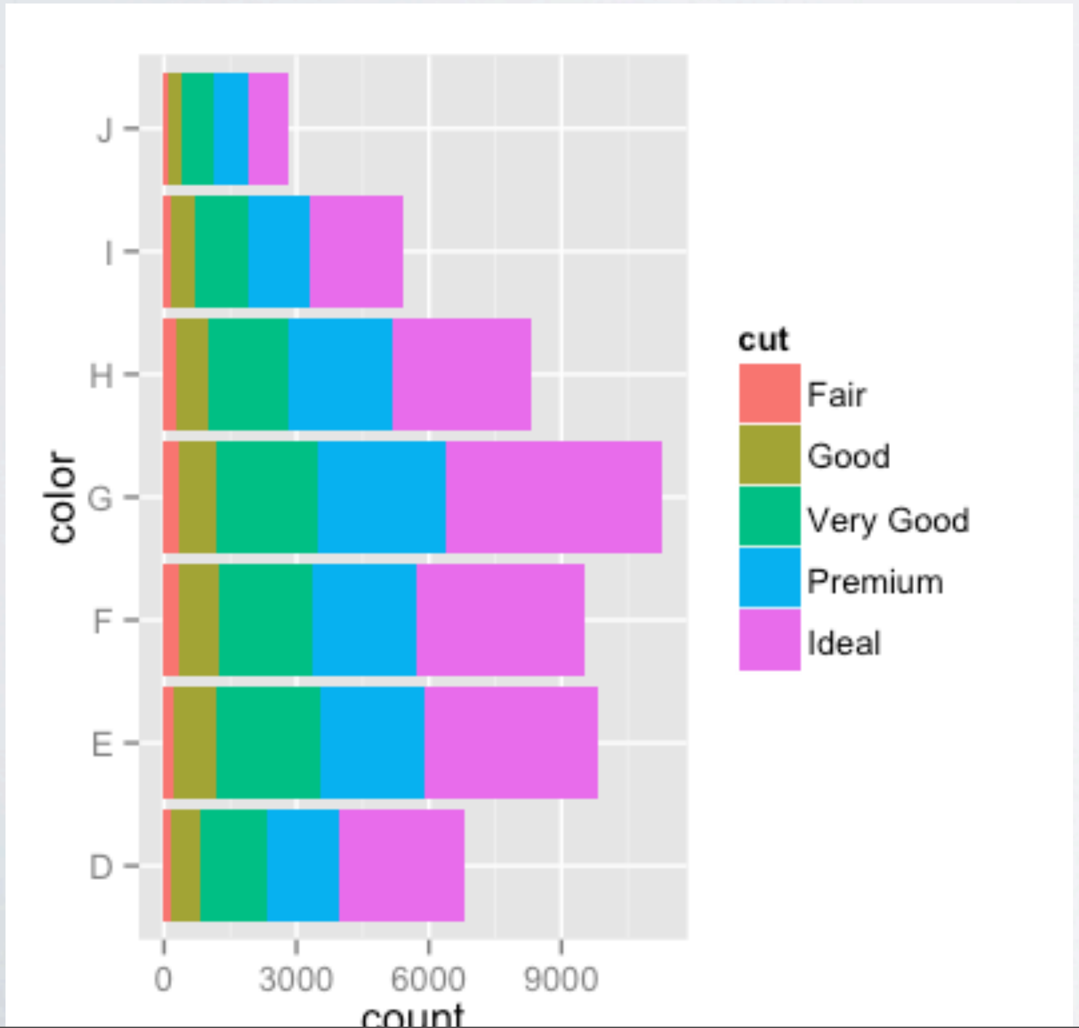


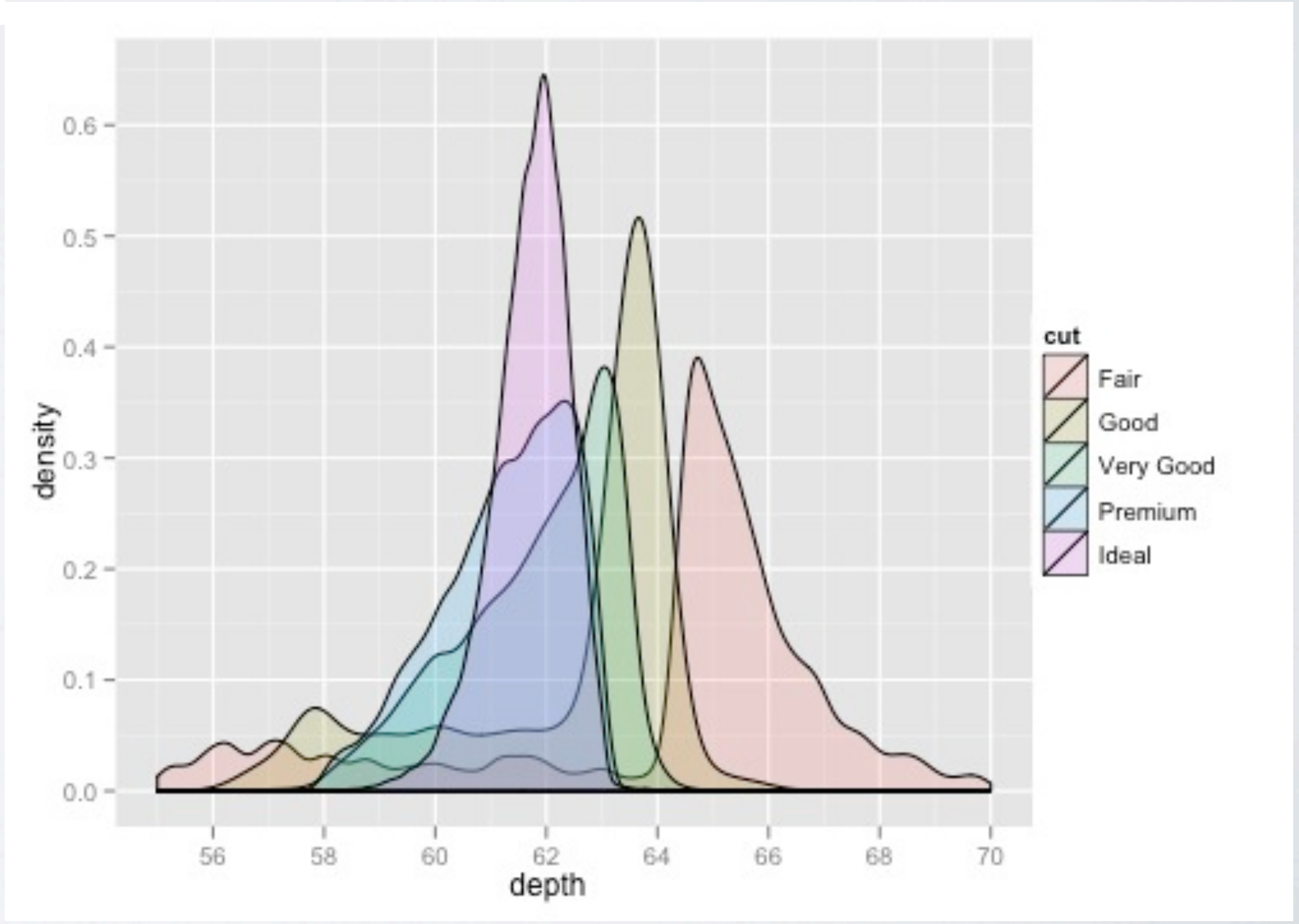


```
ggplot(mtcars, aes(factor(cyl))) + geom_bar()
ggplot(mtcars, aes(factor(cyl), fill=factor(gear))) + geom_bar()
```


mpg
30
25
20
15
factor(cyl)
4
6
8
factor(am)
0
1

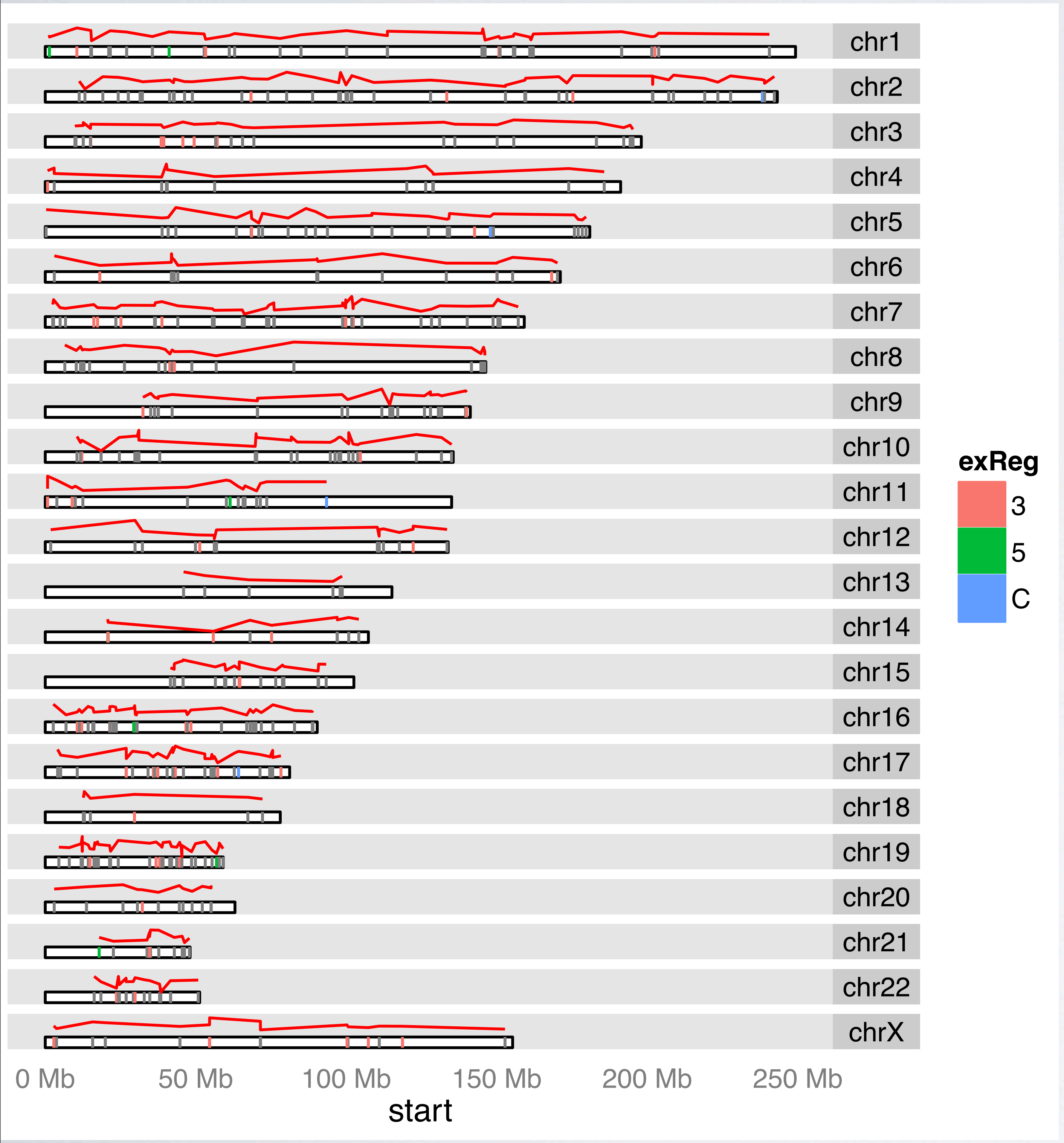

price
15000
10000
5000
0
carat
0
1
2
3
4
5

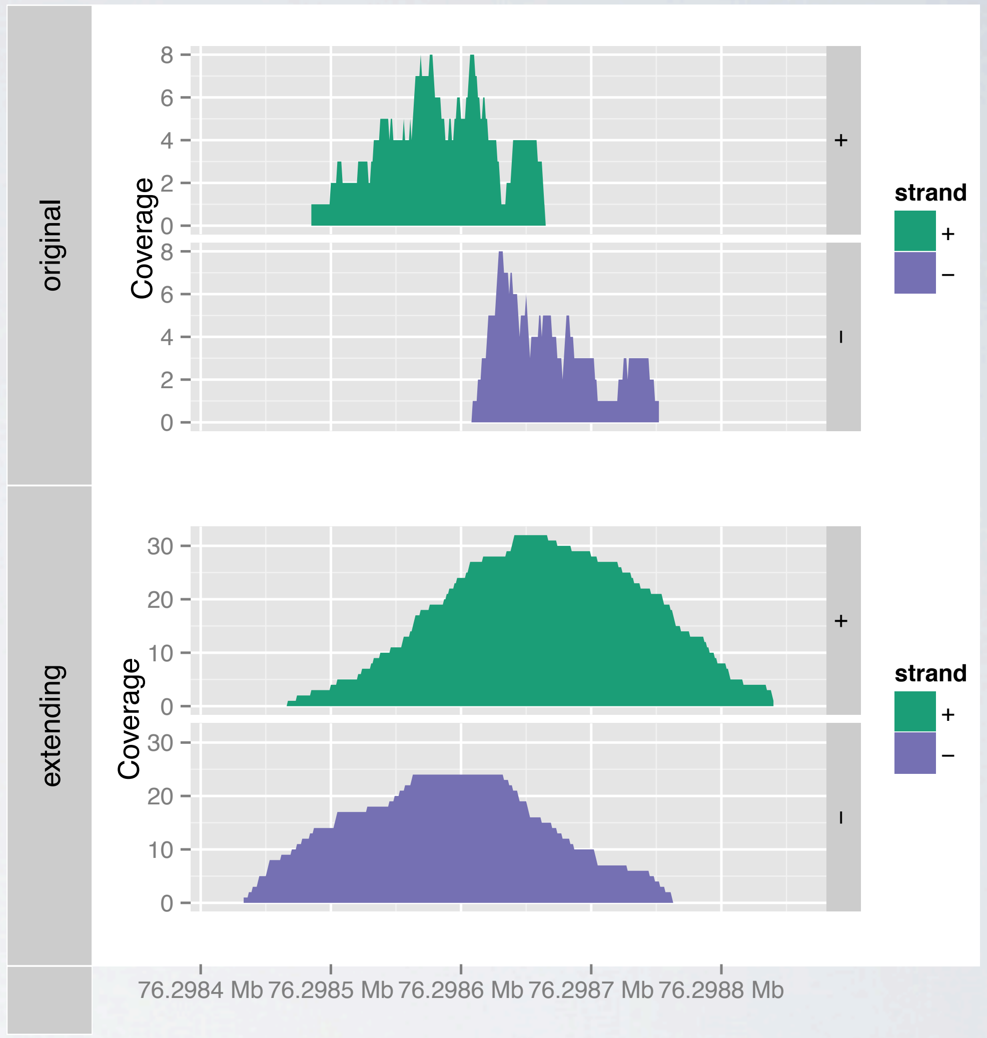

color
J
I
H
G
F
E
D
count
0
3000
6000
9000
cut
Fair
Good
Very Good
Premium
Ideal

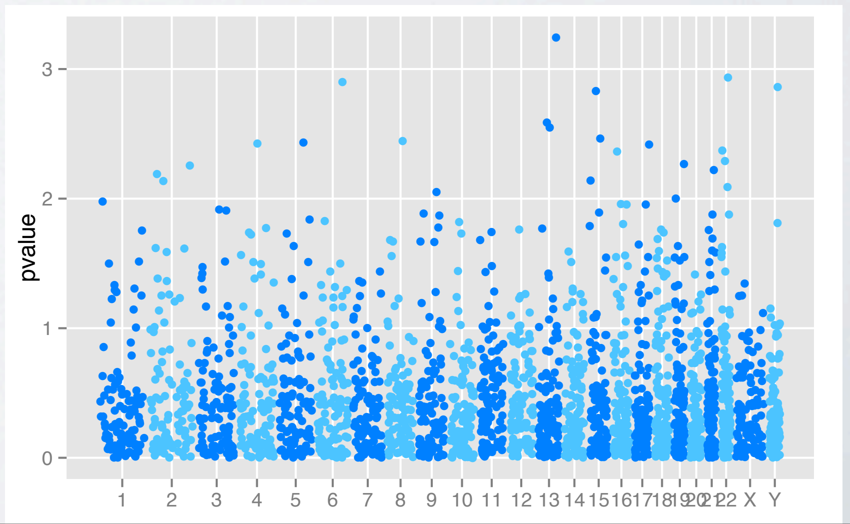

density
0.6
0.5
0.4
0.3
0.2
0.1
0.0
depth
56
58
60
62
64
66
68
70
cut
Fair
Good
Very Good
Premium
Ideal

SOFTWARE **Open Access**

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.

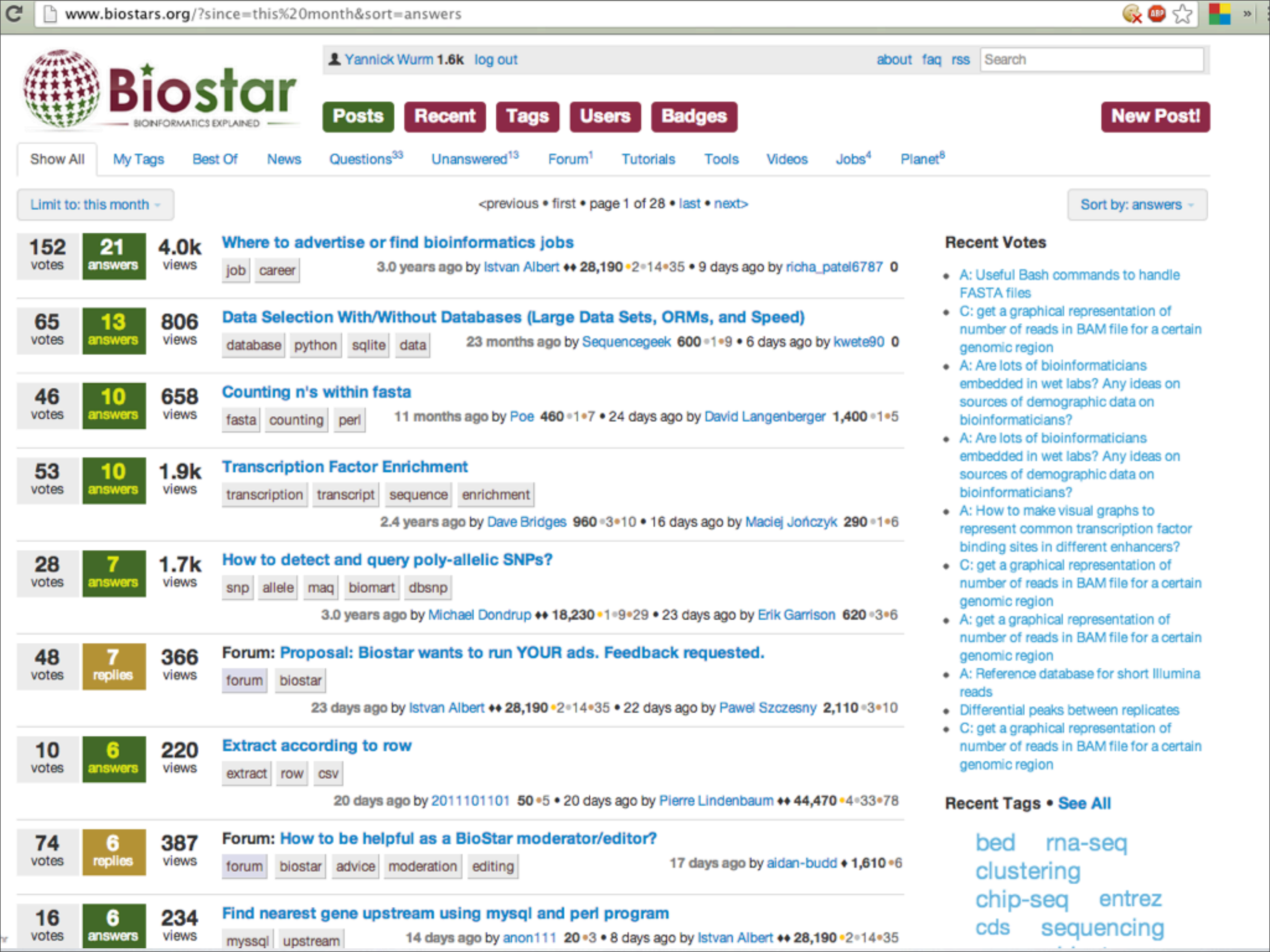
- In real life: Make friends with people. Talk to them.

- 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



www.biostars.org/?since=this%20month&sort=answers

Yannick Wurm 1.6k log out

about faq rss Search

Biostar BIOINFORMATICS EXPLAINED

Posts Recent Tags Users Badges New Post!

Show All My Tags Best Of News Questions33 Unanswered13 Forum1 Tutorials Tools Videos Jobs4 Planet8

Limit to: this month

<previous • first • page 1 of 28 • last • next>

Sort by: answers

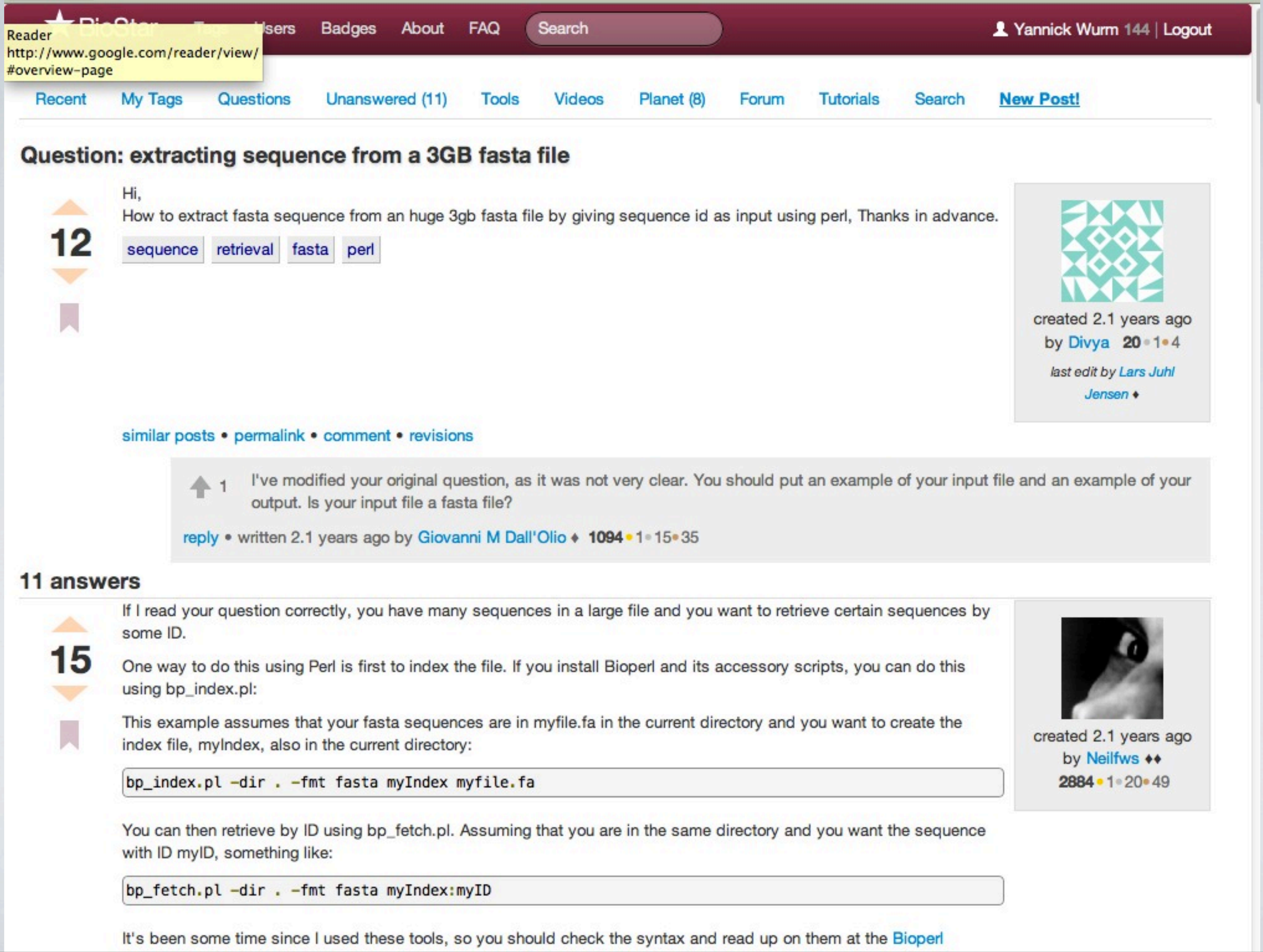
votes	answers/replies	views	post
152	21 answers	4.0k	Where to advertise or find bioinformatics jobs — job career — 3.0 years ago by Istvan Albert ♦♦ 28,190 •2•14•35 • 9 days ago by richa_patel6787 0
65	13 answers	806	Data Selection With/Without Databases (Large Data Sets, ORMs, and Speed) — database python sqlite data — 23 months ago by Sequencegeek 600 •1•9 • 6 days ago by kwete90 0
46	10 answers	658	Counting n's within fasta — fasta counting perl — 11 months ago by Poe 460 •1•7 • 24 days ago by David Langenberger 1,400 •1•5
53	10 answers	1.9k	Transcription Factor Enrichment — transcription transcript sequence enrichment — 2.4 years ago by Dave Bridges 960 •3•10 • 16 days ago by Maciej Jończyk 290 •1•6
28	7 answers	1.7k	How to detect and query poly-allelic SNPs? — snp allele maq biomart dbsnp — 3.0 years ago by Michael Dondrup ♦♦ 18,230 •1•9•29 • 23 days ago by Erik Garrison 620 •3•6
48	7 replies	366	Forum: Proposal: Biostar wants to run YOUR ads. Feedback requested. — forum biostar — 23 days ago by Istvan Albert ♦♦ 28,190 •2•14•35 • 22 days ago by Pawel Szczesny 2,110 •3•10
10	6 answers	220	Extract according to row — extract row csv — 20 days ago by 2011101101 50 •5 • 20 days ago by Pierre Lindenbaum ♦♦ 44,470 •4•33•78
74	6 replies	387	Forum: How to be helpful as a BioStar moderator/editor? — forum biostar advice moderation editing — 17 days ago by aidan-budd ♦ 1,610 •6
16	6 answers	234	Find nearest gene upstream using mysql and perl program — mysql upstream — 14 days ago by anon111 20 •3 • 8 days ago by Istvan Albert ♦♦ 28,190 •2•14•35

Recent Votes

- A: Useful Bash commands to handle FASTA files
- C: get a graphical representation of number of reads in BAM file for a certain genomic region
- A: Are lots of bioinformaticians embedded in wet labs? Any ideas on sources of demographic data on bioinformaticians?
- A: Are lots of bioinformaticians embedded in wet labs? Any ideas on sources of demographic data on bioinformaticians?
- A: How to make visual graphs to represent common transcription factor binding sites in different enhancers?
- C: get a graphical representation of number of reads in BAM file for a certain genomic region
- A: get a graphical representation of number of reads in BAM file for a certain genomic region
- A: Reference database for short Illumina reads
- Differential peaks between replicates
- C: get a graphical representation of number of reads in BAM file for a certain genomic region

Recent Tags • See All

bed rna-seq clustering chip-seq entrez cds sequencing



Reader
http://www.google.com/reader/view/#overview-page

Users Badges About FAQ Search
Yannick Wurm 144 | Logout
Recent My Tags Questions Unanswered (11) Tools Videos Planet (8) Forum Tutorials Search New Post!

Question: extracting sequence from a 3GB fasta file

12

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

sequence retrieval fasta perl

created 2.1 years ago by Divya 20 • 1 • 4

last edit by Lars Juhl Jensen ♦

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

15

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:

```
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:

```
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

created 2.1 years ago by Neilfws ♦♦ 2884 • 1 • 20 • 49

- 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
```

http://www.sequenceserver.com/

1. Installing

```
gem install sequenceserver
```

2. Configure.

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

http://www.sequenceserver.com/

1. Installing

```
gem install sequenceserver
```

2. Configure.

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

3. Launch.

```
sequenceserver
###   Launched SequenceServer at: http://0.0.0.0:4567
```

http://www.sequenceserver.com/

1. Installing

```
gem install sequenceserver
```

(requires a BLAST+ install)

Do you have BLAST-formatted databases? If not:

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

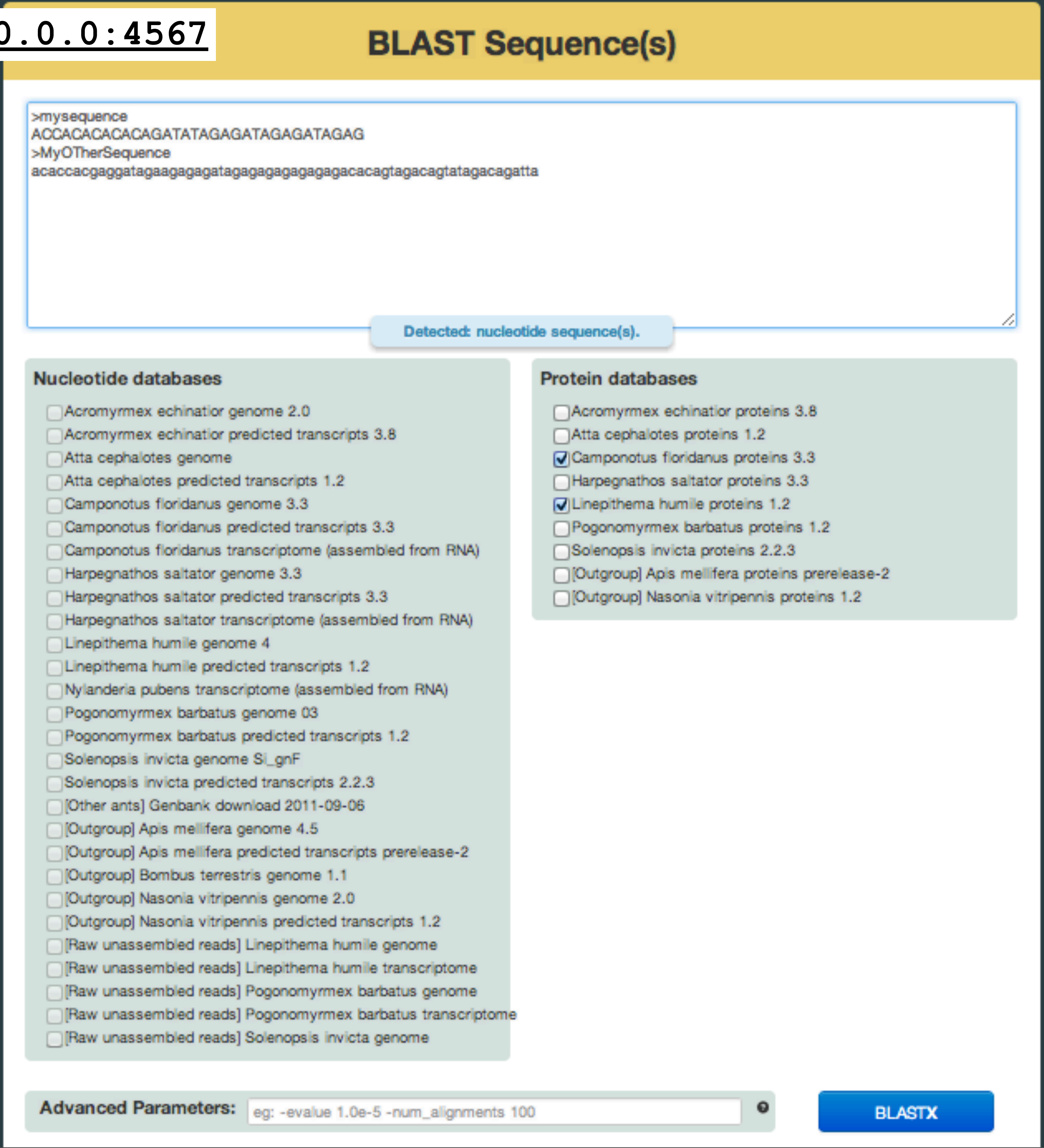
2. Configure.

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

3. Launch.

```
sequenceserver
###   Launched SequenceServer at: http://0.0.0.0:4567
```

http://0.0.0.0:4567



BLAST Sequence(s)

```
>mysequence
ACCACACACACAGATATAGAGATAGAGATAGAG
>MyOTherSequence
acaccacgaggatagaagagagatagagagagagagacacagtagacagtatagacagatta
```

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 Si_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
- [ ] [Outgroup] Nasonia vitripennis predicted transcripts 1.2
- [ ] [Raw unassembled reads] Linepithema humile genome
- [ ] [Raw unassembled reads] Linepithema humile transcriptome
- [ ] [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

Lets 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

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

Best Practices for Scientific Computing

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

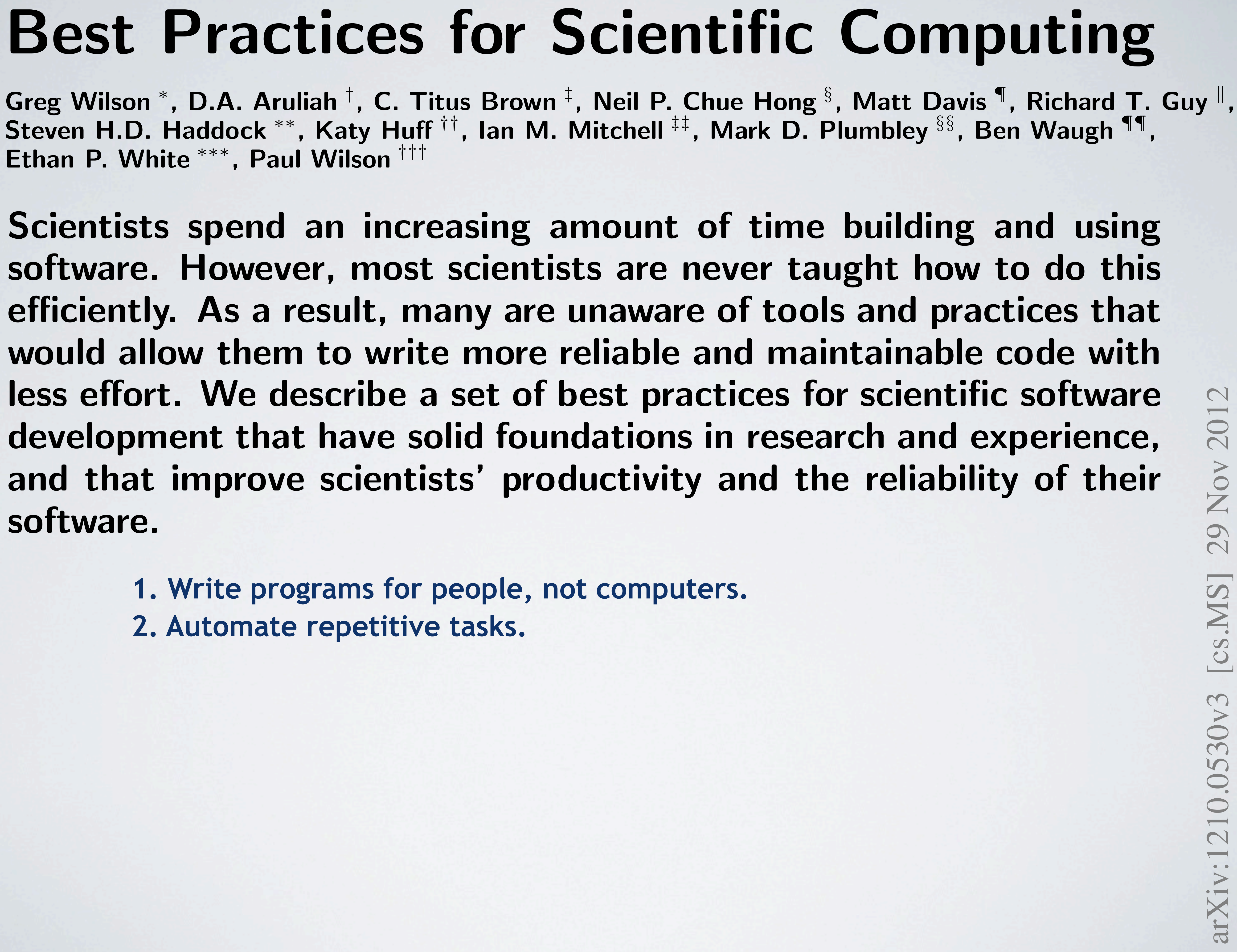
Scientists spend an increasing amount of time building and using software. However, most scientists are never taught how to do this efficiently. As a result, many are unaware of tools and practices that would allow them to write more reliable and maintainable code with less effort. We describe a set 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

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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.

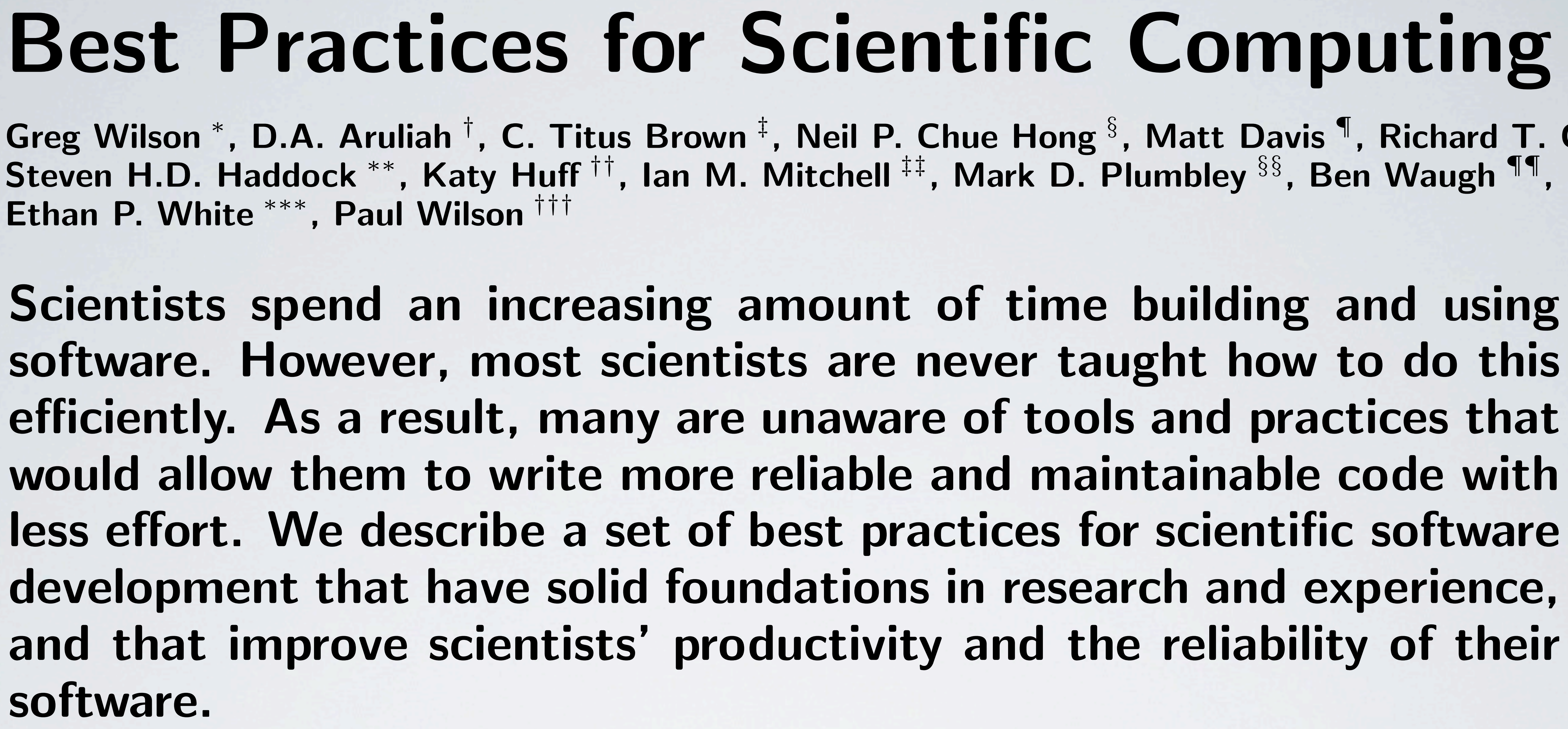
1. Write programs for people, not computers.
2. Automate repetitive tasks.
3. Use the computer to record history.
4. Make incremental changes.
5. Use version control.

Best Practices for Scientific Computing

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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.

arXiv:1210.0530v3 [cs.MS] 29 Nov 2012

Best Practices for Scientific Computing

Greg Wilson [*], D.A. Aruliah [†], C. Titus Brown [‡], Neil P. Chue Hong [§], Matt Davis [¶], Richard T. Guy [‖], Steven H.D. Haddock [**], Katy Huff [††], Ian M. Mitchell [‡‡], Mark D. Plumbley [§§], Ben Waugh [¶¶], Ethan P. White [***], Paul Wilson [†††]

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.