

Review of Piccolo and Frampton manuscript

Stephen Eglen

Summary

This review provides an overview of the main tools and techniques used to ensure computational reproducibility of results. This review is a good fit to the readership of the *Gigascience* journal. Overall I found the manuscript to be clearly written and would recommend it for publication, although I would like the authors to consider the following issues in a potential revision. In particular, I am most critical about the current set of figures.

As a general comment, where possible, the material in the figures should be supported by online files (e.g. knitr, Docker) so that the reader can immediately get *workable* examples to examine, run and amend.

Detailed comments

p4, l2 (page 4, line 2): an extra citation might be (Gronenschild et al. 2012) which showed that even the choice of operating system or neuroimaging software version affects results.

p4, l18-20: as far as I am aware, journals do not encourage direct use of repositories like *github* or *bitbucket* because they offer no long-term storage. It is regarded much more appropriate to use permanent URLs (e.g. DOIs) to point to archived versions of software, such as zenodo. We have written elsewhere on this topic (Eglen et al. 2016).

p6, l10: in addition to the other topics of reproducibility and education, it would be worth mentioning education/training to encourage users to adopt reproducible practices (Wilson 2016).

p6, l15: minor point, but perhaps worth numbering the seven sections that describe the seven approaches reviewed, starting here.

p8, l17: "Make can be configured"; if you are referring to the "-j N" switch, then it is simpler to say that "Make can automatically identify..." as there is no extra configuration needed.

p11 l9: reference 53 at the end of the sentence is not needed, as you refer to it at the start of the sentence.

p13, l3-9: you should probably mention <http://mybinder.org> in this section. It provides a transparent method for interacting with Jupyter documents over the web (Rosenberg and Horn 2016).

p14, l2: I disagree slightly here with the view re: long-running jobs. *knitr* at least can cache intermediate computations transparently which helps enormously.

p14, l8: Do you have examples of Dexy in use within this field? Asking this also in a more general way, it might be worth making a table listing examples explicitly of each of the seven approaches, so that they are easy to find.

p18, l18: I do not see why the VMs are "black boxes". Surely to create the VM all the relevant code must be provided, so that you can at least examine what is done, or extend the analysis (as mentioned on l20). Can you clarify what you mean here.

p22, 11: as well as capturing software in Docker, we have used it recently to capture the entire environment to write our research papers in knitr (<https://hub.docker.com/r/sje30/eglen2015/> and <https://hub.docker.com/r/sje30/waverepo/>).

p22, 114-19: I found the section about other operating systems (Windows/mac) rather clumsy. From the user's perspective, the modern docker toolbox seems to work smoothly enough (at least on macs) that the details at the end of p22 seem irrelevant. It might instead be worth mentioning that docker builds can be automatically triggered, e.g. upon new commits to github.

p 23, 16: "Scientific advancement requires *trust*." I think trust is the wrong word here. e.g. *The Royal Society's motto 'Nullius in verba' is taken to mean 'take nobody's word for it'* (<https://royalsociety.org/about-us/history/>). Rather, what these tools do is promote transparency to reduce the barriers for others to repeat prior work. With this in mind, I'd suggest the authors re-read this first paragraph of the discussion to see whether they think "trust" is what they are promoting here.

Figures

Figure 1: is this really needed in such a review? I don't think it adds anything.

Figure 2: I think should be provided as a text file so that people can run it for themselves.

Figure 3: Text file definitely needed so people can run it for themselves. But also I'd consider making the targets a bit more specific. All of the targets in this example Makefile are PHONY and perhaps it could be rewritten in the more canonical Makefile style? The way it is currently written, it is hard to see how this differs from a shell script. (Each time `make a11` is run, won't the files be downloaded again?)

Figure 4: can you show something that is a bit more bio-relevant, e.g. some genomic analysis?

Figure 5: as figure 4. Can you design Figure 4 and 5 carefully to highlight the differences between knitr and jupyter?

Figure 6-8: I would suggest you drop these. They show the notion of stacks and containers, but most bioscientists won't get much value from them.

References

Eglen, Stephen, Ben Marwick, Yaroslav Halchenko, Michael Hanke, Shoaib Sufi, Pdraig Gleeson, R Angus Silver, et al. 2016. "Towards Standard Practices for Sharing Computer Code and Programs in Neuroscience." *BioRxiv*. doi:[10.1101/045104](https://doi.org/10.1101/045104).

Gronenschild, Ed H B M, Petra Habets, Heidi I L Jacobs, Ron Mengelers, Nico Rozendaal, Jim van Os, and Machteld Marcelis. 2012. "The Effects of FreeSurfer Version, Workstation Type, and Macintosh Operating System Version on Anatomical Volume and Cortical Thickness Measurements." *PLoS One* 7 (6): e38234. doi:[10.1371/journal.pone.0038234](https://doi.org/10.1371/journal.pone.0038234).

Rosenberg, David M, and Charles C Horn. 2016. "Neurophysiological Analytics for All! Free Open-Source Software Tools for Documenting, Analyzing, Visualizing, and Sharing Using Electronic Notebooks." *J. Neurophysiol.*, 20~apr, jn.00137.2016. doi:[10.1152/jn.00137.2016](https://doi.org/10.1152/jn.00137.2016).

Wilson, Greg. 2016. "Software Carpentry: Lessons Learned." *F1000Res*. 3 (28~jan). doi:[10.12688/f1000research.3-62.v2](https://doi.org/10.12688/f1000research.3-62.v2).

GigaScience

Tools and techniques for computational reproducibility

--Manuscript Draft--

Manuscript Number:	
Full Title:	Tools and techniques for computational reproducibility
Article Type:	Review
Funding Information:	
Abstract:	<p>When reporting research findings, scientists document the steps they followed so that others can verify and build upon the research. When those steps have been described in sufficient detail that others can retrace the steps and obtain similar results, the research is said to be reproducible. Computers play a vital role in many research disciplines and present both opportunities and challenges for reproducibility. Computers can be programmed to execute analysis tasks, and those programs can be repeated and shared with others. Due to the deterministic nature of most computer programs, the same analysis tasks, applied to the same data, will often produce the same outputs. However, in practice, computational findings often cannot be reproduced due to complexities in how software is packaged, installed, and executed—and due to limitations in how scientists document analysis steps. Many tools and techniques are available to help overcome these challenges. Here we describe seven such strategies. With a broad scientific audience in mind, we describe strengths and limitations of each approach, as well as circumstances under which each might be applied. No single strategy is sufficient for every scenario; thus we emphasize that it is often useful to combine approaches.</p>
Corresponding Author:	Stephen R Piccolo, Ph.D. Brigham Young University Provo, UT UNITED STATES
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Brigham Young University
Corresponding Author's Secondary Institution:	
First Author:	Stephen R Piccolo, Ph.D.
First Author Secondary Information:	
Order of Authors:	Stephen R Piccolo, Ph.D. Michael B Frampton
Order of Authors Secondary Information:	
Opposed Reviewers:	
Additional Information:	
Question	Response
Experimental design and statistics	No
<p>Full details of the experimental design and statistical methods used should be given in the Methods section, as detailed in our Minimum Standards Reporting Checklist. Information essential to interpreting the data presented should be made available in the figure legends.</p>	

<p>Have you included all the information requested in your manuscript?</p>	
<p>If not, please give reasons for any omissions below.</p> <p>as follow-up to "Experimental design and statistics</p> <p>Full details of the experimental design and statistical methods used should be given in the Methods section, as detailed in our Minimum Standards Reporting Checklist. Information essential to interpreting the data presented should be made available in the figure legends.</p> <p>Have you included all the information requested in your manuscript?</p> <p>"</p>	<p>It is a review article.</p>
<p>Resources</p> <p>A description of all resources used, including antibodies, cell lines, animals and software tools, with enough information to allow them to be uniquely identified, should be included in the Methods section. Authors are strongly encouraged to cite Research Resource Identifiers (RRIDs) for antibodies, model organisms and tools, where possible.</p> <p>Have you included the information requested as detailed in our Minimum Standards Reporting Checklist?</p>	<p>No</p>
<p>If not, please give reasons for any omissions below.</p> <p>as follow-up to "Resources</p> <p>A description of all resources used, including antibodies, cell lines, animals and software tools, with enough information to allow them to be uniquely identified, should be included in the Methods section. Authors are strongly encouraged to cite Research Resource Identifiers (RRIDs) for antibodies, model organisms and tools, where possible.</p>	<p>It is a review article.</p>

<p>Have you included the information requested as detailed in our Minimum Standards Reporting Checklist?</p> <p>"</p>	
<p>Availability of data and materials</p> <p>All datasets and code on which the conclusions of the paper rely must be either included in your submission or deposited in publicly available repositories (where available and ethically appropriate), referencing such data using a unique identifier in the references and in the "Availability of Data and Materials" section of your manuscript.</p> <p>Have you have met the above requirement as detailed in our Minimum Standards Reporting Checklist?</p>	<p>No</p>
<p>If not, please give reasons for any omissions below.</p> <p>as follow-up to "Availability of data and materials</p> <p>All datasets and code on which the conclusions of the paper rely must be either included in your submission or deposited in publicly available repositories (where available and ethically appropriate), referencing such data using a unique identifier in the references and in the "Availability of Data and Materials" section of your manuscript.</p> <p>Have you have met the above requirement as detailed in our Minimum Standards Reporting Checklist?</p> <p>"</p>	<p>It is a review article.</p>

[Click here to view linked References](#)

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 **Tools and techniques for computational reproducibility**

2 Stephen R. Piccolo^{1*}, Michael B. Frampton²

3

4 1 - Department of Biology, Brigham Young University, Provo, UT, USA

5 2 - Department of Computer Science, Brigham Young University, Provo, UT,

6 USA

7

8 * Corresponding author: 4102 LSB, Brigham Young University, Provo, UT,

9 84602; 801-422-7116; stephen_piccolo@byu.edu

10

11 **Keywords**

12 Computational reproducibility; practice of science; literate programming; virtualization;

13 software containers; software frameworks.

14 **Competing Interests**

15 None of the authors of this manuscript have any competing interests to declare.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1

2 **Abstract**

3 When reporting research findings, scientists document the steps they followed so that
4 others can verify and build upon the research. When those steps have been described in
5 sufficient detail that others can retrace the steps and obtain similar results, the research is
6 said to be reproducible. Computers play a vital role in many research disciplines and
7 present both opportunities and challenges for reproducibility. Computers can be
8 programmed to execute analysis tasks, and those programs can be repeated and shared
9 with others. Due to the deterministic nature of most computer programs, the same analysis
10 tasks, applied to the same data, will often produce the same outputs. However, in practice,
11 computational findings often cannot be reproduced due to complexities in how software is
12 packaged, installed, and executed—and due to limitations in how scientists document
13 analysis steps. Many tools and techniques are available to help overcome these challenges.
14 Here we describe seven such strategies. With a broad scientific audience in mind, we
15 describe strengths and limitations of each approach, as well as circumstances under which
16 each might be applied. No single strategy is sufficient for every scenario; thus we
17 emphasize that it is often useful to combine approaches.

18

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 Introduction

2 When reporting research, scientists document the steps they followed to obtain their
3 results. If the description is comprehensive enough that they and others can repeat the
4 procedures and obtain semantically consistent results, the findings are considered to be
5 "reproducible"[1–6]. Reproducible research forms the basic building blocks of science,
6 insofar as it allows researchers to verify and build on each other's work with confidence.

7 Computers play an increasingly important role in many scientific disciplines[7–10]. For
8 example, in the United Kingdom, 92% of academic scientists use some type of software in
9 their research, and 69% of scientists say their research is feasible only with software
10 tools[11]. Thus efforts to increase scientific reproducibility should consider the ubiquity of
11 computers in research.

12 Computers present both opportunities and challenges for scientific reproducibility. On one
13 hand, due to the deterministic nature of most computer programs, many computational
14 analyses can be performed such that others can obtain exactly identical results when
15 applied to the same input data[12]. Accordingly, computational research can be held to a
16 high reproducibility standard. On the other hand, even when no technical barrier prevents
17 reproducibility, scientists often cannot reproduce computational findings due to
18 complexities in how software is packaged, installed, and executed—and due to limitations
19 in how scientists document these steps[13]. This problem is acute in many disciplines,
20 including genomics, signal processing, and ecological modeling[14–16], where data sets are
21 large and computational tools are evolving rapidly. However, the same problem can affect

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 any scientific discipline that requires computers for research. Seemingly minor differences
2 in computational approaches can have major influences on analytical outputs[12,17–21],
3 and the effects of these differences may exceed those that result from experimental
4 factors[22].

5 Journal editors, funding agencies, governmental institutions, and individual scientists have
6 increasingly made calls for the scientific community to embrace practices that support
7 computational reproducibility[23–30]. This movement has been motivated, in part, by
8 scientists' failed efforts to reproduce previously published analyses. For example,
9 Ioannidis, et al. evaluated 18 published research studies that used computational methods
10 to evaluate gene-expression data but were able to reproduce only 2 of those studies[31]. In
11 many cases, a failure to share the study's data was the culprit; however, incomplete
12 descriptions of software-based analyses were also common. Nekrutenko and Taylor
13 examined 50 papers that analyzed next-generation sequencing data and observed that
14 fewer than half provided any details about software versions or parameters[32].
15 Recreating analyses that lack such details can require hundreds of hours of effort[33] and
16 may be impossible, even after consulting the original authors. Failure to reproduce
17 research may also lead to careerist effects, including retractions[34].

18 Noting such concerns, some journals have emphasized the value of placing computer
19 source code in open-access repositories, such as *GitHub* (<https://github.com>) or *BitBucket*
20 (<https://bitbucket.org>). In addition, journals have extended requirements for "Methods"
21 sections, now asking researchers to provide detailed descriptions of 1) how to install

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 software and its dependencies and 2) what parameters and data-preprocessing steps are
2 used in analyses[10,23]. A recent Institute of Medicine report emphasized that, in addition
3 to computer code and research data, "fully specified computational procedures" should be
4 made available to the scientific community[24]. They elaborated that such procedures
5 should include "all of the steps of computational analysis" and that "all aspects of the
6 analysis need to be transparently reported"[24]. Such policies represent important
7 progress. However, it is ultimately the responsibility of individual scientists to ensure that
8 others can verify and build upon their analyses.

9 Describing a computational analysis sufficiently—such that others can reexecute it,
10 validate it, and refine it—requires more than simply stating what software was used, what
11 commands were executed, and where to find the source code[13,26,35–37]. Software is
12 executed within the context of an operating system (for example, *Windows*, *Mac OS*, or
13 *Linux*), which enables the software to interface with computer hardware (Figure 1). In
14 addition, most software relies on a hierarchy of software dependencies, which perform
15 complementary functions and must be installed alongside the main software tool. One
16 version of a given software tool or dependency may behave differently or have a different
17 interface than another version of the same software. In addition, most analytical software
18 offers a range of parameters (or settings) that the user can specify. If any of these variables
19 differs from what the original experimenter used, the software may not execute properly or
20 analytical outputs may differ considerably from what the original experimenter observed.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 Scientists can use various tools and techniques to overcome these challenges and to
2 increase the likelihood that their computational analyses will be reproducible. These
3 techniques range in complexity from simple (e.g., providing written documentation) to
4 advanced (e.g., providing a "virtual" environment that includes an operating system and all
5 software necessary to execute the analysis). This review describes seven strategies across
6 this spectrum. We describe strengths and limitations of each approach, as well as
7 circumstances under which each might be applied. No single strategy will be sufficient for
8 every scenario; therefore, in many cases, it will be most practical to combine multiple
9 approaches. This review focuses primarily on the computational aspects of reproducibility.
10 The related topics of empirical reproducibility, statistical reproducibility, and data sharing
11 have been described elsewhere[38–44]. We believe that with greater awareness and
12 understanding of computational-reproducibility techniques, scientists—including those
13 with limited computational experience—will be more apt to perform computational
14 research in a reproducible manner.

15 **Narrative descriptions are a simple but valuable way to support**
16 **computational reproducibility**

17 The most fundamental strategy for enabling others to reproduce a computational analysis
18 is to provide a detailed, written description of the process. For example, when reporting
19 computational results in a research article, authors customarily provide a narrative that
20 describes the software they used and the analytical steps they followed. Such narratives
21 can be invaluable in enabling others to evaluate the scientific approach and to reproduce

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 the findings. In many situations—for example, when software execution requires user
2 interaction or when proprietary software is used—narratives are the only feasible option
3 for documenting such steps. However, even when a computational analysis uses open-
4 source software and can be fully automated, narratives help others understand how to
5 reexecute an analysis.

6 Although most research articles that use computational methods provide some type of
7 narrative, these descriptions often lack sufficient detail to enable others to retrace those
8 steps [31,32]. Narrative descriptions should indicate the operating system(s), software
9 dependencies, and analytical software that were used and how to obtain them. In addition,
10 narratives should indicate the exact software versions used, the order in which they were
11 executed, and all non-default parameters that were specified. Such descriptions should
12 account for the fact that computer configurations differ vastly, even for computers that use
13 the same operating system. Because it can be difficult for scientists to remember such
14 details after the fact, it is best to record this information throughout the research process,
15 rather than at the time of manuscript preparation[8].

16 The following sections describe techniques for automating computational analyses. These
17 techniques can diminish the need for scientists to write narratives. However, because it is
18 often impractical to automate all computational steps, we expect that, for the foreseeable
19 future, narratives will play a vital role in enabling computational reproducibility.

1 Custom scripts and code can automate a research analysis

2 Scientific software can often be executed in an automated manner via text-based
3 commands. Using such commands—via a *command-line interface*—scientists can indicate
4 which software program(s) should be executed and which parameter(s) should be used.
5 When multiple commands must be executed, they can be compiled into *scripts*, which
6 specify the order in which the commands should be executed (Figure 2). In many cases,
7 scripts also include commands for installing and configuring software. Such scripts serve as
8 valuable documentation not only for individuals who wish to reexecute the analysis but
9 also for the researcher who performed the original analysis[45]. In these cases, no amount
10 of narrative is an adequate substitute for providing the actual commands that were used.
11 When writing command-line scripts, it is essential to explicitly document any software
12 dependencies and input data that are required for each step in the analysis. The *Make*
13 utility (<https://www.gnu.org/software/make>) provides one way to specify such
14 requirements[35]. Before any command is executed, *Make* verifies that each documented
15 dependency is available. Accordingly, researchers can use *Make* files (scripts) to specify a
16 full hierarchy of operating-system components and dependent software that must be
17 present to perform the analysis (Figure 3). In addition, *Make* can be configured to
18 automatically identify any commands that can be executed in parallel, potentially reducing
19 the amount of time required to execute the analysis. Although *Make* was designed
20 originally for UNIX-based operating systems (such as *Mac OS* or *Linux*), similar utilities
21 have since been developed for *Windows* operating systems

1 (http://gnuwin32.sourceforge.net/packages/make.htm). Box 1 lists various utilities that
2 can be used to automate software execution.

Box 1: Utilities that can be used to automate software execution.

- *GNU Make* and *Make for Windows*: Tools for building software from source files and for ensuring that the software's dependencies are met.
- *Snakemake*[46] = An extension of Make that provides a more flexible syntax and makes it easier to execute tasks in parallel.
- *BPipe*[47] = A tool that provides a flexible syntax for users to specify commands to be executed; it maintains an audit trail of all commands that have been executed.
- *GNU Parallel*[48] = A tool for executing commands in parallel across one or more computers.
- *Makeflow*[49] = A tool that can execute commands simultaneously on various types of computer architectures, including computer clusters and cloud environments.
- *SCONS*[50] = An alternative to *GNU Make* that enables users to customize the process of building and executing software using scripts written in the Python programming language.
- *CMAKE* (<https://cmake.org>) = A tool that enables users to execute *Make* scripts more easily on multiple operating systems.

3
4 In addition to creating scripts that execute existing software, many researchers also create
5 new software by writing computer code in a programming language such as Python, C++,
6 Java, or R. Such code may perform relatively simple tasks, such as reformatting data files or
7 invoking third-party software. In other cases, computer code may constitute a manuscript's
8 key intellectual contribution.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 Whether analysis steps are encoded in scripts or as computer code, scientists can support
2 reproducibility by publishing these artefacts alongside research papers. By doing so, the
3 authors enable readers to evaluate the analytical approach in full detail and to extend the
4 analysis more readily[51]. Although scripts and code may be included alongside a
5 manuscript as supplementary material, a better alternative is to store them in a *version-*
6 *control system* (VCS)[8,9,45] and to share these repositories via Web-based services like
7 *GitHub* (<https://github.com>) or *Bitbucket* (<https://bitbucket.org>). With such a VCS
8 repository, scientists can track different versions of scripts and code that have been
9 developed as the research project evolved. In addition, outside observers can see the full
10 version history, contribute revisions to the code, and reuse the code for their own
11 purposes[52]. When submitting a manuscript, the authors may “tag” a specific version of
12 the repository that was used for the final analysis described in the manuscript.

13 **Software frameworks enable easier handling of software dependencies**

14 Virtually all computer scripts and code rely on external software dependencies and
15 operating-system components. For example, suppose that a research study required a
16 scientist to apply *Student’s t-test*. Rather than write code that implements this statistical
17 test, the scientist would likely find an existing software *library* that implements the test and
18 then invoke that library from her code. A considerable amount of time can be saved with
19 this approach, and a wide range of software libraries are freely available. However,
20 software libraries change frequently—invoking the wrong version of a library may result in

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 an error or an unexpected output. Thus to enable others to reproduce an analysis, it is
2 critical to indicate which dependencies (and versions thereof) must be installed.

3 One way to address this challenge is to build on a preexisting software framework. Such
4 frameworks make it easier to access software libraries that are commonly used to perform
5 specific types of analysis task. Typically, such frameworks also make it easier to download
6 and install software dependencies and ensure that the versions of software libraries and
7 their dependencies are compatible with each other. For example, *Bioconductor*[53], created
8 for the *R* statistical programming language[54], is a popular framework that contains
9 hundreds of software packages for analyzing biological data[53]. The *Bioconductor*
10 framework facilitates versioning, documenting, and distributing code. Once a software
11 library has been incorporated into *Bioconductor*, other researchers can find, download,
12 install, and configure it on most operating systems with relative ease. In addition,
13 *Bioconductor* installs software dependencies automatically. These features ease the process
14 of performing an analysis and can help with reproducibility. Various software frameworks
15 exist for other scientific disciplines[55–60]. General-purpose tools for managing software
16 dependencies also exist—for example, *Apache Ivy* (<http://ant.apache.org/ivy>) and *Puppet*
17 (<https://puppetlabs.com>).

18 To best support reproducibility, software frameworks should make it easy for scientists to
19 download and install previous versions of a software tool as well as previous versions of
20 dependencies. Such a design would enable other scientists to reproduce analyses that were
21 conducted with previous versions of a software framework. In the case of *Bioconductor*,

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 considerable extra work may be required to install specific versions of Bioconductor
2 software and their dependencies. To overcome such limitations, scientists may use a
3 software container or virtual machine (see below) to package the specific versions they
4 used in an analysis. Alternatively, they might use third-party solutions such as the aRchive
5 project (<http://bioarchive.github.io>).

6 **Literate programming combines narratives directly with code**

7 Although narratives, scripts, and computer code support reproducibility individually,
8 additional value can be gained from combining these entities. Even though a researcher
9 may provide computer code alongside a research paper, other scientists may have difficulty
10 interpreting how the code accomplishes specific tasks. A longstanding way to address this
11 problem is via code comments, which are human-readable annotations interspersed
12 throughout computer code. However, code comments and other types of documentation
13 often become outdated as code evolves throughout the analysis process[61]. One way to
14 overcome this problem is to use a technique called *literate programming*[62]. With this
15 approach, the scientist writes a narrative of the scientific analysis and intermingles code
16 directly within the narrative. As the code is executed, a document is generated that includes
17 the code, narratives, and any outputs (e.g., figures, tables) that the code produces.

18 Accordingly, literate programming helps ensure that readers understand exactly how a
19 particular research result was obtained. In addition, this approach motivates the scientist
20 to keep the target audience in mind when performing a computational analysis, rather than
21 simply to write code that a computer can parse[62]. Consequently, by reducing barriers of

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 understanding among scientists, literate programming can help to engender greater trust
2 in computational findings.

3 One popular literate-programming tool is *Jupyter*[63]. Using its Web-based interface,
4 scientists can create interactive "notebooks" that combine code, data, mathematical
5 equations, plots, and rich media[64]. Originally known as *IPython* and previously designed
6 exclusively for the *Python* programming language, *Jupyter* (<http://jupyter.org>) now makes
7 it possible to execute code in many different programming languages. Such functionality
8 may be important to scientists who prefer to combine the strengths of different
9 programming languages.

10 *knitr*[65] has also gained considerable popularity as a literate-programming tool. It is
11 written in the *R* programming language and thus can be integrated seamlessly with the
12 array of statistical and plotting tools available in that environment. However, like *Jupyter*,
13 *knitr* can execute code written in multiple programming languages. Commonly, *knitr* is
14 applied to documents that have been authored using *RStudio*[66], an open-source tool with
15 advanced editing and package-management features.

16 *Jupyter* notebooks and *knitr* reports can be saved in various output formats, including
17 HTML and PDF (see examples in Figures 4-5). Increasingly, scientists include such
18 documents with journal manuscripts as supplementary material, enabling others to repeat
19 analysis steps and recreate manuscript figures[67–70].

20 Scientists typically use literate-programming tools for data analysis tasks that can be
21 executed in a modest amount of time (e.g., minutes or hours). It is possible to execute

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 *Jupyter* or *knitr* at the command line; thus longer-running tasks can be executed on high-
2 performance computers. However, this approach runs counter to the interactive nature of
3 notebooks and require additional technical expertise to configure and execute the
4 notebooks.

5 Literate-programming notebooks are suitable for research analyses that require a modest
6 amount of computer code. For analyses that require larger amounts of code, more
7 advanced programming environments may be more suitable—perhaps in combination
8 with a “literate documentation” tool such as Dexy (<http://www.dexy.it>).

9 **Workflow-management systems enable software execution via a**
10 **graphical user interface**

11 Writing computer scripts and code may seem daunting to many researchers. Although
12 various courses and tutorials are helping to make this task less formidable[71–74], many
13 scientists use "workflow management systems" to facilitate the process of executing
14 scientific software[75]. Typically managed via a graphical user interface, workflow
15 management systems enable scientists to upload data and process it using existing tools.
16 For multistep analyses, the output from one tool can be used as input to additional tools,
17 potentially resulting in a series of commands known as a workflow.

18 *Galaxy*[76,77] has gained considerable popularity within the bioinformatics community—
19 especially for performing next-generation sequencing analysis. As users construct
20 workflows, *Galaxy* provides descriptions of how software parameters should be used,

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 examples of how input files should be formatted, and links to relevant discussion forums.

2 To help with processing large data sets and computationally complex algorithms, *Galaxy*

3 also provides an option to execute workflows on cloud-computing services[78]. In addition,

4 researchers can share workflows with each other (<https://usegalaxy.org>); this feature has

5 enabled the *Galaxy* team to build a community that helps to encourage reproducibility,

6 define best practices, and reduce the time required for novices to get started.

7 Various other workflow systems are freely available to the research community (see Box

8 2). For example, *VisTrails* is used by researchers from many disciplines, including climate

9 science, microbial ecology, and quantum mechanics[79]. It enables scientists to design

10 workflows visually, connecting data inputs with analytical modules and the resulting

11 outputs. In addition, *VisTrails* tracks a full history of how each workflow was created. This

12 capability, referred to as "retrospective provenance", makes it possible for others not only

13 to reproduce the final version of an analysis but also to examine previous incarnations of

14 the workflow and examine how each change influenced the analytical outputs[80].

Box 2: Workflow management tools freely available to the research community.

- Galaxy[76,77] - <https://usegalaxy.org>
- VisTrails[79] - <http://www.vistrails.org>
- Kepler[81] - <https://kepler-project.org>
- iPlant Collaborative[82] - <http://www.iplantcollaborative.org>
- GenePattern[83,84] - <http://www.broadinstitute.org/cancer/software/genepattern>
- Taverna[85] - <http://www.taverna.org.uk>
- LONI Pipeline[86] - <http://pipeline.bmap.ucla.edu>

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 Although workflow-management systems offer many advantages, users must accept
2 tradeoffs. For example, although the teams that develop these tools often provide public
3 servers where users can execute workflows, many scientists share these limited resources,
4 so the public servers may not have adequate computational power or storage space to
5 execute large-scale analyses in a timely manner. As an alternative, many scientists install
6 these systems on their own computers; however, configuring and supporting them
7 requires time and expertise. In addition, if a workflow tool does not yet provide a module
8 to support a given analysis, the scientist must create a new module to support it. This task
9 constitutes additional overhead; however, utilities such as the *Galaxy Tool Shed*[87] are
10 helping to facilitate this process.

11 **Virtual machines encapsulate an entire operating system and software**
12 **dependencies**

13 Whether an analysis is executed at the command line, within a literate-programming
14 notebook, or via a workflow-management system, an operating system and relevant
15 software dependencies must be installed before the analysis can be performed. The process
16 of identifying, installing, and configuring such dependencies consumes a considerable
17 amount of scientists' time. Different operating systems (and versions thereof) may require
18 different installation and configuration steps. Furthermore, earlier versions of software
19 dependencies, which may currently be installed on a given computer, may be incompatible
20 with—or produce different outputs than—newer versions.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 One solution is to use virtual machines, which can encapsulate an entire operating system
2 and all software, scripts, code, and data necessary to execute a computational
3 analysis[88,89] (Figure 6). Using virtualization software—such as *VirtualBox* or *VMWare*
4 (see Box 3)—a virtual machine can be executed on practically any desktop, laptop, or
5 server, irrespective of the main ("host") operating system on the computer. For example,
6 even though a scientist's computer may be running a *Windows* operating system, the
7 scientist may perform an analysis on a *Linux* operating system that is running
8 concurrently—within a virtual machine—on the same computer. The scientist has full
9 control over the virtual ("guest") operating system and thus can install software and
10 modify configuration settings as necessary. In addition, a virtual machine can be
11 constrained to use specific amounts of computational resources (e.g., computer memory,
12 processing power), thus enabling system administrators to ensure that multiple virtual
13 machines can be executed simultaneously on the same computer without impacting each
14 other's performance. After executing an analysis, the scientist can export the entire virtual
15 machine to a single, binary file. Other scientists can then use this file to reconstitute the
16 same computational environment that was used for the original analysis. With a few
17 exceptions (see Discussion), these scientists will obtain exactly the same results that the
18 original scientist obtained. This process provides the added benefits that 1) the scientist
19 must only document the installation and configuration steps for a single operating system,
20 2) other scientists need only install the virtualization software and not individual software
21 components, and 3) analyses can be reexecuted indefinitely, so long as the virtualization
22 software remains compatible with current computer systems[90]. Also useful, a team of

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 scientists can employ virtual machines to ensure that each team member has the same
2 computational environment, even though the team members may have different
3 configurations on their host operating systems.

4 One criticism of using virtual machines to support computational reproducibility is that
5 virtual-machine files are large (typically multiple gigabytes), especially if they include raw
6 data files. This imposes a barrier for researchers to share virtual machines with the
7 research community. One option is to use cloud-computing services (see Box 4). Scientists
8 can execute an analysis in the cloud, take a "snapshot" of their virtual machine, and share it
9 with others in that environment[88,91]. Cloud-based services typically provide
10 repositories where virtual-machine files can be stored and shared easily among users.
11 Despite these advantages, some researchers may prefer that their data reside on local
12 computers, rather than in the cloud—at least while the research is being performed. In
13 addition, cloud-based services may use proprietary software, so virtual machines may only
14 be executable within each provider's infrastructure. Furthermore, to use a cloud-service
15 provider, scientists may need to activate a fee-based account.

16 Another criticism of using virtual machines to support computational reproducibility is
17 that the software and scripts used in the analysis will be less easily accessible to other
18 scientists—details of the analysis are effectively concealed behind a “black box”[92].

19 Although other researchers may be able to reexecute the analysis within the virtual
20 machine, it may be more difficult for them to understand and extend the analysis[92]. This
21 problem can be ameliorated when all narratives, scripts, and code are stored in public

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 repositories—separately from the virtual machine—and then imported when the analysis
2 is executed[93]. Another solution is to use a prepackaged virtual machine, such as *Cloud*
3 *BioLinux*, that contains a variety of software tools commonly used within a given research
4 community[94].

5 Scientists can automate the process of building and configuring virtual machines using
6 tools such as *Vagrant* or *Vortex* (see Box 3). For either tool, users can write text-based
7 configuration files that provide instructions for building virtual machines and allocating
8 computational resources to them. In addition, these configuration files can be used to
9 specify analysis steps[93]. Because these files are text based and relatively small (usually a
10 few kilobytes), scientists can share them easily and track different versions of the files via
11 source-control repositories. This approach also mitigates problems that might arise during
12 the analysis stage. For example, even when a computer's *host* operating system must be
13 reinstalled due to a computer hardware failure, the virtual machine can be recreated with
14 relative ease.

Box 3: Virtual-machine software.

Virtualization hypervisors:

- VirtualBox (open source) - <https://www.virtualbox.org>
- Xen (open source) - <http://www.xenproject.org>
- VMWare (partially open source) - <http://www.vmware.com>

Virtual-machine management tools:

- Vagrant (open source) - <https://www.vagrantup.com>
- Vortex (open source) - <https://github.com/websecurify/node-vortex>

1
2
3
4
5
6 **Box 4: Commercial cloud-service providers.**
7

- 8 ● Amazon Web Services - <http://aws.amazon.com>
 - 9 ● Rackspace Cloud - <http://www.rackspace.com/cloud>
 - 10 ● Google Cloud Platform - <https://cloud.google.com/compute>
 - 11 ● Windows Azure - <https://azure.microsoft.com>
- 12
13
14

15 1
16
17
18
19 2 **Software containers ease the process of installing and configuring**
20
21 **dependencies**
22
23
24

25
26 4 Software containers are a lighter-weight alternative to virtual machines. Like virtual
27
28 5 machines, containers can encapsulate operating-system components, scripts, code, and
29
30 6 data into a single package that can be shared with others. Thus, as with virtual machines,
31
32 7 analyses executed within a software container should produce identical outputs,
33
34 8 irrespective of the underlying operating system or whatever software may be installed
35
36 9 outside the container (see Discussion for caveats). As is true for virtual machines, multiple
37
38 10 containers can be executed simultaneously on a single computer, and each container may
39
40 11 contain different software versions and configurations. However, whereas virtual machines
41
42 12 include an entire operating system, software containers interface directly with the
43
44 13 computer's main operating system and extend it as needed (Figure 3). This design provides
45
46 14 less flexibility than virtual machines because containers are specific to a given type of
47
48 15 operating system; however, containers require considerably less computational overhead
49
50
51
52
53
54
55
56 16 than virtual machines and can be initialized much more quickly[95].
57
58

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 The open-source *Docker* utility (<https://www.docker.com>)—which has gained popularity
2 among informaticians since its release in 2013—provides the ability to build, execute, and
3 share software containers for Linux-based operating systems. Users specify a *Docker*
4 container's contents using text-based commands. These instructions can be placed in a
5 "Dockerfile," which other scientists can use to rebuild the container. As with virtual-
6 machine configuration files, Dockerfiles are text based, so they can be shared easily and can
7 be tracked and versioned in source-control repositories. Once a *Docker* container has been
8 built, its contents can be exported to a binary file; these files are generally smaller than
9 virtual-machine files, so they can be shared more easily—for example, via *DockerHub*
10 (<https://hub.docker.com>).

11 A key feature of *Docker* containers is that their contents can be stacked in distinct layers (or
12 "images"). Each image includes software component(s) that address a particular need (see
13 Figure 7 for an example). Within a given research lab, scientists might create general-
14 purpose images that support functionality for multiple projects, and they might create
15 specialized images that address the needs of specific projects. *Docker's* modular design
16 provides the advantage that when images within a container are updated, *Docker* only
17 needs to track the specific components that have changed; users who wish to update to a
18 newer version must download a relatively small update. In contrast, even a minor change
19 to a virtual machine would require users to export and reshare the entire virtual machine.
20 Scientists have begun to share *Docker* images with others who are working in the same
21 subsdiscipline. For example, *nucleotid.es* is a catalog of genome-assembly tools that have

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 been encapsulated in *Docker* images[96,97]. Genome-assembly tools differ considerably in
2 the dependencies that they require and in the parameters that they support. This project
3 provides a means to standardize these assemblers, to circumvent the need to install
4 dependencies for each tool, and to perform benchmarks across the tools. Such projects may
5 help to reduce the reproducibility burden on individual scientists.

6 The use of *Docker* containers for reproducible research comes with caveats. Individual
7 containers are stored and executed in isolation from other containers on the same
8 computer; however, because all containers on a given machine share the same operating
9 system, this isolation is not as complete as it is with virtual machines. This means, for
10 example, that a given container is not guaranteed to have access to a specific amount of
11 computer memory or processing power—multiple containers may have to compete for
12 these resources[95]. In addition, containers may be more vulnerable to security
13 breaches[95]. Another caveat is that *Docker* containers can only be executed on Linux-
14 based operating systems. For other operating systems, *Docker* containers must be executed
15 within a virtual machine (for example, see <http://boot2docker.io>). Although this
16 configuration offsets some benefits of using containers, combining virtual machines with
17 containers may provide a happy medium for many scientists, allowing them to use a non-
18 Linux *host* operating system, while receiving the benefits of containers within the *guest*
19 operating system.

1
2
3
4 1 Efforts are ongoing to develop and refine software-container technologies. Box 5 lists
5
6 2 various tools that are currently available. In coming years, these technologies promise to
7
8
9 3 play an influential role within the scientific community.
10
11

12
13
14 **Box 5: Open-source containerization software.**

- 15 ● Docker - <https://www.docker.com>
 - 16 ● Linux Containers - <https://linuxcontainers.org>
 - 17 ● Lmctfy - <https://github.com/google/lmctfy>
 - 18 ● OpenVZ - <http://openvz.org>
 - 19 ● Warden - <http://docs.cloudfoundry.org/concepts/architecture/warden.html>
- 20
21
22
23
24

25 4

26
27
28
29 **5 Discussion**

30
31
32
33 6 Scientific advancement requires trust. This review provides a comprehensive, though
34
35 7 inexhaustive, list of techniques that can help to engender such trust. Principally, scientists
36
37 8 must perform research in such ways that they can trust their own findings[3,45]. Science
38
39 9 philosopher Karl Popper contended that "[w]e do not take even our own observations
40
41 10 quite seriously, or accept them as scientific observations, until we have repeated and tested
42
43 11 them"[2]. Indeed, in many cases, the individuals who benefit most from computational
44
45 12 reproducibility are those who performed the original analysis. But reproducible practices
46
47 13 can also help scientists garner each other's trust[45,98]. When other scientists can
48
49 14 reproduce an analysis and determine exactly how its conclusions were drawn, they may be
50
51 15 more apt to cite the work and build upon it. In contrast, when others fail to reproduce
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 research findings, it can derail scientific progress and lead to embarrassment, accusations,
2 and retractions.

3 We have described seven tools and techniques for computational reproducibility. None of
4 these approaches is sufficient for every scenario in isolation. Rather scientists will often
5 find value in combining approaches. For example, a researcher who uses a literate-
6 programming notebook (which combines narratives with code) might incorporate the
7 notebook into a software container so that others can execute it without needing to install
8 specific software dependencies. The container might also include a workflow-management
9 system to ease the process of integrating multiple tools and incorporating best practices for
10 the analysis. This container could be packaged within a virtual machine to ensure that it
11 can be executed on many operating systems (see Figure 8). In determining a
12 reproducibility strategy, scientists must evaluate the tradeoff between robustness and
13 practicality.

14 The call for computational reproducibility relies on the premise that reproducible science
15 will bolster the efficiency of the overall scientific enterprise[99]. Although reproducible
16 practices may require additional time and effort, these practices provide ancillary benefits
17 that help offset those expenditures[45]. Primarily, the scientists who perform a study may
18 experience increased efficiency[45]. For example, before and after a manuscript is
19 submitted for publication, it faces scrutiny from co-authors and peer reviewers who may
20 suggest alterations to the analysis. Having a complete record of all analysis steps and being
21 able to retrace those steps precisely, makes it faster and easier to implement the requested

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 alterations[45,100]. Reproducible practices can also improve the efficiency of team science
2 because colleagues can more easily communicate their research protocols and inspect each
3 other's work; one type of relationship where this is critical is that between academic
4 advisors and mentees[100]. Finally, when research protocols are shared transparently with
5 the broader community, scientific advancement increases because scientists can learn
6 more easily from each other's work and duplicate each other's efforts less frequently[100].
7 Reproducible practices do not necessarily ensure that others can obtain results that are
8 perfectly identical to what the original scientists obtained. Indeed, this objective may be
9 infeasible for some types of computational analysis, including those that use randomization
10 procedures, floating-point operations, or specialized computer hardware[89,101]. In such
11 cases, the goal may shift to ensuring that others can obtain results that are semantically
12 consistent with the original findings[5,6]. In addition, in studies where vast computational
13 resources are needed to perform an analysis or where data sets are distributed
14 geographically[102–104], full reproducibility may be infeasible. Alternatively, it may be
15 infeasible to reallocate computational resources for analyses that are highly
16 computationally intensive[8]. In these cases, researchers can provide relatively simple
17 examples that demonstrate the methodology[8]. When legal restrictions prevent
18 researchers from sharing software or data publicly, or when software is available only via a
19 Web interface, researchers should document the analysis steps as well as possible and
20 describe why such components cannot be shared[24].

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 Computational reproducibility does not guarantee against analytical biases or ensure that
2 software produces scientifically valid results[105]. As with any research, a poor study
3 design, confounding effects, or improper use of analytical software may plague even the
4 most reproducible analyses[105,106]. On one hand, increased transparency puts scientists
5 at a greater risk that such problems will be exposed. On the other hand, scientists who are
6 fully transparent about their scientific approach may be more likely to avoid such pitfalls,
7 knowing that they will be more vulnerable to such criticisms. Either way, the scientific
8 community benefits.

9 Lastly, we emphasize that some reproducibility is better than none. As Voltaire said, the
10 perfect should not be the enemy of the good[107]. Although some of the practices
11 described in this review require more technical expertise than others, these practices are
12 freely accessible to all scientists and provide long-term benefits to the researcher and to
13 the scientific community. Indeed, as scientists act in good faith to perform these practices,
14 where feasible, the pace of scientific progress will surely increase.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1

2 **Figure Legends**

3 **Figure 1: Basic computer architecture.** Computer hardware consists of hardware
4 devices, including central processing units, hard drives, random access memory, keyboard,
5 mouse, etc. Operating systems enable software to interface with hardware; popular
6 operating-system families are *Windows*, *Mac OS*, and *Linux*. Users interact with computers
7 via software interfaces. In scientific computing, software enables users to execute
8 algorithms, analyze data, generate graphics, etc. To execute properly, most software tools
9 depend on specific versions of software dependencies, which must be installed on the same
10 operating system.

11 **Figure 2: Example of a command-line script.** This script can be used to align DNA
12 sequence data to a reference genome. First it downloads software and data files necessary
13 for the analysis. Then it extracts (“unzips”) these files, aligns the data to a reference
14 genome for Ebolavirus. Finally, it converts, sorts, and indexes the aligned data.

15 **Figure 3: Example of a *Make* file.** This file performs the same function as the command-
16 line script shown in Figure 2, except that it is formatted for the *Make* utility. Accordingly, it
17 is structured so that specific tasks must be executed before other tasks, in a hierarchical
18 structure. For example, the “download” target must be completed before the downloaded
19 files can be prepared (“prepare” target). The “all” target specifies all of the targets that
20 must be executed, in order.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 **Figure 4: Example of a *Jupyter* notebook.** This example contains code (in the Python
2 programming language) for generating random numbers and plotting them in a graph
3 within a *Jupyter* notebook. Importantly, the code and output object (graph) are contained
4 within the same document.

5 **Figure 5: Example of a document that has been created using *knitr*.** This example
6 contains code (in the R language) for generating random numbers and plotting them in a
7 graph. The *knitr* tool was used to generate the document, which combines the code and the
8 output object (figure).

9 **Figure 6: Architecture of virtual machines.** Virtual machines encapsulate analytical
10 software and dependencies within a "guest" operating system, which may be different than
11 the main ("host") operating system. A virtual machine executes in the context of
12 virtualization software, which executes alongside whatever other software is installed on
13 the computer.

14 **Figure 7: Architecture of software containers.** Software containers encapsulate
15 analytical software and dependencies. In contrast to virtual machines, containers execute
16 within the context of the computer's main operating system.

17 **Figure 8: Example of a *Docker* container that could be used for genomics research.**
18 This container would enable researchers to preprocess various types of molecular data,
19 using tools from *Bioconductor* and *Galaxy*, and to analyze the resulting data within an
20 *IPython* notebook. Each box within the container represents a distinct *Docker* image. These
21 images are layered such that some images depend on others (for example, the *Bioconductor*

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 image depends on R). At its base, the container includes operating-system libraries, which
2 may not be present (or may be configured differently) on the computer's main operating
3 system.

4

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

References

1. Fisher RA. The Design of Experiments. New York: Hafner Press; 1935.
2. Popper KR. 2002. The logic of scientific discovery. London Routledge; 1959.
3. Peng RD. Reproducible research in computational science. *Science*. 2011;334:1226–7.
4. Russell JF. If a job is worth doing, it is worth doing twice. *Nature*. 2013;496:7.
5. Feynman RP, Leighton RB, Sands M. Six Easy Pieces: Essentials of Physics Explained by Its Most Brilliant Teacher. Perseus Books; 1994. p. 34–5.
6. Murray-Rust P, Murray-Rust D. Reproducible Physical Science and the Declaratron. In: Stodden VC, Leisch F, Peng RD, editors. Implementing Reproducible Research. CRC Press; 2014. p. 113.
7. Hey AJG, Tansley S, Tolle KM, Others. The fourth paradigm: data-intensive scientific discovery. Microsoft Research Redmond, WA; 2009.
8. Millman KJ, Pérez F. Developing Open-Source Scientific Practice. Implementing Reproducible Research. CRC Press; 2014;149.
9. Wilson G, Aruliah DA, Brown CT, Chue Hong NP, Davis M, Guy RT, et al. Best practices for scientific computing. *PLoS Biol*. 2014;12:e1001745.
10. Software with impact. *Nat. Methods*. 2014;11:211.
11. Hong NC. We are the 92% [Internet]. Figshare; 2014. Available from: <http://dx.doi.org/10.6084/M9.FIGSHARE.1243288>
12. Sacks J, Welch WJ, Mitchell TJ, Wynn HP. Design and Analysis of Computer Experiments. *Stat. Sci.* Institute of Mathematical Statistics; 1989;4:409–23.
13. Garijo D, Kinnings S, Xie L, Xie L, Zhang Y, Bourne PE, et al. Quantifying reproducibility in computational biology: the case of the tuberculosis drugome. *PLoS One*. 2013;8:e80278.
14. Error prone. *Nature*. 2012;487:406.
15. Vandewalle P, Barrenetxea G, Jovanovic I, Ridolfi A, Vetterli M. Experiences with Reproducible Research in Various Facets of Signal Processing Research. 2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07. IEEE; 2007. p. IV – 1253 – IV – 1256.
16. Cassey P, Cassey P, Blackburn T, Blackburn T. Reproducibility and Repeatability in Ecology. *Bioscience*. 2006;56:958–9.
17. Murphy JM, Sexton DMH, Barnett DN, Jones GS, Webb MJ, Collins M, et al. Quantification of modelling uncertainties in a large ensemble of climate change simulations. *Nature*. 2004;430:768–

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 72.

2 18. McCarthy DJ, Humburg P, Kanapin A, Rivas MA, Gaulton K, Cazier J-B, et al. Choice of transcripts
3 and software has a large effect on variant annotation. *Genome Med.* 2014;6:26.

4 19. Neuman JA, Isakov O, Shomron N. Analysis of insertion-deletion from deep-sequencing data:
5 Software evaluation for optimal detection. *Brief. Bioinform.* 2013;14:46–55.

6 20. Bradnam KR, Fass JN, Alexandrov A, Baranay P, Bechner M, Birol I, et al. Assemblathon 2:
7 evaluating de novo methods of genome assembly in three vertebrate species. *Gigascience.*
8 2013;2:10.

9 21. Bilal E, Dutkowski J, Guinney J, Jang IS, Logsdon BA, Pandey G, et al. Improving Breast Cancer
10 Survival Analysis through Competition-Based Multidimensional Modeling. *PLoS Comput. Biol.*
11 2013;9:e1003047.

12 22. Moskvina OV, McIlwain S, Ong IM. CAMDA 2014: Making sense of RNA-Seq data: From low-level
13 processing to functional analysis. *Systems Biomedicine.* 2014;2:31–40.

14 23. Reducing our irreproducibility. *Nature.* 2013;496:398–398.

15 24. Michael CM, Nass SJ, Omenn GS, editors. *Evolution of Translational Omics: Lessons Learned and
16 the Path Forward.* Washington, D.C.: The National Academies Press; 2012.

17 25. Collins FS, Tabak LA. Policy: NIH plans to enhance reproducibility. *Nature.* 2014;505:612–3.

18 26. Chambers JM. S as a Programming Environment for Data Analysis and Graphics. *Problem
19 Solving Environments for Scientific Computing, Proc. 17th Symp. on the Interface of Stat. and Comp.*
20 North Holland; 1985. p. 211–4.

21 27. LeVeque RJ, Mitchell IM, Stodden V. Reproducible research for scientific computing: Tools and
22 strategies for changing the culture. *Computing in Science and Engineering.* 2012;14:13.

23 28. Stodden V, Guo P, Ma Z. Toward Reproducible Computational Research: An Empirical Analysis
24 of Data and Code Policy Adoption by Journals. *PLoS One.* 2013;8:2–9.

25 29. Morin A, Urban J, Adams PD, Foster I, Sali A, Baker D, et al. Research priorities. Shining light into
26 black boxes. *Science.* 2012;336:159–60.

27 30. Rebooting review. *Nat. Biotechnol.* 2015;33:319.

28 31. Ioannidis JP a., Allison DB, Ball C a., Coulibaly I, Cui X, Culhane AC, et al. Repeatability of
29 published microarray gene expression analyses. *Nat. Genet.* 2009;41:149–55.

30 32. Nekrutenko A, Taylor J. Next-generation sequencing data interpretation: enhancing
31 reproducibility and accessibility. *Nat. Rev. Genet.* Nature Publishing Group; 2012;13:667–72.

32 33. Baggerly KA, Coombes KR. Deriving chemosensitivity from cell lines: Forensic bioinformatics
33 and reproducible research in high-throughput biology. *Ann. Appl. Stat.* 2009;3:1309–34.

1
2
3
4 1 34. Decullier E, Huot L, Samson G, Maisonneuve H. Visibility of retractions: a cross-sectional one-
5 2 year study. BMC Res. Notes. 2013;6:238.
6
7 3 35. Claerbout JF, Karrenbach M. Electronic Documents Give Reproducible Research a New Meaning.
8 4 Meeting of the Society of Exploration Geophysics. New Orleans, LA; 1992.
9
10 5 36. Stodden V, Miguez S. Best Practices for Computational Science: Software Infrastructure and
11 6 Environments for Reproducible and Extensible Research. Journal of Open Research Software.
12 7 2014;2:21.
13
14 8 37. Ravel J, Wommack KE. All hail reproducibility in microbiome research. Microbiome. 2014;2:8.
15
16 9 38. Stodden V. 2014: What scientific idea is ready for retirement? [Internet].
17 10 <http://edge.org/response-detail/25340>. 2014. Available from: <http://edge.org/response->
18 11 [detail/25340](http://edge.org/response-detail/25340)
19
20 12 39. Birney E, Hudson TJ, Green ED, Gunter C, Eddy S, Rogers J, et al. Prepublication data sharing.
21 13 Nature. 2009;461:168–70.
22
23 14 40. Hothorn T, Leisch F. Case studies in reproducibility. Brief. Bioinform. 2011;12:288–300.
24
25 15 41. Schofield PN, Bubela T, Weaver T, Portilla L, Brown SD, Hancock JM, et al. Post-publication
26 16 sharing of data and tools. Nature. 2009;461:171–3.
27
28 17 42. Piwowar H a, Day RS, Fridsma DB. Sharing detailed research data is associated with increased
29 18 citation rate. PLoS One [Internet]. 2007;2. Available from:
30 19 <http://dx.doi.org/10.1371/journal.pone.0000308>
31
32 20 43. Johnson VE. Revised standards for statistical evidence. Proc. Natl. Acad. Sci. U. S. A.
33 21 2013;110:19313–7.
34
35 22 44. Halsey LG, Curran-everett D, Vowler SL, Drummond GB. The fickle P value generates
36 23 irreproducible results. Nat. Methods. 2015;12:179–85.
37
38 24 45. Sandve GK, Nekrutenko A, Taylor J, Hovig E. Ten Simple Rules for Reproducible Computational
39 25 Research. PLoS Comput. Biol. 2013;9:1–4.
40
41 26 46. Köster J, Rahmann S. Snakemake--a scalable bioinformatics workflow engine. Bioinformatics.
42 27 2012;28:2520–2.
43
44 28 47. Sadedin SP, Pope B, Oshlack A. Bpipe : A Tool for Running and Managing Bioinformatics
45 29 Pipelines. Bioinformatics. 2012;28:1525–6.
46
47 30 48. Tange O. GNU Parallel - The Command-Line Power Tool. ;login: The USENIX Magazine.
48 31 Frederiksberg, Denmark; 2011;36:42–7.
49
50 32 49. Albrecht M, Donnelly P, Bui P, Thain D. Makeflow: A portable abstraction for data intensive
51 33 computing on clusters, clouds, and grids. Proceedings of the 1st ACM SIGMOD Workshop on
52 34 Scalable Workflow Execution Engines and Technologies. 2012.
53
54
55
56
57
58
59
60
61
62
63
64
65

1
2
3
4 1 50. Knight S, Austin C, Crain C, Leblanc S, Roach A. Scons software construction tool [Internet].
5 2 2011. Available from: <http://www.scons.org>
6
7 3 51. Code share. *Nature*. 2014;514:536.
8
9 4 52. Loeliger J, McCullough M. *Version Control with Git: Powerful Tools and Techniques for*
10 5 *Collaborative Software Development*. "O'Reilly Media, Inc."; 2012. p. 456.
11
12 6 53. Huber W, Carey VJ, Gentleman R, Anders S, Carlson M, Carvalho BS, et al. Orchestrating high-
13 7 throughput genomic analysis with Bioconductor. *Nat. Methods*. Nature Publishing Group;
14 8 2015;12:115–21.
15
16 9 54. R Core Team. *R: A Language and Environment for Statistical Computing* [Internet]. Vienna,
17 10 Austria: R Foundation for Statistical Computing; 2014. Available from: <http://www.r-project.org>
18
19 11 55. Tóth G, Sokolov IV, Gombosi TI, Chesney DR, Clauer CR, De Zeeuw DL, et al. Space Weather
20 12 Modeling Framework: A new tool for the space science community. *J. Geophys. Res.*
21 13 2005;110:A12226.
22
23 14 56. Tan E, Choi E, Thoutireddy P, Gurnis M, Aivazis M. *GeoFramework: Coupling multiple models of*
24 15 *mantle convection within a computational framework*. *Geochem. Geophys. Geosyst.* [Internet].
25 16 2006;7. Available from: <http://doi.wiley.com/10.1029/2005GC001155>
26
27 17 57. Heisen B, Boukhelef D, Esenov S, Hauf S, Kozlova I, Maia L, et al. *Karabo: An Integrated Software*
28 18 *Framework Combining Control, Data Management, and Scientific Computing Tasks*. 14th
29 19 *International Conference on Accelerator & Large Experimental Physics Control Systems*,
30 20 *ICALEPCS2013*. San Francisco, CA; 2013.
31
32 21 58. Schneider CA, Rasband WS, Eliceiri KW. *NIH Image to ImageJ: 25 years of image analysis*. *Nat.*
33 22 *Methods*. Nature Publishing Group; 2012;9:671–5.
34
35 23 59. Schindelin J, Arganda-Carreras I, Frise E, Kaynig V, Longair M, Pietzsch T, et al. *Fiji: an open-*
36 24 *source platform for biological-image analysis*. *Nat. Methods*. Nature Publishing Group, a division of
37 25 *Macmillan Publishers Limited*. All Rights Reserved.; 2012;9:676–82.
38
39 26 60. Biasini M, Schmidt T, Bienert S, Mariani V, Studer G, Haas J, et al. *OpenStructure: an integrated*
40 27 *software framework for computational structural biology*. *Acta Crystallogr. D Biol. Crystallogr.*
41 28 2013;69:701–9.
42
43 29 61. Martin RC. *Clean code: a handbook of agile software craftsmanship*. Pearson Education; 2009.
44
45 30 62. Knuth DE. *Literate Programming*. *Comput. J.* 1984;27:97–111.
46
47 31 63. Pérez F, Granger BE. *IPython: a System for Interactive Scientific Computing*. *Computing in*
48 32 *Science and Engineering*. IEEE Computer Society; 2007;9:21–9.
49
50 33 64. Shen H. *Interactive notebooks: Sharing the code*. *Nature*. 2014;515:151–2.
51
52 34 65. Xie Y. *Dynamic Documents with R and knitr*. CRC Press; 2013. p. 216.
53
54
55
56
57
58
59
60
61
62
63
64
65

1
2
3
4 1 66. RStudio Team. RStudio: Integrated Development for R [Internet]. [cited 2015 Nov 20]. Available
5 2 from: <http://www.rstudio.com>
6
7 3 67. Gross AM, Orosco RK, Shen JP, Egloff AM, Carter H, Hofree M, et al. Multi-tiered genomic analysis
8 4 of head and neck cancer ties TP53 mutation to 3p loss. *Nat. Genet.* Nature Publishing Group;
9 5 2014;46:1–7.
10
11 6 68. Ding T, Schloss PD. Dynamics and associations of microbial community types across the human
12 7 body. *Nature.* Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights
13 8 Reserved.; 2014;509:357–60.
14
15 9 69. Ram Y, Hadany L. The probability of improvement in Fisher’s geometric model: A probabilistic
16 10 approach. *Theor. Popul. Biol.* 2015;99:1–6.
17
18 11 70. Meadow JF, Altrichter AE, Kembel SW, Moriyama M, O’Connor TK, Womack AM, et al. Bacterial
19 12 communities on classroom surfaces vary with human contact. *Microbiome.* 2014;2:7.
20
21 13 71. White E. Programming for Biologists [Internet]. Available from:
22 14 <http://www.programmingforbiologists.org>
23
24 15 72. Wilson G. Software Carpentry: lessons learned. *F1000Res.* 2014;3:62.
25
26 16 73. Peng RD. Coursera course: Computing for Data Analysis [Internet]. Available from:
27 17 <https://www.coursera.org/course/compdata>
28
29 18 74. Bioconductor - Courses and Conferences [Internet]. [cited 2015 Nov 20]. Available from:
30 19 <http://master.bioconductor.org/help/course-materials>
31
32 20 75. Gil Y, Deelman E, Ellisman M, Fahringer T, Fox G, Gannon D, et al. Examining the challenges of
33 21 scientific workflows. *Computer .* 2007;40:24–32.
34
35 22 76. Giardine B, Riemer C, Hardison RC, Burhans R, Elnitski L, Shah P, et al. Galaxy: a platform for
36 23 interactive large-scale genome analysis. *Genome Res.* 2005;15:1451–5.
37
38 24 77. Goecks J, Nekrutenko A, Taylor J. Galaxy: a comprehensive approach for supporting accessible,
39 25 reproducible, and transparent computational research in the life sciences. *Genome Biol.*
40 26 2010;11:R86.
41
42 27 78. Afgan E, Baker D, Coraor N, Goto H, Paul IM, Makova KD, et al. Harnessing cloud computing with
43 28 Galaxy Cloud. *Nat. Biotechnol.* 2011;29:972–4.
44
45 29 79. Callahan SP, Freire J, Santos E, Scheidegger CE, Silva CT, Vo HT. VisTrails: Visualization Meets
46 30 Data Management. *Proceedings of the 2006 ACM SIGMOD International Conference on Management*
47 31 *of Data.* New York, NY, USA: ACM; 2006. p. 745–7.
48
49 32 80. Davidson SB, Freire J. Provenance and scientific workflows. *Proceedings of the 2008 ACM*
50 33 *SIGMOD international conference on Management of data - SIGMOD ’08.* 2008. p. 1345.
51
52 34 81. Altintas I, Berkley C, Jaeger E, Jones M, Ludascher B, Mock S. Kepler: an extensible system for
53 35 design and execution of scientific workflows. *Proceedings. 16th International Conference on*

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 Scientific and Statistical Database Management, 2004. IEEE; 2004. p. 423–4.

2 82. Goff SA, Vaughn M, McKay S, Lyons E, Stapleton AE, Gessler D, et al. The iPlant Collaborative:
3 Cyberinfrastructure for Plant Biology. *Front. Plant Sci. Frontiers*; 2011;2:34.

4 83. Reich M, Liefeld T, Gould J, Lerner J, Tamayo P, Mesirov JP. GenePattern 2.0. *Nat. Genet.*
5 2006;38:500–1.

6 84. Reich M, Liefeld J, Thorvaldsdottir H, Ocana M, Polk E, Jang D, et al. GenomeSpace: An
7 environment for frictionless bioinformatics. *Cancer Res.* 2012;72:3966–3966.

8 85. Wolstencroft K, Haines R, Fellows D, Williams A, Withers D, Owen S, et al. The Taverna workflow
9 suite: designing and executing workflows of Web Services on the desktop, web or in the cloud.
10 *Nucleic Acids Res.* 2013;41:557–61.

11 86. Rex DE, Ma JQ, Toga AW. The LONI Pipeline Processing Environment. *Neuroimage.*
12 2003;19:1033–48.

13 87. Lazarus R, Kaspi A, Ziemann M. Creating re-usable tools from scripts: The Galaxy Tool Factory.
14 *Bioinformatics.* 2012;28:3139–40.

15 88. Dudley JT, Butte AJ. In silico research in the era of cloud computing. *Nat. Biotechnol. Nature*
16 *Publishing Group*; 2010;28:1181–5.

17 89. Hurley DG, Budden DM, Crampin EJ. Virtual Reference Environments: a simple way to make
18 research reproducible. *Brief. Bioinform.* 2014;1–3.

19 90. Gent IP. The Recomputation Manifesto. arXiv [Internet]. 2013; Available from:
20 <http://arxiv.org/abs/1304.3674>

21 91. Howe B. Virtual Appliances, Cloud Computing, and Reproducible Research. *Comput. Sci. Eng.*
22 *IEEE Computer Society*; 2012;14:36–41.

23 92. Brown CT. Virtual machines considered harmful for reproducibility [Internet]. 2012. Available
24 from: <http://ivory.idyll.org/blog/vms-considered-harmful.html>

25 93. Piccolo SR. Building portable analytical environments to improve sustainability of
26 computational-analysis pipelines in the sciences [Internet]. 2014. Available from:
27 <http://dx.doi.org/10.6084/m9.figshare.1112571>

28 94. Krampis K, Booth T, Chapman B, Tiwari B, Bica M, Field D, et al. Cloud BioLinux: pre-configured
29 and on-demand bioinformatics computing for the genomics community. *BMC Bioinformatics.*
30 *BioMed Central Ltd*; 2012;13:42.

31 95. Felter W, Ferreira A, Rajamony R, Rubio J. An Updated Performance Comparison of Virtual
32 Machines and Linux Containers [Internet]. IBM Research Division; 2014. Available from:
33 [http://domino.research.ibm.com/library/CyberDig.nsf/papers/0929052195DD819C85257D2300](http://domino.research.ibm.com/library/CyberDig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf)
34 [681E7B/\\$File/rc25482.pdf](http://domino.research.ibm.com/library/CyberDig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf)

35 96. Belmann P, Dröge J, Bremges A, McHardy AC, Sczyrba A, Barton MD. Bioboxes: standardised

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1 containers for interchangeable bioinformatics software. *Gigascience*. 2015;4:47.

2 97. Barton M. nucleotides · genome assembler benchmarking [Internet]. [cited 2015 Nov 20].
3 Available from: <http://nucleotides>

4 98. Honess MJ. Reproducibility as a Methodological Imperative in Experimental Research. PSA:
5 Proceedings of the Biennial Meeting of the Philosophy of Science Association. Philosophy of Science
6 Association; 1990. p. 585–99.

7 99. Crick T. “ Share and Enjoy ”: Publishing Useful and Usable Scientific Models. Available from:
8 <http://arxiv.org/abs/1409.0367v2>

9 100. Donoho DL. An invitation to reproducible computational research. *Biostatistics*. 2010;11:385–
10 8.

11 101. Goldberg D. What Every Computer Scientist Should Know About Floating-point Arithmetic.
12 *ACM Comput. Surv.* New York, NY, USA: ACM; 1991;23:5–48.

13 102. Shirts M, Pande VS. COMPUTING: Screen Savers of the World Unite! *Science*. 2000;290:1903–4.

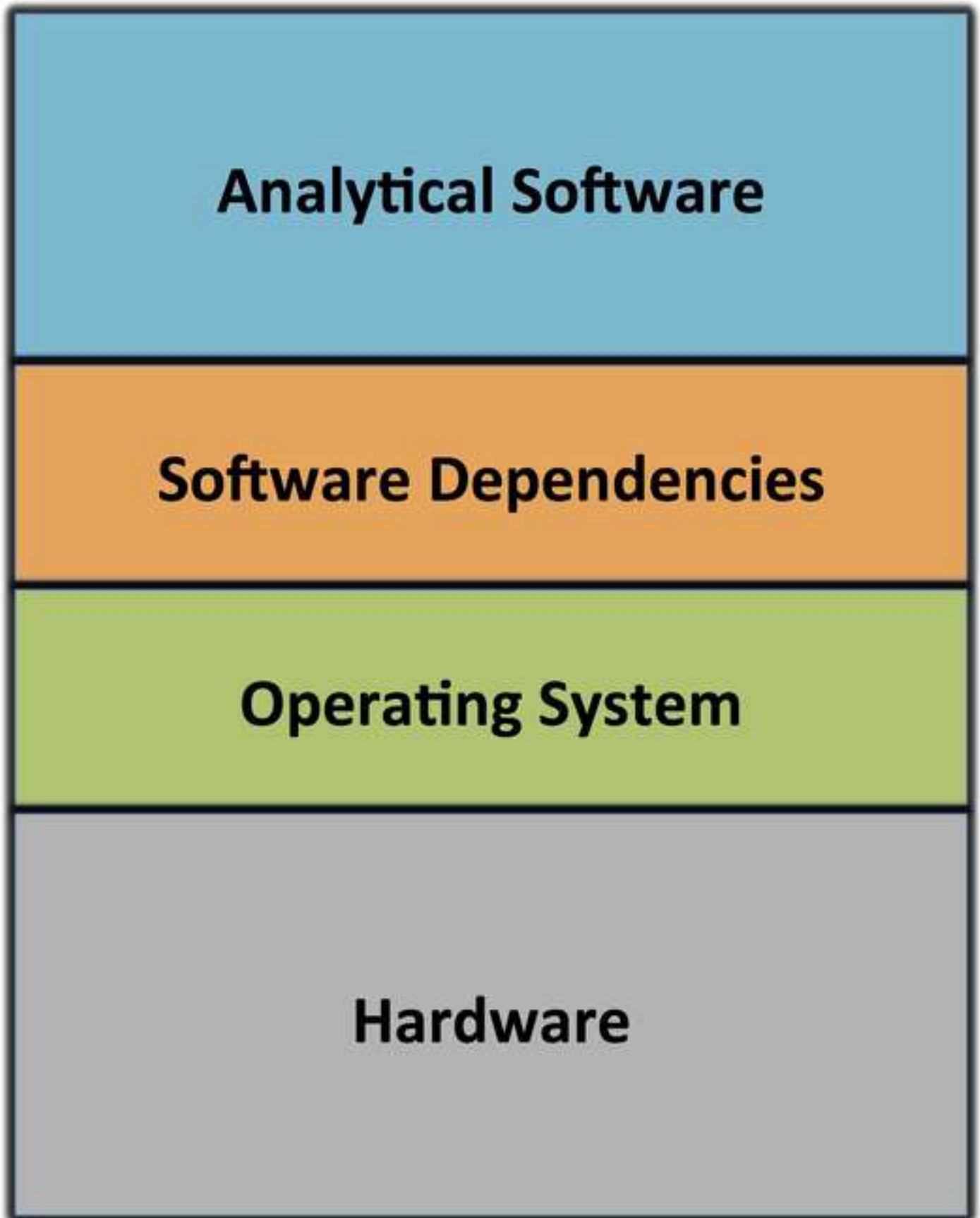
14 103. Bird I. Computing for the Large Hadron Collider. *Annu. Rev. Nucl. Part. Sci.* 2011;61:99–118.

15 104. Anderson DP. BOINC: A System for Public Resource Computing and Storage. Proceedings of the
16 Fifth IEEE/ACM International Workshop on Grid Computing (GRID’04). 2004.

17 105. Ransohoff DF. Bias as a threat to the validity of cancer molecular-marker research. *Nat. Rev.*
18 *Cancer*. 2005;5:142–9.




19 106. Bild AH, Chang JT, Johnson WE, Piccolo SR. A field guide to genomics research. *PLoS Biol.*
20 2014;12:e1001744.


21 107. Ratcliffe S. *Concise Oxford Dictionary of Quotations*. Oxford University Press; 2011. p. 389.















```
1  #!/bin/bash
2
3  # Download software, reference genome, and FASTQ files
4  wget http://downloads.sourceforge.net/project/bio-bwa/bwakit/bwakit-0.7.12_x64-linux.tar.bz2
5  wget -O refGenomeFiles.zip https://ndownloader.figshare.com/files/4841809
6  wget -O FASTQ.zip https://ndownloader.figshare.com/articles/3114454/versions/1
7
8  # Extract software and reference genome files
9  tar -jxvf bwakit-0.7.12_x64-linux.tar.bz2
10 unzip refGenomeFiles.zip
11 unzip FASTQ.zip
12
13 # Align FASTQ data to reference genome and save to SAM file
14 bwa.kit/bwa mem KJ660346.fa SRR1972917_1.fastq SRR1972917_1.fastq > SRR1972917_aligned.sam
15
16 # Convert SAM file to BAM file
17 bwa.kit/samtools view -bS SRR1972917_aligned.sam > SRR1972917_aligned.bam
18
19 # Sort BAM file
20 bwa.kit/samtools sort SRR1972917_aligned.bam SRR1972917_aligned_sorted
21
22 # Index BAM file
23 bwa.kit/samtools index SRR1972917_aligned_sorted.bam
```

```
1 all: download prepare align buildbam
2
3 buildbam: SRR1972917_aligned.sam
4     # Convert SAM file to BAM file
5     bwa.kit/samtools view -bS SRR1972917_aligned.sam > SRR1972917_aligned.bam
6
7     # Sort BAM file
8     bwa.kit/samtools sort SRR1972917_aligned.bam SRR1972917_aligned_sorted
9
10    # Index BAM file
11    bwa.kit/samtools index SRR1972917_aligned_sorted.bam
12
13 align: bwa.kit/bwa SRR1972917_1.fastq SRR1972917_2.fastq KJ660346.fa
14     # Align FASTQ data to reference genome and save to SAM file
15     bwa.kit/bwa mem KJ660346.fa SRR1972917_1.fastq SRR1972917_1.fastq > SRR1972917_aligned.sam
16
17 download:
18     # Download software, reference genome, and FASTQ files
19     wget http://downloads.sourceforge.net/project/bio-bwa/bwakit/bwakit-0.7.12_x64-linux.tar.bz2
20     wget -O refGenomeFiles.zip https://ndownloader.figshare.com/files/4841809
21     wget -O FASTQ.zip https://ndownloader.figshare.com/articles/3114454/versions/1
22
23 prepare: refGenomeFiles.zip FASTQ.zip bwakit-0.7.12_x64-linux.tar.bz2
24     # Extract software and reference genome files
25     tar -jxvf bwakit-0.7.12_x64-linux.tar.bz2
26     unzip refGenomeFiles.zip
27     unzip FASTQ.zip
```

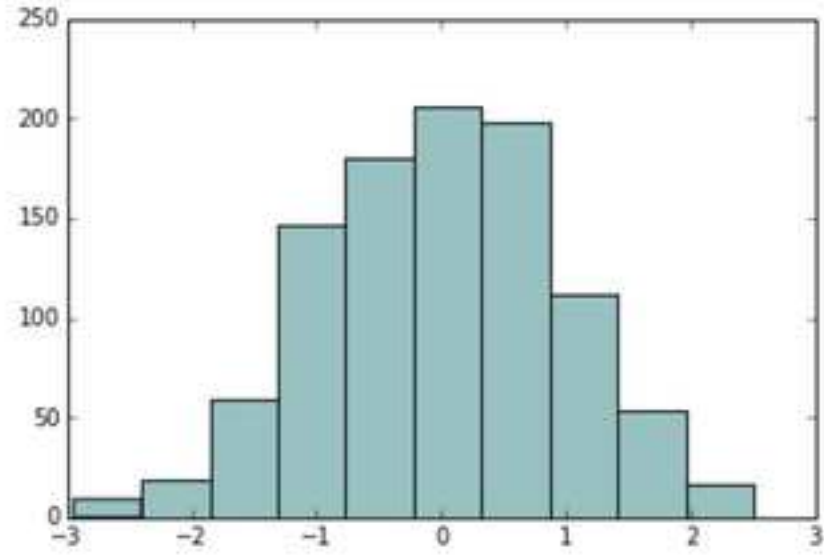
 **jupyter** Example (autosaved)  on 

File Edit View Insert Cell Kernel Help Python 3 

          Code  Cell Toolbar: None 

```
In [1]: get_ipython().magic('matplotlib inline')
import matplotlib.pyplot as plt
from numpy.random import normal
```

```
In [2]: nums = normal(size=1000)
plt.hist(nums, color="#99C1C2")
plt.show()
```



A histogram showing the distribution of 1000 random numbers generated from a normal distribution. The x-axis ranges from -3 to 3, and the y-axis ranges from 0 to 250. The bars are colored teal. The distribution is centered around 0, with the highest frequency (approximately 205) occurring between 0 and 1.

```
In [ ]:
```

R Markdown Example

1. Set the random seed.

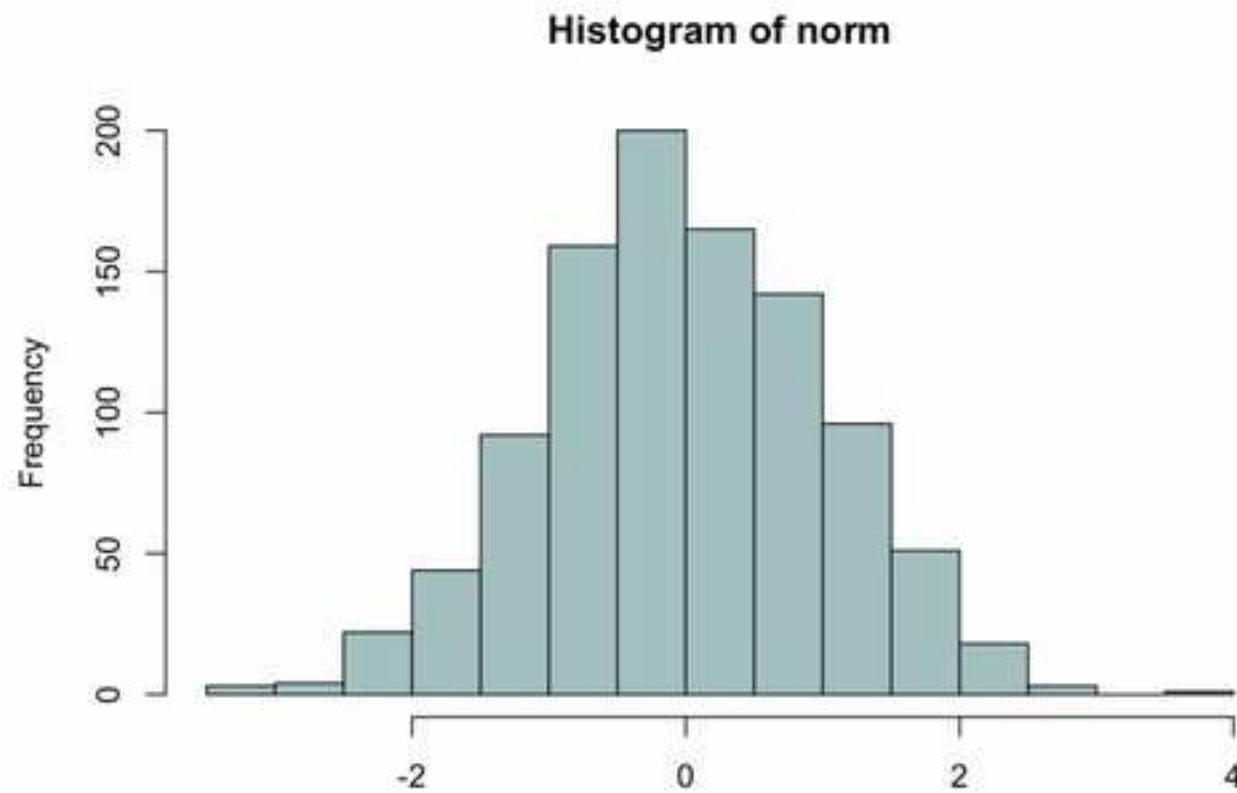
```
set.seed(99)
```

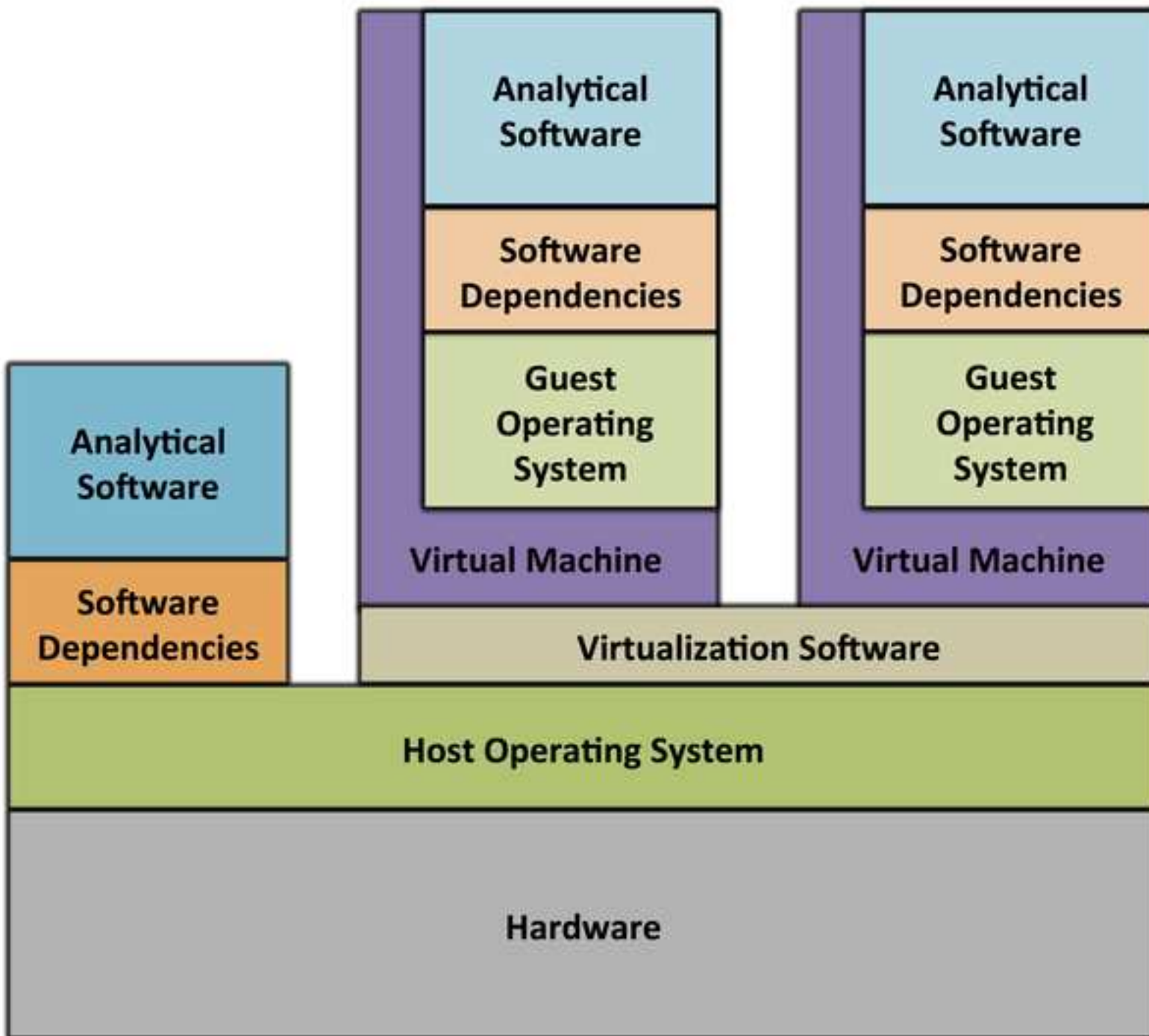
2. Generate 1000 random normal numbers

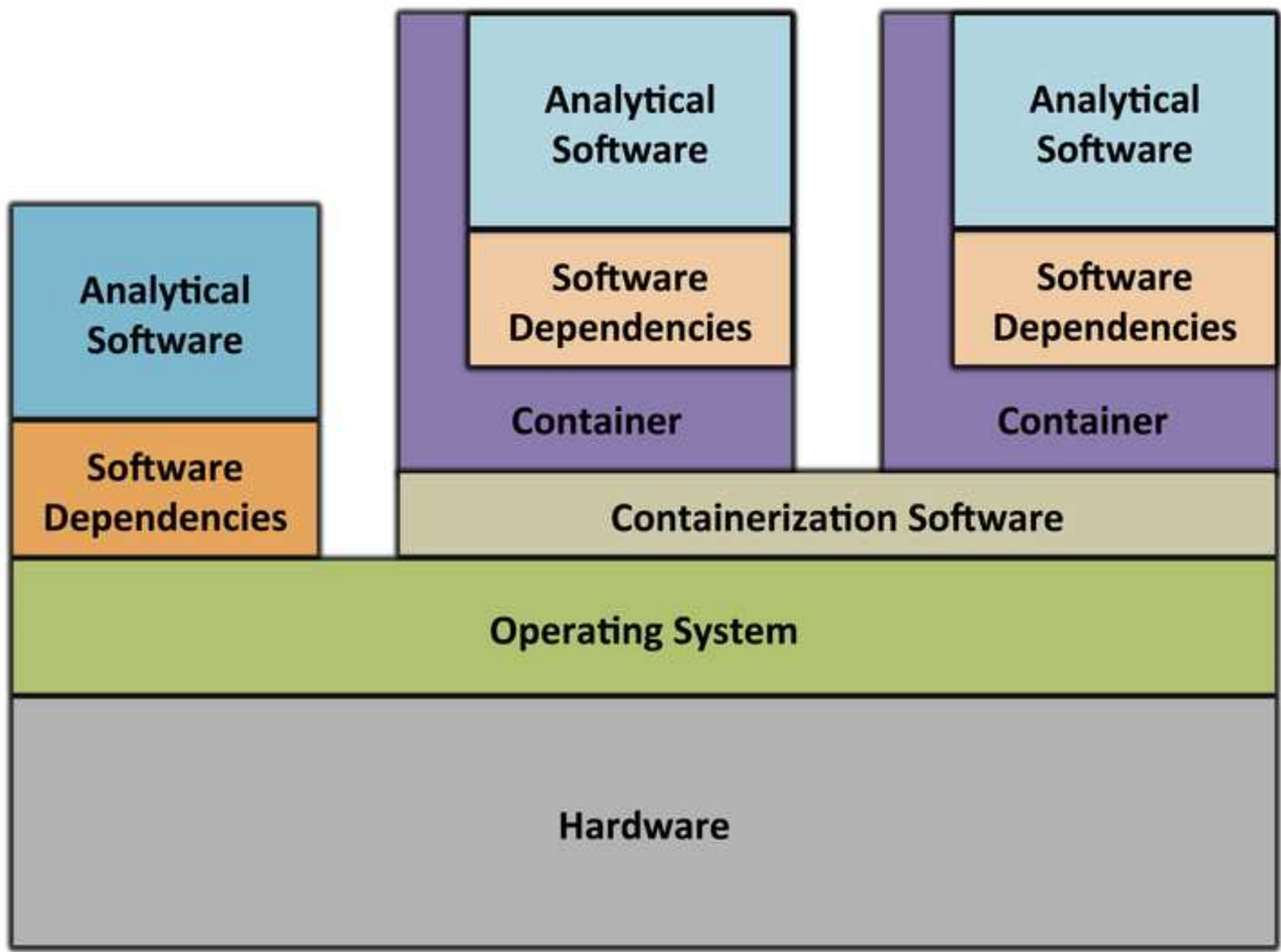
```
norm <- rnorm(1000)
```

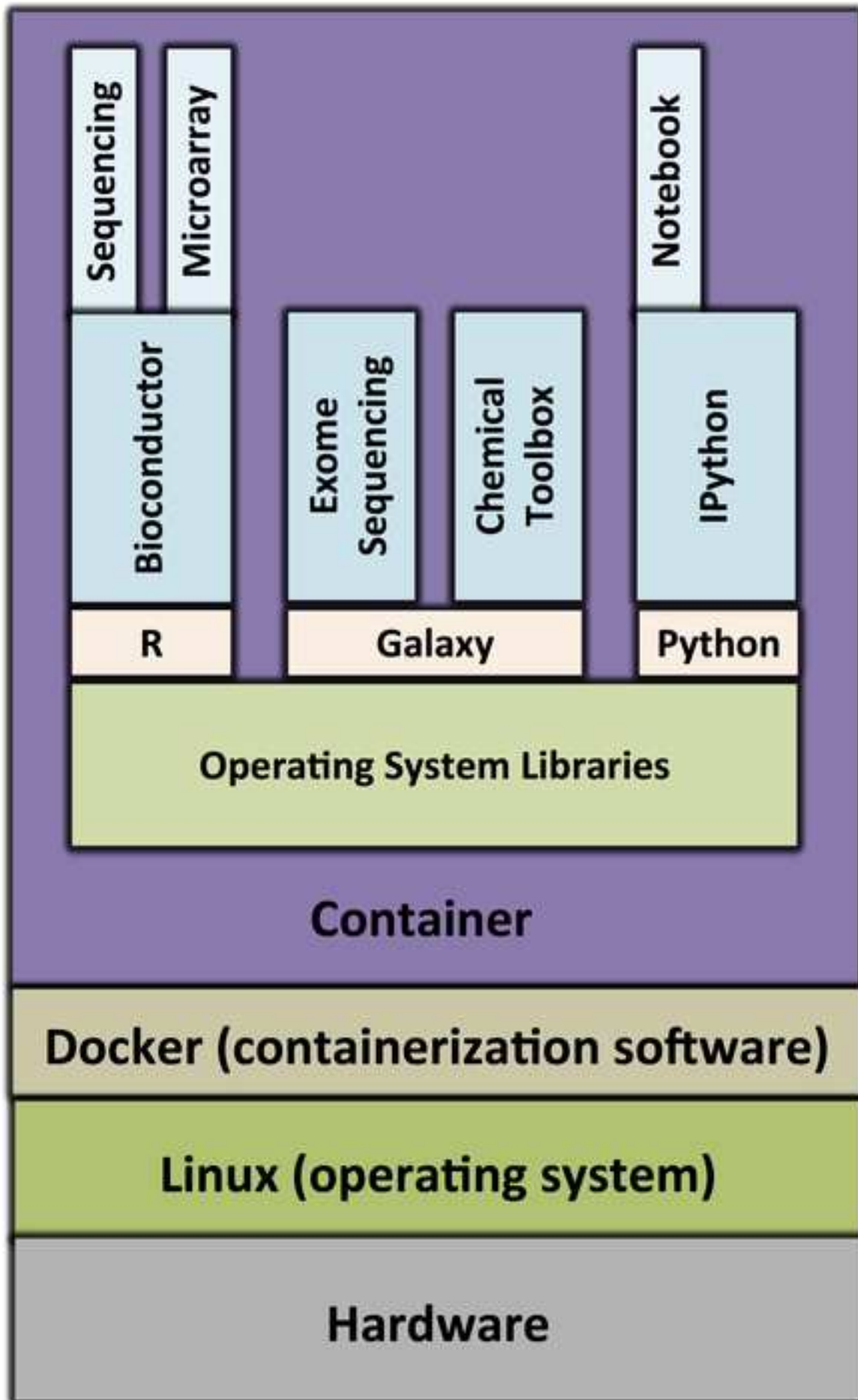
3. Plot them as a histogram

```
hist(norm, col="#99C1C2")
```









GigaScience

Tools and techniques for computational reproducibility

--Manuscript Draft--

Manuscript Number:	GIGA-D-16-00020R1
Full Title:	Tools and techniques for computational reproducibility
Article Type:	Review
Funding Information:	
Abstract:	<p>When reporting research findings, scientists document the steps they followed so that others can verify and build upon the research. When those steps have been described in sufficient detail that others can retrace the steps and obtain similar results, the research is said to be reproducible. Computers play a vital role in many research disciplines and present both opportunities and challenges for reproducibility. Computers can be programmed to execute analysis tasks, and those programs can be repeated and shared with others. The deterministic nature of most computer programs means that the same analysis tasks, applied to the same data, will often produce the same outputs. However, in practice, computational findings often cannot be reproduced because of complexities in how software is packaged, installed, and executed—and because of limitations associated with how scientists document analysis steps. Many tools and techniques are available to help overcome these challenges; here we describe seven such strategies. With a broad scientific audience in mind, we describe the strengths and limitations of each approach, as well as the circumstances under which each might be applied. No single strategy is sufficient for every scenario; thus we emphasize that it is often useful to combine approaches.</p>
Corresponding Author:	Stephen R Piccolo, Ph.D. Brigham Young University Provo, UT UNITED STATES
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Brigham Young University
Corresponding Author's Secondary Institution:	
First Author:	Stephen R Piccolo, Ph.D.
First Author Secondary Information:	
Order of Authors:	Stephen R Piccolo, Ph.D. Michael B Frampton
Order of Authors Secondary Information:	
Response to Reviewers:	<p>Dear Dr. Edmunds:</p> <p>We thank you and the reviewers for providing helpful comments on our review article, "Tools and techniques for computational reproducibility" (GIGA-D-16-00020). We have edited the manuscript according to the reviewers' suggestions. We have also added a reference to several papers that have used a Docker container for their analysis, and to Eglen, et al., which used knitr. These changes have improved our manuscript.</p> <p>Below is a point-by-point response to the reviewers' comments.</p> <p>Thank you for considering our article. Please let me know if we can answer any questions you may have.</p> <p>Warm regards,</p> <p>Stephen R. Piccolo, PhD Assistant Professor</p>

Department of Biology
Brigham Young University
(801) 422-7116
Stephen_Piccolo@byu.edu

** In the point-by-point replies below, the reviewer's comments are shown first. Our responses are prefixed by >> characters. **

Reviewer #1 (Titus Brown)

In this paper, Piccolo et al. do a nice (and I think comprehensive?) job of outlining strategies for computational reproducibility. The point is well made that science is increasingly dependent on computational reproducibility (and that in theory we should be able to do computational reproducibility easily and well) and hence we should explore effective approaches that are actually being used.

I know of no other paper that covers this array of material, and this is a quite nice exposition that I would recommend to many. I can't evaluate how broadly it will appeal to a diverse audience but it seems very readable to me.

>> We thank the reviewer for taking time to review our manuscript and for these positive comments!

This paper is a nice complement to Ten Simple Rules..., <http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003285>, in that it is longer, more in depth, and provides more explicit recommendations. It is also more up to date.

>> Thanks for this comment. Our goal, in part, was to extend beyond what the Ten Simple Rules... article covers. We cite that article multiple times within our manuscript.

I reviewed this paper previously, and it has been updated in light of my (and other) reviewers' comments. I was positive about it then, and the revisions are good ;). One note is that the 'binder' service (<http://mybinder.org>) should be mentioned as a harbinger of the future. I have a blog post summary of it here: ivory.idyll.org/blog/2016-mybinder.html

Signed,
C. Titus Brown
ctbrown@ucdavis.edu

>> Thanks for this suggestion. We have updated the second sentence of the Discussion section. We mention Binder (and Everware) and describe them as potential "harbingers of the future." (I hope it's OK that we borrowed this phrase from Titus.)

Reviewer #2 (Stephen Eglén)

Summary

This review provides an overview of the main tools and techniques used to ensure computational reproducibility of results. This review is a good fit to the readership of the Gigascience journal.

>> We thank the reviewer for taking time to review this manuscript and for these positive comments!

Overall I found the manuscript to be clearly written and would recommend it for publication, although I would like the authors to consider the following issues in a potential revision. In particular, I am most critical about the current set of figures. As a general comment, where possible, the material in the figures should be supported by

online files (e.g. knitr, Docker) so that the reader can immediately get workable examples to examine, run and amend.

>> We have addressed these comments below. We now provide executable files to complement Figures 1-4. We have included these in the supplementary material. We have also improved these materials based on the reviewer's comments.

Detailed comments

p4, l2 (page 4, line 2): an extra citation might be (Gronenschild et al. 2012) which showed that even the choice of operating system or neuroimaging software version affects results.

>> We have added this citation at the suggested location.

p4, l18-20: as far as I am aware, journals do not encourage direct use of repositories like github or bitbucket because they offer no long-term storage. It is regarded much more appropriate to use permanent URLs (e.g. DOIs) to point to archived versions of software, such as zenodo. We have written elsewhere on this topic (Eglen et al. 2016).

>> We have updated this text so that it no longer mentions GitHub and Bitbucket as examples of storage locations for archived versions of code. Now we mention Zenodo and figshare as services that provide permanent DOIs.

p6, l10: in addition to the other topics of reproducibility and education, it would be worth mentioning education/training to encourage users to adopt reproducible practices (Wilson 2016).

>> We have added a reference to Wilson 2016 and have mentioned "education about reproducibility" as a topic that is covered elsewhere.

p6, l15: minor point, but perhaps worth numbering the seven sections that describe the seven approaches reviewed, starting here.

>> We have numbered these sections as an aid to the reader.

p8, l17: "Make can be configured"; if you are referring to the "-j N" switch, then it is simpler to say that "Make can automatically identify. . ." as there is no extra configuration needed.

>> We have made this change to the text. Thanks for pointing it out.

p11 l9: reference 53 at the end of the sentence is not needed, as you refer to it at the start of the sentence.

>> We have removed this citation at the end of the sentence.

p13, l3-9: you should probably mention <http://mybinder.org> in this section. It provides a transparent method for interacting with Jupyter documents over the web (Rosenberg and Horn 2016).

>> Thanks for this suggestion. We have updated the second sentence of the Discussion section. We mention Binder (and Everware, a similar service) and cite the paper by Rosenberg and Horn.

p14, l2: I disagree slightly here with the view re: long-running jobs. knitr at least can cache intermediate computations transparently which helps enormously.

>> We have modified this paragraph to indicate that long-running notebooks can be executed at the command line.

p14, l8: Do you have examples of Dexy in use within this field? Asking this also in a more general way, it might be worth making a table listing examples explicitly of each of the seven approaches, so that they are easy to find.

>> We haven't found any substantive examples where Dexy is used in a biology-focused study. We have included it more as an indicator that alternative literate programming environments may be useful in cases where Jupyter and knitr are not suitable.

>> Although it may be useful to add a table that lists examples where each of the seven methods is used, we have cited various such studies throughout the text. We hope these references will provide useful examples, but we wish to avoid putting too much emphasis on any individual study.

p18, l18: I do not see why the VMs are "black boxes". Surely to create the VM all the relevant code must be provided, so that you can at least examine what is done, or extend the analysis (as mentioned on l20). Can you clarify what you mean here.

>> We have rewritten this paragraph to clarify our points. Our goal was to emphasize the importance of providing easy access to scripts and code that are used within a virtual machine for an analysis. Although it would be possible for other scientists to examine the full contents of a virtual machine, it is less convenient (and perhaps less transparent). So we encourage the idea of storing these scripts and code in a public repository, separate from the virtual machine. C. Titus Brown (reviewer #1) has also emphasized this point on his blog, which we cite: <http://ivory.idyll.org/blog/vms-considered-harmful.html>.

p22, l1: as well as capturing software in Docker, we have used it recently to capture the entire environment to write our research papers in knitr (<https://hub.docker.com/r/sje30/eglen2015/> and <https://hub.docker.com/r/sje30/waverepo/>).

>> Thanks for pointing to these. We now cite these papers as examples of enabling others to execute analyses described in research papers.

p22, l14-19: I found the section about other operating systems (Windows/mac) rather clumsy.

From the user's perspective, the modern docker toolbox seems to work smoothly enough (at least on macs) that the details at the end of p22 seem irrelevant. It might instead be worth mentioning that docker builds can be automatically triggered, e.g. upon new commits to github.

>> Thanks for this suggestion. Indeed, the Docker toolbox has been updated recently so that it is much more convenient to install and execute Docker on non-Linux machines. We have simplified the text accordingly. We have also clarified that the overhead of using a virtual machine to execute Docker containers on Windows or Mac operating systems offsets some of the performance benefits of containers.

p 23, l6: "Scientific advancement requires trust." I think trust is the wrong word here. e.g. The Royal Society's motto 'Nullius in verba' is taken to mean 'take nobody's word for it' (<https://royalsociety.org/about-us/history/>). Rather, what these tools do is promote transparency to reduce the barriers for others to repeat prior work. With this in mind, I'd suggest the authors re-read this first paragraph of the discussion to see whether they think "trust" is what they are promoting here.

>> Thanks for this insight. We have revised the first paragraph of the Discussion section. It no longer uses the word "trust" but instead focuses on the ideas of explicitly documenting research steps and being transparent about the way those steps are

shared.

Figures

Figure 1: is this really needed in such a review? I don't think it adds anything.

>> After considering this feedback, we agree that this figure is not needed, and we have removed it from the manuscript.

Figure 2: I think should be provided as a text file so that people can run it for themselves.

>> We now provide a text version of this script as a supplementary file.

Figure 3: Text file definitely needed so people can run it for themselves. But also I'd consider making the targets a bit more specific. All of the targets in this example Makefile are PHONY and perhaps it could be rewritten in the more canonical Makefile style? The way it is currently written, it is hard to see how this differs from a shell script. (Each time make all is run, won't the files be downloaded again?)

>> Thanks for pointing this out. We have modified the example Makefile so that it follows a more conventional style. Now it will only download the files (and execute the other steps) one time (if the steps executed successfully). In addition, we now provide a text version as a supplementary file.

Figure 4: can you show something that is a bit more bio-relevant, e.g. some genomic analysis?

>> We have updated this figure so that it shows a plot of simulated gene-expression data.

Figure 5: as figure 4. Can you design Figure 4 and 5 carefully to highlight the differences between knitr and jupyter?

>> We have updated this figure so it also shows a plot simulated-gene expression data. We are unsure which differences between knitr and Jupyter that the reviewer would like us to point out. However, we believe the updated figures show a more distinct difference between the two.

Figure 6-8: I would suggest you drop these. They show the notion of stacks and containers, but most bioscientists won't get much value from them.

>> We respect the reviewer's point of view; however, given feedback from scientists who have read our preprint, we feel that some scientists will benefit from these figures. Thus we prefer to retain them.

References

Eglen, Stephen, Ben Marwick, Yaroslav Halchenko, Michael Hanke, Shoaib Sufi, Padraig Gleeson, R Angus Silver, et al. 2016. "Towards Standard Practices for Sharing Computer Code and Programs in Neuroscience." *BioRxiv*. doi:10.1101/045104.

Gronenschild, Ed H B M, Petra Habets, Heidi I L Jacobs, Ron Mengelers, Nico Rozendaal, Jim van Os, and Machteld Marcelis. 2012. "The Effects of FreeSurfer Version, Workstation Type, and Macintosh Operating System Version on Anatomical Volume and Cortical Thickness Measurements." *PLoS One* 7 (6): e38234. doi:10.1371/journal.pone.0038234.

Rosenberg, David M, and Charles C Horn. 2016. "Neurophysiological Analytics for All! Free OpenSource Software Tools for Documenting, Analyzing, Visualizing, and Sharing Using Electronic Notebooks." *J. Neurophysiol.*, 20~apr, jn.00137.2016. doi:10.1152/jn.00137.2016.

Wilson, Greg. 2016. "Software Carpentry: Lessons Learned." F1000Res. 3 (28~jan). doi:10.12688/f1000research.3-62.v2.

Additional Information:

Question

Response

Experimental design and statistics

No

Full details of the experimental design and statistical methods used should be given in the Methods section, as detailed in our [Minimum Standards Reporting Checklist](#). Information essential to interpreting the data presented should be made available in the figure legends.

Have you included all the information requested in your manuscript?

If not, please give reasons for any omissions below.

as follow-up to "**Experimental design and statistics**

Full details of the experimental design and statistical methods used should be given in the Methods section, as detailed in our [Minimum Standards Reporting Checklist](#). Information essential to interpreting the data presented should be made available in the figure legends.

Have you included all the information requested in your manuscript?

"

It is a review article.

Resources

No

A description of all resources used, including antibodies, cell lines, animals and software tools, with enough information to allow them to be uniquely identified, should be included in the Methods section. Authors are strongly encouraged to cite [Research Resource Identifiers](#) (RRIDs) for antibodies, model organisms and tools, where possible.

Have you included the information requested as detailed in our [Minimum Standards Reporting Checklist](#)?

<p>If not, please give reasons for any omissions below.</p> <p>as follow-up to "Resources</p> <p>A description of all resources used, including antibodies, cell lines, animals and software tools, with enough information to allow them to be uniquely identified, should be included in the Methods section. Authors are strongly encouraged to cite Research Resource Identifiers (RRIDs) for antibodies, model organisms and tools, where possible.</p> <p>Have you included the information requested as detailed in our Minimum Standards Reporting Checklist?</p> <p>"</p>	<p>It is a review article.</p>
<p>Availability of data and materials</p> <p>All datasets and code on which the conclusions of the paper rely must be either included in your submission or deposited in publicly available repositories (where available and ethically appropriate), referencing such data using a unique identifier in the references and in the "Availability of Data and Materials" section of your manuscript.</p> <p>Have you have met the above requirement as detailed in our Minimum Standards Reporting Checklist?</p>	<p>No</p>
<p>If not, please give reasons for any omissions below.</p> <p>as follow-up to "Availability of data and materials</p> <p>All datasets and code on which the conclusions of the paper rely must be either included in your submission or deposited in publicly available repositories (where available and ethically appropriate), referencing such data using a unique identifier in the references and in the "Availability of Data and Materials" section of your manuscript.</p> <p>Have you have met the above requirement as detailed in our Minimum</p>	<p>It is a review article.</p>

[Standards Reporting Checklist?](#)

"

[Click here to view linked References](#)

Tools and techniques for computational reproducibility

Stephen R. Piccolo^{1*}, Michael B. Frampton²

¹Department of Biology, Brigham Young University, Provo, UT, USA

²Department of Computer Science, Brigham Young University, Provo, UT, USA

* Corresponding author

4102 LSB

Brigham Young University

Provo

UT 84602

USA

Telephone: +1 801-422-7116

Email: stephen_piccolo@byu.edu

Twitter: @stevepiccolo

ORCID: 0000-0003-2001-5640

1
2
3
4 **Abstract**
5
6
7

8 When reporting research findings, scientists document the steps they followed so that
9
10 others can verify and build upon the research. When those steps have been described in
11
12 sufficient detail that others can retrace the steps and obtain similar results, the research is
13
14 said to be reproducible. Computers play a vital role in many research disciplines and
15
16 present both opportunities and challenges for reproducibility. Computers can be
17
18 programmed to execute analysis tasks, and those programs can be repeated and shared
19
20 with others. The deterministic nature of most computer programs means that the same
21
22 analysis tasks, applied to the same data, will often produce the same outputs. However, in
23
24 practice, computational findings often cannot be reproduced because of complexities in
25
26 how software is packaged, installed, and executed—and because of limitations associated
27
28 with how scientists document analysis steps. Many tools and techniques are available to
29
30 help overcome these challenges; here we describe seven such strategies. With a broad
31
32 scientific audience in mind, we describe the strengths and limitations of each approach, as
33
34 well as the circumstances under which each might be applied. No single strategy is
35
36 sufficient for every scenario; thus we emphasize that it is often useful to combine
37
38 approaches.
39
40
41
42
43
44
45
46
47

48 **Keywords:** Computational reproducibility; Practice of science; Literate programming;
49
50 Virtualization; Software containers; Software frameworks
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

Introduction

When reporting research, scientists document the steps they followed to obtain their results. If the description is comprehensive enough that they and others can repeat the procedures and obtain semantically consistent results, the findings are considered to be “reproducible” [1–6]. Reproducible research forms the basic building blocks of science, insofar as it allows researchers to verify and build on each other's work with confidence.

Computers play an increasingly important role in many scientific disciplines [7–10]. For example, in the United Kingdom, 92% of academic scientists use some type of software in their research, and 69% of scientists say their research is feasible only with software tools [11]. Thus efforts to increase scientific reproducibility should consider the ubiquity of computers in research.

Computers present both opportunities and challenges for scientific reproducibility. On one hand, the deterministic nature of most computer programs means that identical results can be obtained from many computational analyses applied to the same input data [12].

Accordingly, computational research can be held to a high reproducibility standard. On the other hand, even when no technical barrier prevents reproducibility, scientists often cannot reproduce computational findings because of complexities in how software is packaged, installed, and executed—and because of limitations associated with how scientists document these steps [13]. This problem is acute in many disciplines, including genomics, signal processing, and ecological modeling [14–16], which have large data sets and rapidly evolving computational tools. However, the same problem can affect any

1
2
3
4 scientific discipline requiring computers for research. Seemingly minor differences in
5
6 computational approaches can have major influences on analytical outputs [12, 17–22],
7
8 and the effects of these differences may exceed those resulting from experimental factors
9
10 [23].
11
12
13

14
15 Journal editors, funding agencies, governmental institutions, and individual scientists have
16
17 increasingly made calls for the scientific community to embrace practices to support
18
19 computational reproducibility [24–31]. This movement has been motivated, in part, by
20
21 scientists' failed efforts to reproduce previously published analyses. For example, Ioannidis
22
23 et al. evaluated 18 published research studies that used computational methods to evaluate
24
25 gene expression data, but they were able to reproduce only two of those studies [32]. In
26
27 many cases, the culprit was a failure to share the study's data; however, incomplete
28
29 descriptions of software-based analyses were also common. Nekrutenko and Taylor
30
31 examined 50 papers that analyzed next-generation sequencing data and observed that
32
33 fewer than half provided any details about software versions or parameters [33].
34
35
36
37
38

39
40 Recreating analyses that lack such details can require hundreds of hours of effort [34] and
41
42 may be impossible, even after consulting the original authors. Failure to reproduce
43
44 research may also lead to careerist effects, including retractions [35].
45
46
47

48
49 Noting such concerns, some journals have emphasized the value of placing computer code
50
51 in open access repositories. It is most useful when scientists provide direct access to an
52
53 archived version of the code via a uniform resource locator (URL). For example, Zenodo.org
54
55 and figshare.com provide permanent digital object identifiers (DOI) that can link to
56
57
58
59
60
61
62
63
64
65

1
2
3
4 software code (and other digital objects) used in publications. In addition, some journals
5
6 have extended requirements for “Methods” sections, now asking researchers to provide
7
8 detailed descriptions of 1) how to install software and its dependencies, and 2) what
9
10 parameters and data preprocessing steps are used in analyses [10, 24]. A 2012 Institute of
11
12 Medicine report emphasized that, in addition to computer code and research data, “fully
13
14 specified computational procedures” should be made available to the scientific community
15
16 [25]. The report’s authors elaborated that such procedures should include “all of the steps
17
18 of computational analysis”, and that “all aspects of the analysis need to be transparently
19
20 reported” [25]. Such policies represent important progress. However, it is ultimately the
21
22 responsibility of individual scientists to ensure that others can verify and build upon their
23
24 analyses.
25
26
27
28
29
30

31
32 Describing a computational analysis sufficiently—such that others can re-execute, validate,
33
34 and refine it—requires more than simply stating what software was used, what commands
35
36 were executed, and where to find the source code [13, 27, 36–38]. Software is executed
37
38 within the context of an operating system (for example, Windows, Mac OS, or Linux), which
39
40 enables the software to interface with computer hardware. In addition, most software
41
42 relies on a hierarchy of software dependencies, which perform complementary functions
43
44 and must be installed alongside the main software tool. One version of a given software
45
46 tool or dependency may behave differently or have a different interface than another
47
48 version of the same software. In addition, most analytical software offers a range of
49
50 parameters (or settings) that the user can specify. If any of these variables differs from
51
52 those used by the original experimenter, the software may not execute properly or
53
54
55
56
57
58

1
2
3
4 analytical outputs may differ considerably from those observed by the original
5
6
7 experimenter.
8
9

10 Scientists can use various tools and techniques to overcome these challenges and to
11
12 increase the likelihood that their computational analyses will be reproducible. These
13
14 techniques range in complexity from simple (e.g., providing written documentation) to
15
16 advanced (e.g., providing a virtual environment that includes an operating system and all
17
18 the software necessary to execute the analysis). This review describes seven strategies
19
20 across this spectrum. We describe many of the strengths and limitations of each approach,
21
22 as well as the circumstances under which each might be applied. No single strategy will be
23
24 sufficient for every scenario; therefore, in many cases, it will be most practical to combine
25
26 multiple approaches. This review focuses primarily on the computational aspects of
27
28 reproducibility. The related topics of empirical reproducibility, statistical reproducibility,
29
30 data sharing, and education about reproducibility have been described elsewhere [39–46].
31
32 We believe that with greater awareness and understanding of computational
33
34 reproducibility techniques, scientists—including those with limited computational
35
36 experience—will be more apt to perform computational research in a reproducible
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1
2
3
4 **1. Narrative descriptions are a simple but valuable way to support**
5
6
7 **computational reproducibility**
8
9

10 The most fundamental strategy for enabling others to reproduce a computational analysis
11 is to provide a detailed, written description of the process. For example, when reporting
12 computational results in a research article, authors customarily provide a narrative that
13 describes the software they used and the analytical steps they followed. Such narratives
14 can be invaluable in enabling others to evaluate the scientific approach and to reproduce
15 the findings. In many situations—for example, when software execution requires user
16 interaction or when proprietary software is used—narratives are the only feasible option
17 for documenting such steps. However, even when a computational analysis uses open-
18 source software and can be fully automated, narratives help others understand how to re-
19 execute an analysis.
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

36 Although most articles about research that uses computational methods provide some type
37 of narrative, these descriptions often lack sufficient detail to enable others to retrace those
38 steps [32, 33]. Narrative descriptions should indicate the operating system(s), software
39 dependencies, and analytical software that were used, and how to obtain them. In addition,
40 narratives should indicate the exact software versions used, the order in which they were
41 executed, and all non-default parameters that were specified. Such descriptions should
42 account for the fact that computer configurations can differ vastly, even for computers with
43 the same operating system. Because it can be difficult for scientists to remember such
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1
2
3
4 details after the fact, it is best to record this information throughout the research process,
5
6 rather than at the time of manuscript preparation [8].
7
8
9

10 The following sections describe techniques for automating computational analyses. These
11
12 techniques can diminish the need for scientists to write narratives. However, because it is
13
14 often impractical to automate all computational steps, we expect that, for the foreseeable
15
16 future, narratives will play a vital role in enabling computational reproducibility.
17
18
19
20
21

22 **2. Custom scripts and code can automate research analysis**

23
24

25 Scientific software can often be executed in an automated manner via text-based
26
27 commands. Using such commands—via a command-line interface—scientists can indicate
28
29 the software program(s) to be executed and which parameter(s) should be used. When
30
31 multiple commands must be executed, they can be compiled into scripts specifying the
32
33 order in which the commands should be executed (Figure 1; Additional file 1). In many
34
35 cases, scripts also include commands for installing and configuring software. Such scripts
36
37 serve as valuable documentation not only for individuals who wish to re-execute the
38
39 analysis, but also for the researcher who performed the original analysis [47]. In these
40
41 cases, no amount of narrative is an adequate substitute for providing the actual commands
42
43 that were used.
44
45
46
47
48
49
50

51 When writing command-line scripts, it is essential to explicitly document any software
52
53 dependencies and input data that are required for each step in the analysis. The Make
54
55 utility [114] provides one way to specify such requirements [36]. Before any command is
56
57
58
59
60
61
62
63
64
65

1
2
3
4 executed, Make verifies that each documented dependency is available. Accordingly,
5
6 researchers can use Make files (scripts) to specify a full hierarchy of operating system
7
8 components and dependent software that must be present to perform the analysis (Figure
9
10 2; Additional file 2). In addition, Make can automatically identify any commands that can be
11
12 executed in parallel, potentially reducing the amount of time required for the analysis.
13
14 Although Make was originally designed for UNIX-based operating systems (such as Mac OS
15
16 or Linux), similar utilities have since been developed for Windows operating systems
17
18 [116]. Box 1 lists various utilities that can be used to automate software execution.
19
20
21
22
23
24

25
26 **Box 1. Utilities that can be used to automate software execution**
27

- 28
29 ● GNU Make and Make for Windows: tools for building software from source files
30 and for ensuring that the software's dependencies are met.
31
32
- 33
34 ● Snakemake [48]: an extension of Make that provides a more flexible syntax and
35 makes it easier to execute tasks in parallel.
36
37
- 38
39 ● BPipe [49]: a tool that provides a flexible syntax for users to specify commands to
40 be executed; it maintains an audit trail of all commands that have been executed.
41
42
- 43
44 ● GNU Parallel [50]: a tool for executing commands in parallel across one or more
45 computers.
46
47
- 48
49 ● Makeflow [51]: a tool that can execute commands simultaneously on various
50
51
52
53
54
55
56

1
2
3
4
5 types of computer architectures, including computer clusters and cloud
6 environments.
7
8

- 9
- 10 ● SCONS [52]: an alternative to GNU Make that enables users to customize the
11 process of building and executing software using scripts written in the Python
12 programming language.
13
14
 - 15 ● CMAKE.org: a tool that enables users to execute Make scripts more easily on
16 multiple operating systems.
17
18
19
20
21
22
23
24
25

26
27 As well as creating scripts to execute existing software, many researchers also create new
28 software by writing computer code in a programming language such as Python, C++, Java,
29 or R. Such code may perform relatively simple tasks, such as reformatting data files or
30 invoking third-party software. In other cases, computer code may constitute a manuscript's
31 key intellectual contribution.
32
33
34
35
36
37
38
39

40 Whether analysis steps are encoded in scripts or as computer code, scientists can support
41 reproducibility by publishing these artifacts alongside research papers. By doing so,
42 authors enable readers to evaluate the analytical approach in full detail and to extend the
43 analysis more readily [53]. Although scripts and code may be included alongside a
44 manuscript as supplementary material, a better alternative is to store them in a public
45 repository with a permanent URL. It is often also useful to store code in a version control
46 system (VCS) [8, 9, 47], and to share it via Web-based services like GitHub.com or
47 Bitbucket.org [54]. With such a VCS repository, scientists can track the different versions of
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1
2
3
4 scripts and code that have been developed throughout the evolution of the research
5
6 project. In addition, outside observers can see the full version history, contribute revisions
7
8 to the code, and reuse the code for their own purposes [55]. When submitting a
9
10 manuscript, the authors may “tag” a specific version of the repository that was used for the
11
12 final analysis described in the manuscript.
13
14
15
16
17
18

19 **3. Software frameworks enable easier handling of software** 20 21 **dependencies** 22 23 24

25 Virtually all computer scripts and code relies on external software dependencies and
26
27 operating system components. For example, suppose a research study required a scientist
28
29 to apply Student’s t-test. Rather than write code to implement this statistical test, the
30
31 scientist would likely find an existing software library that implements the test and then
32
33 invoke that library from their code. Much time can be saved with this approach, and a wide
34
35 range of software libraries are freely available. However, software libraries change
36
37 frequently; invoking the wrong version of a library may result in an error or an unexpected
38
39 output. Thus, to enable others to reproduce an analysis, it is critical to indicate which
40
41 dependencies (and versions thereof) must be installed.
42
43
44
45
46
47

48 One way to address this challenge is to build on a pre-existing software framework, which
49
50 makes it easier to access software libraries that are commonly used to perform specific
51
52 types of analysis task. Typically, such frameworks also make it easier to download and
53
54 install software dependencies, and to ensure that the versions of software libraries and
55
56
57
58
59
60
61
62
63
64
65

1
2
3
4 their dependencies are compatible with each other. For example, Bioconductor [56],
5
6 created for the R statistical programming language [57], is a popular framework that
7
8 contains hundreds of software packages for analyzing biological data. The Bioconductor
9
10 framework facilitates versioning, documenting, and distributing code. Once a software
11
12 library has been incorporated into Bioconductor, other researchers can find, download,
13
14 install, and configure it on most operating systems with relative ease. In addition,
15
16
17 Bioconductor installs software dependencies automatically. These features ease the
18
19 process of performing an analysis, and can help with reproducibility. Various software
20
21 frameworks exist for other scientific disciplines [58–63]. General purpose tools for
22
23 managing software dependencies also exist, for example, Apache Ivy [116] and Puppet
24
25 [117].
26
27
28
29
30

31
32 To best support reproducibility, software frameworks should make it easy for scientists to
33
34 download and install previous versions of a software tool, as well as previous versions of
35
36 dependencies. Such a design enables other scientists to reproduce analyses that were
37
38 conducted with previous versions of a software framework. In the case of Bioconductor,
39
40 considerable extra work may be required to install specific versions of Bioconductor
41
42 software and their dependencies. To overcome these limitations, scientists may use a
43
44 software container or virtual machine to package together the specific versions they used
45
46
47 in an analysis. Alternatively, they might use third-party solutions such as the aRchive
48
49 project [118].
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

4. Literate programming combines narratives with code

Although narratives, scripts, and computer code individually support reproducibility, there is additional value in combining these entities. Even though a researcher may provide computer code alongside a research paper, other scientists may have difficulty interpreting how the code accomplishes specific tasks. A longstanding way to address this problem is via code comments: human-readable annotations interspersed throughout computer code. However, code comments and other types of documentation often become outdated as code evolves throughout the analysis process [64]. One way to overcome this problem is to use a technique called literate programming [65]. In this approach, the scientist writes a narrative of the scientific analysis and intermingles code directly within the narrative. As the code is executed, a document is generated that includes the code, narratives, and any outputs (e.g., figures, tables) of the code. Accordingly, literate programming helps ensure that readers understand exactly how a particular research result was obtained. In addition, this approach motivates the scientist to keep the target audience in mind when performing a computational analysis, rather than simply to write code that a computer can parse [65]. Consequently, by reducing barriers of understanding among scientists, literate programming can help to engender greater trust in computational findings.

One popular literate programming tool is Jupyter [66]. Using Jupyter.org's Web-based interface, scientists can create interactive "notebooks" that combine code, data, mathematical equations, plots, and rich media [67]. Originally known as IPython, and previously designed exclusively for the Python programming language, Jupyter now makes

1
2
3
4 it possible to execute code in many different programming languages. Such functionality
5
6 may be important to scientists who prefer to combine the strengths of different
7
8 programming languages.
9

10
11 knitr [68] has also gained considerable popularity as a literate programming tool. It is
12
13 written in the R programming language, and thus can be integrated seamlessly with the
14
15 array of statistical and plotting tools available in that environment. However, like Jupyter,
16
17 knitr can execute code written in multiple programming languages. Commonly, knitr is
18
19 applied to documents that have been authored using RStudio [69], an open-source tool
20
21 with advanced editing and package management features.
22
23
24
25
26
27

28 Jupyter notebooks and knitr reports can be saved in various output formats, including
29
30 hypertext markup language (HTML) and portable document format (PDF; see examples in
31
32 Figures 3 and 4; Additional files 3 and 4). Increasingly, scientists include such documents
33
34 as supplementary materials to journal manuscripts, enabling others to repeat analysis
35
36 steps and recreate manuscript figures [70–73].
37
38
39
40

41 Scientists typically use literate programming tools for data analysis tasks that can be
42
43 executed interactively, in a modest amount of time (e.g., minutes or hours). However, it is
44
45 possible to execute Jupyter or knitr at the command line; thus longer running tasks can be
46
47 executed on high-performance computers.
48
49
50

51
52 Literate programming notebooks are suitable for research analyses that require a modest
53
54 amount of computer code. For analyses needing larger amounts of code, more advanced
55
56
57
58
59
60
61
62
63
64
65

1
2
3
4 programming environments may be more suitable, perhaps in combination with a “literate
5
6 documentation” tool such as DEXY.it.
7
8
9

10 11 **5. Workflow management systems enable software to be executed via a** 12 13 **graphical user interface** 14

15
16
17
18 Writing computer scripts and code seems daunting to many researchers. Although various
19
20 courses and tutorials are helping to make this task less formidable [46, 74–76], many
21
22 scientists use “workflow management systems” to facilitate the execution of scientific
23
24 software [77]. Typically managed via a graphical user interface, workflow management
25
26 systems enable scientists to upload data and process them using existing tools. For
27
28 multistep analyses, the output from one tool can be used as input to additional tools,
29
30 resulting in a series of commands known as a workflow.
31
32

33
34
35
36 Galaxy [78, 79] has gained considerable popularity within the bioinformatics community,
37
38 especially for performing next-generation sequencing analysis. As users construct
39
40 workflows, Galaxy provides descriptions of how software parameters should be used,
41
42 examples of how input files should be formatted, and links to relevant discussion forums.
43
44 To help with processing large data sets and computationally complex algorithms, Galaxy
45
46 also provides an option to execute workflows on cloud-computing services [80]. In
47
48 addition, researchers can share workflows with each other at UseGalaxy.org; this feature
49
50 has enabled the Galaxy team to build a community that encourages reproducibility, helps
51
52 define best practices, and reduce the time required for novices to get started.
53
54
55
56
57
58
59
60
61
62
63
64
65

1
2
3
4 Various other workflow systems are freely available to the research community (see Box
5
6 2). For example, VisTrails.org is used by researchers from many disciplines, including
7
8 climate science, microbial ecology, and quantum mechanics [81]. It enables scientists to
9
10 design visual workflows, and connect data inputs with analytical modules and the resulting
11
12 outputs. In addition, VisTrails tracks a full history of how each workflow was created. This
13
14 capability, referred to as “retrospective provenance”, makes it possible for others to not
15
16 only reproduce the final version of an analysis, but also to examine previous incarnations of
17
18 the workflow and how each change influenced the analytical outputs [82].
19
20
21
22
23
24

25
26 **Box 2: Workflow management tools freely available to the research community.**
27

- 28 ● Galaxy [78, 79]
 - 29 ● VisTrails [81]
 - 30 ● Kepler-project.org [83]
 - 31 ● CyVerse.org (formerly known as The iPlant Collaborative) [84]
 - 32 ● GenePattern [85, 86, 119]
 - 33 ● Taverna.org.uk [87]
 - 34 ● LONI Pipeline [88, 120]
- 35
36
37
38
39
40
41
42
43
44
45
46
47
48

49 Although workflow management systems offer many advantages, users must accept
50 tradeoffs. For example, although the teams that develop these tools often provide public
51 servers where users can execute workflows, many scientists share these resources, limiting
52 the computational power or storage space available to execute large-scale analyses in a
53
54
55
56
57
58

1
2
3
4 timely manner. As an alternative, many scientists install these systems on their own
5
6 computers; however, configuring and supporting them requires time and expertise. In
7
8 addition, if a workflow tool does not yet provide a module to support a given analysis, the
9
10 scientist must create one. This task constitutes additional overheads; however, utilities
11
12 such as the Galaxy Tool Shed [89] are helping to facilitate this process.
13
14
15
16
17

18 19 **6. Virtual machines encapsulate an entire operating system and software** 20 21 **dependencies**

22
23
24
25 Whether within a literate programming notebook, or via a workflow management system,
26
27 an operating system and relevant software dependencies must be installed before an
28
29 analysis is executed. The process of identifying, installing, and configuring such
30
31 dependencies consumes a considerable amount of scientists' time. Different operating
32
33 systems (and versions thereof) may require different installation and configuration steps.
34
35 Furthermore, earlier versions of software dependencies, which may currently be installed
36
37 on a given computer, may be incompatible with—or produce different outputs than—
38
39 newer versions.
40
41
42
43
44

45
46 One solution is to use virtual machines, which can encapsulate an entire operating system
47
48 and all software, scripts, code, and data necessary to execute a computational analysis [90,
49
50 91] (Figure 5). Using virtualization software such as VirtualBox or VMWare (see Box 3), a
51
52 virtual machine can be executed on practically any desktop, laptop, or server, irrespective
53
54 of the main (“host”) operating system on the computer. For example, even though a
55
56
57
58
59

1
2
3
4 scientist's computer may be running a Windows operating system, they may perform an
5
6 analysis on a Linux operating system that is running concurrently—within a virtual
7
8 machine—on the same computer. The scientist has full control over the virtual (“guest”)
9
10 operating system, and thus can install software and modify configuration settings as
11
12 necessary. In addition, a virtual machine can be constrained to use specific amounts of
13
14 computational resources (e.g., computer memory, processing power), thus enabling system
15
16 administrators to ensure that multiple virtual machines can be executed simultaneously on
17
18 the same computer without impacting each other's performance. After executing an
19
20 analysis, the scientist can export the entire virtual machine to a single, binary file. Other
21
22 scientists can then use this file to reconstitute the same computational environment that
23
24 was used for the original analysis. With a few exceptions (see Discussion), these scientists
25
26 will obtain exactly the same results as the original scientist. This process provides the
27
28 added benefits that 1) the scientist must only document the installation and configuration
29
30 steps for a single operating system, 2) other scientists need only install the virtualization
31
32 software and not individual software components, and 3) analyses can be re-executed
33
34 indefinitely, so long as the virtualization software remains compatible with current
35
36 computer systems [92]. The fact that a team of scientists can employ virtual machines to
37
38 ensure that each team member has the same computational environment is also useful
39
40 because team members may have different configurations on their host operating systems.
41
42 One criticism of using virtual machines to support computational reproducibility is that
43
44 virtual machine files are large (typically multiple gigabytes), especially if they include raw
45
46 data files. This imposes a barrier for researchers to share virtual machines with the
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1
2
3
4 research community. One option is to use cloud-computing services (see Box 4). Scientists
5
6 can execute an analysis in the cloud, take a “snapshot” of their virtual machine, and share it
7
8 with others in that environment [90, 93]. Cloud-based services typically provide
9
10 repositories where virtual machine files can easily be stored and shared among users.
11
12 Despite these advantages, some researchers may prefer their data to reside on local
13
14 computers, rather than in the cloud—at least while the research is being performed. In
15
16 addition, cloud-based services may use proprietary software, so virtual machines may only
17
18 be executable within each provider’s infrastructure. Furthermore, to use a cloud service
19
20 provider, scientists may need to activate a fee-based account.
21
22
23
24
25
26

27
28 When using virtual machines to support reproducibility, it is important that other scientists
29
30 can not only re-execute the analysis, but also examine the scripts and code used within the
31
32 virtual machine [94]. Although it is possible for others to examine the contents of a virtual
33
34 machine directly, it is preferable to store the scripts and code in public repositories—
35
36 separately from the virtual machine—so others can examine and extend the analysis more
37
38 easily [95]. In addition, scientists can use a virtual machine that has been prepackaged for a
39
40 particular research discipline. For example, CloudBioLinux contains a variety of
41
42 bioinformatics tools commonly used by genomics researchers [96]. The scripts for building
43
44 this virtual machine are stored in a public repository [121].
45
46
47
48
49

50
51 Scientists can automate the process of building and configuring virtual machines using
52
53 tools such as Vagrant or Vortex (see Box 3). For either tool, users can write text-based
54
55 configuration files that provide instructions for building virtual machines and allocating
56
57
58

1
2
3
4 computational resources to them. In addition, these configuration files can be used to
5
6 specify analysis steps [95]. Because these files are text based and relatively small (usually a
7
8 few kilobytes), scientists can share them easily and track different versions of the files via
9
10 source control repositories. This approach also mitigates problems that might arise during
11
12 the analysis stage. For example, even when a computer's host operating system must be
13
14 reinstalled because of a computer hardware failure, the virtual machine can be recreated
15
16
17
18
19 with relative ease.
20
21
22

23
24 **Box 3: Virtual machine software.**

25
26
27 Virtualization hypervisors:

- 28 ● VirtualBox.org (open source)
- 29 ● XenProject.org (open source)
- 30 ● VMWare.com (partially open source)

31
32
33
34
35
36
37
38
39 Virtual machine management tools:

- 40 ● VagrantUP.com (open source)
- 41 ● Vortex (open source) [122]

42
43
44
45
46
47
48
49
50
51
52
53
54 **Box 4: Commercial cloud-service providers.**
55
56
57
58

- [Amazon Web Services \[123\]](#)
- [Rackspace.com/Cloud](#)
- [Google Cloud Platform \[124\]](#)
- [Windows Azure \[125\]](#)

7. Software containers ease the process of installing and configuring dependencies

Software containers are a lighter weight alternative to virtual machines. Like virtual machines, containers encapsulate operating system components, scripts, code, and data into a single package that can be shared with others. Thus, as with virtual machines, analyses executed within a software container should produce identical outputs, irrespective of the underlying operating system or the software that may be installed outside the container (see Discussion for caveats). As is true for virtual machines, multiple containers can be executed simultaneously on a single computer, and each container may contain different software versions and configurations. However, whereas virtual machines include an entire operating system, software containers interface directly with the computer's main operating system and extend it as needed (Figure 6). This design provides less flexibility than virtual machines because containers are specific to a given type of

1
2
3
4 operating system; however, containers require considerably less computational overhead
5
6 than virtual machines, and can be initialized much more quickly [97].
7
8
9

10 The open source Docker.com utility, which has gained popularity among informaticians
11 since its release in 2013, provides the ability to build, execute, and share software
12 containers for Linux-based operating systems. Users specify a Docker container's contents
13 using text-based commands. These instructions can be placed in a "Dockerfile", which other
14 scientists can use to rebuild the container. As with virtual machine configuration files,
15 Dockerfiles are text based, so they can be shared easily, and can be tracked and versioned
16 in source control repositories. Once a Docker container has been built, its contents can be
17 exported to a binary file; these files are generally smaller than virtual machine files, so they
18 can be shared more easily—for example, via [hub.Docker.com](https://hub.docker.com).
19
20
21
22
23
24
25
26
27
28
29
30
31

32
33 A key feature of Docker containers is that their contents can be stacked in distinct layers
34 (or "images"). Each image includes software components to address a particular need.
35 Within a given research lab, scientists might create general purpose images to support
36 functionality for multiple projects, and specialized images to address the needs of specific
37 projects. An advantage of Docker's modular design is that when images within a container
38 are updated, Docker only needs to track the specific components that have changed; users
39 who wish to update to a newer version must download a relatively small update. In
40 contrast, even a minor change to a virtual machine would require users to export and
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1
2
3
4 Scientists have begun to share Docker images that enable others to execute analyses
5
6 described in research papers [98–100], and to facilitate benchmarking efforts among
7
8 researchers in a given subdiscipline. For example, nucleotid.es is a catalog of genome-
9
10 assembly tools that have been encapsulated in Docker images [101, 102]. Genome
11
12 assembly tools differ considerably in the dependencies they require, and in the parameters
13
14 they support. This project provides a means to standardize these assemblers, circumvent
15
16 the need to install dependencies for each tool, and perform benchmarks across the tools.
17
18 Such projects may help to reduce the reproducibility burden on individual scientists.
19
20
21
22
23

24
25 The use of Docker containers for reproducible research comes with caveats. Individual
26
27 containers are stored and executed in isolation from other containers on the same
28
29 computer; however, because all containers on a given machine share the same operating
30
31 system, this isolation is not as complete as it is with virtual machines. This means, for
32
33 example, that a given container is not guaranteed to have access to a specific amount of
34
35 computer memory or processing power—multiple containers may have to compete for
36
37 these resources [97]. In addition, containers may be more vulnerable to security breaches
38
39 [97]. Because Docker containers can only be executed on Linux-based operating systems,
40
41 they must be executed within a virtual machine on Windows and Mac operating systems.
42
43
44
45
46
47 Docker provides installation packages to facilitate this integration; however, the overhead
48
49 of using a virtual machine offsets some of the performance benefits of using containers.
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1
2
3
4 Efforts are ongoing to develop and refine software container technologies. Box 5 lists
5
6 various tools that are currently available. In the coming years, these technologies promise
7
8 to play an influential role within the scientific community.
9
10

11
12
13
14 **Box 5: Open-source containerization software.**

- 15 ● Docker.com
- 16
- 17 ● LinuxContainers.org
- 18
- 19 ● lmctfy [126]
- 20
- 21 ● OpenVZ.org
- 22
- 23 ● Warden [127]
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36

37 **Conclusions**

38
39
40 Scientific advancement requires researchers to explicitly document the research steps they
41
42 performed and to transparently share those steps with other researchers. This review
43
44 provides a comprehensive, though not exhaustive, list of techniques that can help meet
45
46 these requirements for computational analyses. Science philosopher Karl Popper
47
48 contended that, “[w]e do not take even our own observations quite seriously, or accept
49
50 them as scientific observations, until we have repeated and tested them” [2]. Indeed, in
51
52 many cases, the individuals who benefit most from computational reproducibility are those
53
54
55
56
57
58
59
60
61
62
63
64
65

1
2
3
4 who performed the original analysis, but reproducible and transparent practices can also
5
6 increase the level at which a scientist's work is accepted by other scientists [47, 103]. When
7
8 other scientists can reproduce an analysis and determine exactly how its conclusions were
9
10 drawn, they may be more apt to cite and build upon the work. In contrast, when others fail
11
12 to reproduce research findings, it can derail scientific progress and may lead to
13
14 embarrassment, accusations, and retractions.
15
16
17
18
19

20 We have described seven tools and techniques for facilitating computational
21
22 reproducibility. None of these approaches is sufficient for every scenario in isolation;
23
24 rather, scientists will often find value in combining approaches. For example, a researcher
25
26 who uses a literate programming notebook (that combines narratives with code) might
27
28 incorporate the notebook into a software container so that others can execute it without
29
30 needing to install specific software dependencies. The container might also include a
31
32 workflow management system to ease the process of integrating multiple tools and
33
34 incorporating best practices for the analysis. This container could be packaged within a
35
36 virtual machine or cloud-computing environment to ensure that it can be executed
37
38 consistently (see Figure 7). Binder [104] and Everware [128] are two services that allow
39
40 researchers to execute Jupyter notebooks within a Web browser, using a Docker container
41
42 to package the underlying software, and a cloud-computing environment to execute it.
43
44
45 Although still under active development, such services may be harbingers of the future for
46
47 computationally reproducible science.
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

1
2
3
4 The call for computational reproducibility relies on the premise that reproducible science
5 will bolster the efficiency of the overall scientific enterprise [105]. Although reproducible
6 practices may require additional time and effort, these practices provide ancillary benefits
7 that help offset those expenditures [47]. Primarily, scientists may experience increased
8 efficiency in their research [47]. For example, before and after a manuscript is submitted
9 for publication, it faces scrutiny from co-authors and peer reviewers who may suggest
10 alterations to the analysis. Having a complete record of all the analysis steps, and being
11 able to retrace those steps precisely, makes it faster and easier to implement the requested
12 alterations [47,106]. Reproducible practices can also improve the efficiency of team science
13 because colleagues can more easily communicate their research protocols and inspect each
14 other's work; one type of relationship where this is critical is that between academic
15 advisors and mentees [106]. Finally, when research protocols are shared transparently
16 with the broader community, scientific advancement increases because scientists can learn
17 more easily from each other's work and there is less duplication of effort [106].

18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40 Reproducible practices do not necessarily ensure that others can obtain identical results to
41 those obtained by the original scientists. Indeed, this objective may be infeasible for some
42 types of computational analysis, including those that use randomization procedures,
43 floating-point operations, or specialized computer hardware [91, 107]. In such cases, the
44 goal may shift to ensuring that others can obtain results that are semantically consistent
45 with the original findings [5,6]. In addition, in studies where vast computational resources
46 are needed to perform an analysis, or where data sets are distributed geographically [108–
47 110], full reproducibility may be infeasible. Alternatively, it may be infeasible to reallocate

1
2
3
4 computational resources for highly computationally intensive analyses [8]. In these cases,
5
6 researchers can provide relatively simple examples to demonstrate the methodology [8].
7
8
9 When legal restrictions prevent researchers from publicly sharing software or data, or
10
11 when software is available only via a Web interface, researchers should document the
12
13 analysis steps as well as possible and describe why such components cannot be shared
14
15 [25].
16
17
18
19

20 Computational reproducibility does not guarantee against analytical biases, or ensure that
21
22 software produces scientifically valid results [111]. As with any research, a poor study
23
24 design, confounding effects, or improper use of analytical software may plague even the
25
26 most reproducible analyses [111, 112]. On one hand, increased transparency puts
27
28 scientists at a greater risk that such problems will be exposed. On the other hand, scientists
29
30 who are fully transparent about their scientific approach may be more likely to avoid such
31
32 pitfalls, knowing that they will be more vulnerable to such criticisms. Either way, the
33
34 scientific community benefits.
35
36
37
38
39

40 Lastly, we emphasize that some reproducibility is better than none. Although some of the
41
42 practices described in this review require more technical expertise than others, they are
43
44 freely accessible to all scientists, and provide long-term benefits to the researcher and to
45
46 the scientific community. Indeed, as scientists act in good faith to perform these practices,
47
48 where feasible, the pace of scientific progress will surely increase.
49
50
51
52
53
54
55
56

57 **List of abbreviations**

58
59
60
61
62
63
64
65

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

DOI: Digital object identifier

HTML: Hypertext markup language

PDF: Portable document format

URL: Uniform resource locator

VCS: Version control system

Acknowledgements

SRP acknowledges startup funds provided by Brigham Young University. We thank research-community members and reviewers who provided valuable feedback on this manuscript.

Additional files

Additional file 1: This script is supporting material for Figure 1. It can be used to align DNA sequence data to a reference genome. First, it downloads the software and data files necessary for the analysis. Then, it extracts (“unzips”) these files, and aligns the data to a reference genome for Ebola virus. Finally, it converts, sorts, and indexes the aligned data.

Additional file 2: This Make file is supporting material for Figure 2. It performs the same function as Additional file 1, except that it is formatted for the Make utility. Accordingly, it is structured so that specific tasks must be executed before other tasks, in a hierarchical manner.

Additional file 3: This Jupyter notebook is supporting material for Figure 3. It contains code (in the Python programming language) for generating random numbers and plotting them in a graph.

Additional file 4: This document contains code (in the R language) for generating random numbers and plotting them on a graph. This document is in R Markdown format and can be compiled using knitr.

Competing Interests

Neither of the authors of this manuscript have any competing interests to declare.

Authors’ contributions

SRP wrote the manuscript and created figures. MBF created figures and helped to revise the manuscript.

1
2
3
4
5
6
7
8 **References**
9

- 10
11 1. Fisher RA. The Design of Experiments. New York: Hafner Press; 1935.
12
13
14
15 2. Popper KR. 2002. The logic of scientific discovery. London Routledge; 1959.
16
17
18 3. Peng RD. Reproducible research in computational science. Science. 2011;334:1226–7.
19
20
21
22 4. Russell JF. If a job is worth doing, it is worth doing twice. Nature. 2013;496:7.
23
24
25 5. Feynman RP, Leighton RB, Sands M. Six Easy Pieces: Essentials of Physics Explained by
26
27
28 Its Most Brilliant Teacher. Perseus Books; 1994. p. 34–5.
29
30
31 6. Murray-Rust P, Murray-Rust D. Reproducible Physical Science and the Declaratron. In:
32
33
34 Stodden VC, Leisch F, Peng RD, editors. Implementing Reproducible Research. CRC Press;
35
36
37 2014. p. 113.
38
39
40 7. Hey AJG, Tansley S, Tolle KM, Others. The fourth paradigm: data-intensive scientific
41
42
43 discovery. Microsoft Research Redmond, WA; 2009.
44
45
46 8. Millman KJ, Pérez F. Developing Open-Source Scientific Practice. Implementing
47
48
49 Reproducible Research. CRC Press; 2014;149.
50
51
52 9. Wilson G, Aruliah DA, Brown CT, Chue Hong NP, Davis M, Guy RT, et al. Best practices for
53
54
55 scientific computing. PLoS Biol. 2014;12:e1001745.
56
57
58 10. Software with impact. Nat. Methods. 2014;11:211.
59
60
61
62
63
64
65

- 1
2
3
4 11. Hong NC. We are the 92% [Internet]. Figshare; 2014. Available from:
5
6 <http://dx.doi.org/10.6084/M9.FIGSHARE.1243288>
7
8
9
- 10 12. Sacks J, Welch WJ, Mitchell TJ, Wynn HP. Design and Analysis of Computer Experiments.
11
12 Stat. Sci. Institute of Mathematical Statistics; 1989;4:409–23.
13
14
- 15 13. Garijo D, Kinnings S, Xie L, Xie L, Zhang Y, Bourne PE, et al. Quantifying reproducibility
16
17 in computational biology: the case of the tuberculosis drugome. PLoS One. 2013;8:e80278.
18
19
20
21
- 22 14. Error prone. Nature. 2012;487:406.
23
24
- 25 15. Vandewalle P, Barrenetxea G, Jovanovic I, Ridolfi A, Vetterli M. Experiences with
26
27 Reproducible Research in Various Facets of Signal Processing Research. 2007 IEEE
28
29 International Conference on Acoustics, Speech and Signal Processing - ICASSP '07. IEEE;
30
31 2007. p. IV – 1253 – IV – 1256.
32
33
34
35
- 36 16. Cassey P, Cassey P, Blackburn T, Blackburn T. Reproducibility and Repeatability in
37
38 Ecology. Bioscience. 2006;56:958–9.
39
40
41
- 42 17. Murphy JM, Sexton DMH, Barnett DN, Jones GS, Webb MJ, Collins M, et al. Quantification
43
44 of modelling uncertainties in a large ensemble of climate change simulations. Nature.
45
46 2004;430:768–72.
47
48
49
- 50 18. McCarthy DJ, Humburg P, Kanapin A, Rivas MA, Gaulton K, Cazier J-B, et al. Choice of
51
52 transcripts and software has a large effect on variant annotation. Genome Med. 2014;6:26.
53
54
55
- 56 19. Neuman JA, Isakov O, Shomron N. Analysis of insertion-deletion from deep-sequencing
57
58

- 1
2
3
4 data: Software evaluation for optimal detection. *Brief. Bioinform.* 2013;14:46–55.
5
6
7
8 20. Bradnam KR, Fass JN, Alexandrov A, Baranay P, Bechner M, Birol I, et al. Assemblathon
9
10 2: evaluating de novo methods of genome assembly in three vertebrate species.
11
12 *Gigascience.* 2013;2:10.
13
14
15
16 21. Bilal E, Dutkowski J, Guinney J, Jang IS, Logsdon BA, Pandey G, et al. Improving Breast
17
18 Cancer Survival Analysis through Competition-Based Multidimensional Modeling. *PLoS*
19
20 *Comput. Biol.* 2013;9:e1003047.
21
22
23
24 22. Gronenschild EHB, Habets P, Jacobs HIL, Mengelers R, Rozendaal N, van Os J, et al. The
25
26 effects of FreeSurfer version, workstation type, and Macintosh operating system version on
27
28 anatomical volume and cortical thickness measurements. *PLoS One.* 2012;7:e38234.
29
30
31
32
33 23. Moskvin OV, McIlwain S, Ong IM. CAMDA 2014: Making sense of RNA-Seq data: From
34
35 low-level processing to functional analysis. *Systems Biomedicine.* 2014;2:31–40.
36
37
38
39 24. Reducing our irreproducibility. *Nature.* 2013;496:398–398.
40
41
42
43 25. Michael CM, Nass SJ, Omenn GS, editors. *Evolution of Translational Omics: Lessons*
44
45 *Learned and the Path Forward.* Washington, D.C.: The National Academies Press; 2012.
46
47
48
49 26. Collins FS, Tabak L a. Policy: NIH plans to enhance reproducibility. *Nature.*
50
51 2014;505:612–3.
52
53
54 27. Chambers JM. S as a Programming Environment for Data Analysis and Graphics.
55
56 *Problem Solving Environments for Scientific Computing, Proc. 17th Symp. on the Interface*
57
58

1
2
3
4 of Stat. and Comp. North Holland; 1985. p. 211–4.
5
6

7
8 28. LeVeque RJ, Mitchell IM, Stodden V. Reproducible research for scientific computing:
9
10 Tools and strategies for changing the culture. *Computing in Science and Engineering*.
11
12 2012;14:13.
13
14

15
16 29. Stodden V, Guo P, Ma Z. Toward Reproducible Computational Research: An Empirical
17
18 Analysis of Data and Code Policy Adoption by Journals. *PLoS One*. 2013;8:2–9.
19
20

21
22 30. Morin A, Urban J, Adams PD, Foster I, Sali A, Baker D, et al. Research priorities. Shining
23
24 light into black boxes. *Science*. 2012;336:159–60.
25
26

27
28 31. Rebooting review. *Nat. Biotechnol*. 2015;33:319.
29
30

31
32 32. Ioannidis JP a., Allison DB, Ball C a., Coulibaly I, Cui X, Culhane AC, et al. Repeatability of
33
34 published microarray gene expression analyses. *Nat. Genet*. 2009;41:149–55.
35
36

37
38 33. Nekrutenko A, Taylor J. Next-generation sequencing data interpretation: enhancing
39
40 reproducibility and accessibility. *Nat. Rev. Genet*. Nature Publishing Group; 2012;13:667–
41
42 72.
43
44

45
46 34. Baggerly K a., Coombes KR. Deriving chemosensitivity from cell lines: Forensic
47
48 bioinformatics and reproducible research in high-throughput biology. *Ann. Appl. Stat*.
49
50 2009;3:1309–34.
51
52

53
54 35. Decullier E, Huot L, Samson G, Maisonneuve H. Visibility of retractions: a cross-sectional
55
56 one-year study. *BMC Res. Notes*. 2013;6:238.
57
58

- 1
2
3
4 36. Claerbout JF, Karrenbach M. Electronic Documents Give Reproducible Research a New
5
6 Meaning. Meeting of the Society of Exploration Geophysics. New Orleans, LA; 1992.
7
8
9
10 37. Stodden V, Miguez S. Best Practices for Computational Science: Software Infrastructure
11
12 and Environments for Reproducible and Extensible Research. Journal of Open Research
13
14 Software. 2014;2:21.
15
16
17
18 38. Ravel J, Wommack KE. All hail reproducibility in microbiome research. Microbiome.
19
20 2014;2:8.
21
22
23
24 39. Stodden V. 2014: What scientific idea is ready for retirement? [Internet].
25
26 <http://edge.org/response-detail/25340>. 2014. Available from: <http://edge.org/response->
27
28 [detail/25340](http://edge.org/response-)
29
30
31
32
33 40. Birney E, Hudson TJ, Green ED, Gunter C, Eddy S, Rogers J, et al. Prepublication data
34
35 sharing. Nature. 2009;461:168–70.
36
37
38
39 41. Hothorn T, Leisch F. Case studies in reproducibility. Brief. Bioinform. 2011;12:288–300.
40
41
42 42. Schofield PN, Bubela T, Weaver T, Portilla L, Brown SD, Hancock JM, et al. Post-
43
44 publication sharing of data and tools. Nature. 2009;461:171–3.
45
46
47
48 43. Piwowar H a., Day RS, Fridsma DB. Sharing detailed research data is associated with
49
50 increased citation rate. PLoS One. 2007;2.
51
52 <http://dx.doi.org/10.1371/journal.pone.0000308>
53
54
55
56 44. Johnson VE. Revised standards for statistical evidence. Proc. Natl. Acad. Sci. U. S. A.
57
58

1
2
3
4 2013;110:19313–7.
5
6

7
8 45. Halsey LG, Curran-everett D, Vowler SL, Drummond GB. The fickle P value generates
9
10 irreproducible results. Nat. Methods. 2015;12:179–85.
11
12

13
14 46. Wilson G. Software Carpentry: lessons learned. F1000Res. 2016;3:62.
15
16

17
18 47. Sandve GK, Nekrutenko A, Taylor J, Hovig E. Ten Simple Rules for Reproducible
19
20 Computational Research. PLoS Comput. Biol. 2013;9:1–4.
21
22

23
24 48. Köster J, Rahmann S. Snakemake--a scalable bioinformatics workflow engine.
25
26 Bioinformatics. 2012;28:2520–2.
27
28

29
30 49. Sadedin SP, Pope B, Oshlack A. Bpipe : A Tool for Running and Managing Bioinformatics
31
32 Pipelines. Bioinformatics. 2012;28:1525–6.
33
34

35
36 50. Tange O. GNU Parallel - The Command-Line Power Tool. ;login: The USENIX Magazine.
37
38 Frederiksberg, Denmark; 2011;36:42–7.
39
40

41
42 51. Albrecht M, Donnelly P, Bui P, Thain D. Makeflow: A portable abstraction for data
43
44 intensive computing on clusters, clouds, and grids. Proceedings of the 1st ACM SIGMOD
45
46 Workshop on Scalable Workflow Execution Engines and Technologies. 2012.
47
48

49
50 52. Knight S, Austin C, Crain C, Leblanc S, Roach A. Scons software construction tool
51
52 [Internet]. 2011. Available from: <http://www.scons.org>
53
54

55
56 53. Code share. Nature. 2014;514:536.
57
58
59
60
61
62
63
64
65

- 1
2
3
4 54. Blischak JD, Davenport ER, Wilson G. A Quick Introduction to Version Control with Git
5
6 and GitHub. PLoS Comput. Biol. 2016;12:e1004668.
7
8
9
- 10 55. Loeliger J, McCullough M. Version Control with Git: Powerful Tools and Techniques for
11
12 Collaborative Software Development. "O'Reilly Media, Inc."; 2012. p. 456.
13
14
15
- 16 56. Huber W, Carey VJ, Gentleman R, Anders S, Carlson M, Carvalho BS, et al. Orchestrating
17
18 high-throughput genomic analysis with Bioconductor. Nat. Methods. Nature Publishing
19
20 Group; 2015;12:115–21.
21
22
23
- 24 57. R Core Team. R: A Language and Environment for Statistical Computing [Internet].
25
26 Vienna, Austria: R Foundation for Statistical Computing; 2014. Available from:
27
28 <http://www.r-project.org>
29
30
31
32
- 33 58. Tóth G, Sokolov IV, Gombosi TI, Chesney DR, Clauer CR, De Zeeuw DL, et al. Space
34
35 Weather Modeling Framework: A new tool for the space science community. J. Geophys.
36
37 Res. 2005;110:A12226.
38
39
40
- 41 59. Tan E, Choi E, Thoutireddy P, Gurnis M, Aivazis M. GeoFramework: Coupling multiple
42
43 models of mantle convection within a computational framework. Geochem. Geophys.
44
45 Geosyst. [Internet]. 2006;7. Available from: <http://doi.wiley.com/10.1029/2005GC001155>
46
47
48
49
- 50 60. Heisen B, Boukhelef D, Esenov S, Hauf S, Kozlova I, Maia L, et al. Karabo: An Integrated
51
52 Software Framework Combining Control, Data Management, and Scientific Computing
53
54 Tasks. 14th International Conference on Accelerator & Large Experimental Physics Control
55
56 Systems, ICALEPCS2013. San Francisco, CA; 2013.
57
58
59

- 1
2
3
4 61. Schneider CA, Rasband WS, Eliceiri KW. NIH Image to ImageJ: 25 years of image
5
6 analysis. *Nat. Methods*. Nature Publishing Group; 2012;9:671–5.
7
8
9
10 62. Schindelin J, Arganda-Carreras I, Frise E, Kaynig V, Longair M, Pietzsch T, et al. Fiji: an
11
12 open-source platform for biological-image analysis. *Nat. Methods*. Nature Publishing Group,
13
14 a division of Macmillan Publishers Limited. All Rights Reserved.; 2012;9:676–82.
15
16
17
18 63. Biasini M, Schmidt T, Bienert S, Mariani V, Studer G, Haas J, et al. OpenStructure: an
19
20 integrated software framework for computational structural biology. *Acta Crystallogr. D*
21
22 *Biol. Crystallogr.* 2013;69:701–9.
23
24
25
26
27 64. Martin RC. *Clean code: a handbook of agile software craftsmanship*. Pearson Education;
28
29 2009.
30
31
32
33 65. Knuth DE. *Literate Programming*. *Comput. J.* 1984;27:97–111.
34
35
36
37 66. Pérez F, Granger BE. *IPython: a System for Interactive Scientific Computing*. *Computing*
38
39 *in Science and Engineering*. IEEE Computer Society; 2007;9:21–9.
40
41
42
43 67. Shen H. *Interactive notebooks: Sharing the code*. *Nature*. 2014;515:151–2.
44
45
46
47 68. Xie Y. *Dynamic Documents with R and knitr*. CRC Press; 2013. p. 216.
48
49
50 69. RStudio Team. *RStudio: Integrated Development for R* [Internet]. [cited 2015 Nov 20].
51
52 Available from: <http://www.rstudio.com>
53
54
55 70. Gross AM, Orosco RK, Shen JP, Egloff AM, Carter H, Hofree M, et al. Multi-tiered genomic
56
57 analysis of head and neck cancer ties TP53 mutation to 3p loss. *Nat. Genet.* *Nature*
58
59

1
2
3
4 Publishing Group; 2014;46:1–7.
5
6

7
8 71. Ding T, Schloss PD. Dynamics and associations of microbial community types across the
9 human body. *Nature*. Nature Publishing Group, a division of Macmillan Publishers Limited.
10 All Rights Reserved.; 2014;509:357–60.
11
12
13
14

15
16 72. Ram Y, Hadany L. The probability of improvement in Fisher’s geometric model: A
17 probabilistic approach. *Theor. Popul. Biol.* 2015;99:1–6.
18
19
20

21
22 73. Meadow JF, Altrichter AE, Kembel SW, Moriyama M, O’Connor TK, Womack AM, et al.
23 Bacterial communities on classroom surfaces vary with human contact. *Microbiome*.
24 2014;2:7.
25
26
27
28
29

30
31 74. White E. Programming for Biologists [Internet]. Available from:
32 <http://www.programmingforbiologists.org>
33
34

35
36 75. Peng RD. Coursera course: Computing for Data Analysis [Internet]. Available from:
37 <https://www.coursera.org/course/compdata>
38
39
40

41
42 76. Bioconductor - Courses and Conferences [Internet]. [cited 2015 Nov 20]. Available
43 from: <http://master.bioconductor.org/help/course-materials>
44
45
46
47

48
49 77. Gil Y, Deelman E, Ellisman M, Fahringer T, Fox G, Gannon D, et al. Examining the
50 challenges of scientific workflows. *Computer* . 2007;40:24–32.
51
52
53

54
55 78. Giardine B, Riemer C, Hardison RC, Burhans R, Elnitski L, Shah P, et al. Galaxy: a
56 platform for interactive large-scale genome analysis. *Genome Res.* 2005;15:1451–5.
57
58
59

1
2
3
4 79. Goecks J, Nekrutenko A, Taylor J. Galaxy: a comprehensive approach for supporting
5
6 accessible, reproducible, and transparent computational research in the life sciences.
7

8
9 Genome Biol. 2010;11:R86.
10

11
12 80. Afgan E, Baker D, Coraor N, Goto H, Paul IM, Makova KD, et al. Harnessing cloud
13
14 computing with Galaxy Cloud. Nat. Biotechnol. 2011;29:972–4.
15
16

17
18 81. Callahan SP, Freire J, Santos E, Scheidegger CE, Silva CT, Vo HT. VisTrails: Visualization
19
20 Meets Data Management. Proceedings of the 2006 ACM SIGMOD International Conference
21
22 on Management of Data. New York, NY, USA: ACM; 2006. p. 745–7.
23
24

25
26 82. Davidson SB, Freire J. Provenance and scientific workflows. Proceedings of the 2008
27
28 ACM SIGMOD international conference on Management of data - SIGMOD '08. 2008. p.
29
30 1345.
31
32

33
34 83. Altintas I, Berkley C, Jaeger E, Jones M, Ludascher B, Mock S. Kepler: an extensible
35
36 system for design and execution of scientific workflows. Proceedings. 16th International
37
38 Conference on Scientific and Statistical Database Management, 2004. IEEE; 2004. p. 423–4.
39
40
41

42
43 84. Goff SA, Vaughn M, McKay S, Lyons E, Stapleton AE, Gessler D, et al. The iPlant
44
45 Collaborative: Cyberinfrastructure for Plant Biology. Front. Plant Sci. Frontiers; 2011;2:34.
46
47
48

49
50 85. Reich M, Liefeld T, Gould J, Lerner J, Tamayo P, Mesirov JP. GenePattern 2.0. Nat. Genet.
51
52 2006;38:500–1.
53
54

55
56 86. Reich M, Liefeld J, Thorvaldsdottir H, Ocana M, Polk E, Jang D, et al. GenomeSpace: An
57
58

1
2
3
4 environment for frictionless bioinformatics. *Cancer Res.* 2012;72:3966–3966.
5
6

7
8 87. Wolstencroft K, Haines R, Fellows D, Williams A, Withers D, Owen S, et al. The Taverna
9
10 workflow suite: designing and executing workflows of Web Services on the desktop, web or
11
12 in the cloud. *Nucleic Acids Res.* 2013;41:557–61.
13
14

15
16 88. Rex DE, Ma JQ, Toga AW. The LONI Pipeline Processing Environment. *Neuroimage.*
17
18 2003;19:1033–48.
19
20

21
22 89. Lazarus R, Kaspi A, Ziemann M. Creating re-usable tools from scripts: The Galaxy Tool
23
24 Factory. *Bioinformatics.* 2012;28:3139–40.
25
26

27
28 90. Dudley JT, Butte AJ. In silico research in the era of cloud computing. *Nat. Biotechnol.*
29
30 Nature Publishing Group; 2010;28:1181–5.
31
32

33
34 91. Hurley DG, Budden DM, Crampin EJ. Virtual Reference Environments: a simple way to
35
36 make research reproducible. *Brief. Bioinform.* 2014;1–3.
37
38

39
40 92. Gent IP. The Recomputation Manifesto. arXiv [Internet]. 2013; Available from:
41
42 <http://arxiv.org/abs/1304.3674>
43
44

45
46 93. Howe B. Virtual Appliances, Cloud Computing, and Reproducible Research. *Comput. Sci.*
47
48 Eng. IEEE Computer Society; 2012;14:36–41.
49
50

51
52 94. Brown CT. Virtual machines considered harmful for reproducibility [Internet]. 2012.
53
54 Available from: <http://ivory.idyll.org/blog/vms-considered-harmful.html>
55
56

57
58 95. Piccolo SR. Building portable analytical environments to improve sustainability of
59
60

1
2
3
4 computational-analysis pipelines in the sciences [Internet]. 2014. Available from:
5
6 <http://dx.doi.org/10.6084/m9.figshare.1112571>
7
8
9

10 96. Krampis K, Booth T, Chapman B, Tiwari B, Bick M, Field D, et al. Cloud BioLinux: pre-
11 configured and on-demand bioinformatics computing for the genomics community. BMC
12 Bioinformatics. 2012;13:42.
13
14
15

16
17
18 97. Felter W, Ferreira A, Rajamony R, Rubio J. An Updated Performance Comparison of
19 Virtual Machines and Linux Containers [Internet]. IBM Research Division; 2014. Available
20 from:
21 [http://domino.research.ibm.com/library/CyberDig.nsf/papers/0929052195DD819C8525](http://domino.research.ibm.com/library/CyberDig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf)
22 [7D2300681E7B/\\$File/rc25482.pdf](http://domino.research.ibm.com/library/CyberDig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf)
23
24
25
26
27
28
29
30

31
32 98. Eglen SJ, Weeks M, Jessop M, Simonotto J, Jackson T, Sernagor E. A data repository and
33 analysis framework for spontaneous neural activity recordings in developing retina.
34 Gigascience. 2014;3:3.
35
36
37
38
39

40 99. Eglen SJ. Bivariate spatial point patterns in the retina: a reproducible review. Journal de
41 la Société Française de Statistique. 2016;157:33–48.
42
43
44

45
46 100. Bremges A, Maus I, Belmann P, Eikmeyer F, Winkler A, Albersmeier A, et al. Deeply
47 sequenced metagenome and metatranscriptome of a biogas-producing microbial
48 community from an agricultural production-scale biogas plant. Gigascience. 2015;4:33.
49
50
51
52

53
54 101. Belmann P, Dröge J, Bremges A, McHardy AC, Sczyrba A, Barton MD. Bioboxes:
55 standardised containers for interchangeable bioinformatics software. Gigascience.
56
57
58
59

1
2
3
4 2015;4:47.
5
6

7
8 102. Barton M. nucleotides · genome assembler benchmarking [Internet]. [cited 2015 Nov
9
10 20]. Available from: <http://nucleotid.es>
11
12

13
14 103. Hones MJ. Reproducibility as a Methodological Imperative in Experimental Research.
15
16 PSA: Proceedings of the Biennial Meeting of the Philosophy of Science Association.
17
18 Philosophy of Science Association; 1990. p. 585–99.
19
20

21
22 104. Rosenberg DM, Horn CC. Neurophysiological analytics for all! Free open-source
23
24 software tools for documenting, analyzing, visualizing, and sharing using electronic
25
26 notebooks. J. Neurophysiol. American Physiological Society; 2016;jn.00137.2016.
27
28

29
30 105. Crick T. “ Share and Enjoy ”: Publishing Useful and Usable Scientific Models. Available
31
32 from: <http://arxiv.org/abs/1409.0367v2>
33
34

35
36 106. Donoho DL. An invitation to reproducible computational research. Biostatistics.
37
38 2010;11:385–8.
39
40

41
42 107. Goldberg D. What Every Computer Scientist Should Know About Floating-point
43
44 Arithmetic. ACM Comput. Surv. New York, NY, USA: ACM; 1991;23:5–48.
45
46

47
48 108. Shirts M, Pande VS. COMPUTING: Screen Savers of the World Unite! Science.
49
50 2000;290:1903–4.
51
52

53
54 109. Bird I. Computing for the Large Hadron Collider. Annu. Rev. Nucl. Part. Sci.
55
56 2011;61:99–118.
57
58

- 1
2
3
4 110. Anderson DP. BOINC: A System for Public Resource Computing and Storage.
5
6 Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04).
7
8 2004.
9
10
11
12 111. Ransohoff DF. Bias as a threat to the validity of cancer molecular-marker research.
13
14 Nat. Rev. Cancer. 2005;5:142–9.
15
16
17
18 112. Bild AH, Chang JT, Johnson WE, Piccolo SR. A field guide to genomics research. PLoS
19
20 Biol. 2014;12:e1001744.
21
22
23
24 113. GNU Make [Internet]. 2016. Available from <https://www.gnu.org/software/make>
25
26
27
28 114. Make for Windows [Internet]. 2016. Available from
29
30 <http://gnuwin32.sourceforge.net/packages/make.htm>
31
32
33
34 115. Ivy, the agile dependency manager [Internet]. 2016. Available from
35
36 <http://ant.apache.org/ivy>
37
38
39
40 116. Puppet [Internet]. 2016. Available from <https://puppetlabs.com>
41
42
43
44 117. aRchive: Enabling reproducibility of Bioconductor package versions (for Galaxy)
45
46 [Internet]. 2016. Available from <http://bioarchive.github.io>
47
48
49
50 118. GenePattern: A platform for reproducible bioinformatics [Internet]. 2016. Available
51
52 from <http://www.broadinstitute.org/cancer/software/genepattern>
53
54
55
56 119. LONI Pipeline Processing Environment [Internet]. 2016. Available from
57
58 <http://www.loni.usc.edu/Software/Pipeline>
59
60
61
62
63
64
65

- 1
2
3
4 120. CloudBioLinux: configure virtual (or real) machines with tools for biological analyses
5
6
7 [Internet]. 2016. Available from <https://github.com/chapmanb/cloudbiolinux>
8
9
- 10 121. Vortex [Internet]. 2016. Available from <https://github.com/websecurify/node-vortex>
11
12
- 13 122. Amazon Web Services [Internet]. 2016. Available from <http://aws.amazon.com>
14
15
- 16 123. Google Cloud Platform [Internet]. 2016. Available from
17
18
19 <https://cloud.google.com/compute>
20
21
- 22 124. Microsoft Azure [Internet]. 2016. Available from <https://azure.microsoft.com>
23
24
25
- 26 125. lmctfy - Let Me Contain That For You [Internet]. 2016. Available from
27
28
29 <https://github.com/google/lmctfy>
30
31
- 32 126. Warden [Internet]. 2016. Available from
33
34
35 <http://docs.cloudfoundry.org/concepts/architecture/warden.html>
36
37
- 38 127. everware [Internet]. 2016. Available from <https://github.com/everware/everware>
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

Figure legends

Figure 1. Example of a command line script. This script can be used to align DNA sequence data to a reference genome. First, it downloads the software and data files necessary for the analysis. Then, it extracts (“unzips”) these files, and aligns the data to a reference genome for Ebola virus. Finally, it converts, sorts, and indexes the aligned data. See Additional file 1 for an executable version of this script.

Figure 2. Example of a Make file. This file performs the same function as the command line script shown in Figure 1, except that it is formatted for the Make utility. Accordingly, it is structured so that specific tasks must be executed before other tasks, in a hierarchical manner. See Additional file 2 for an executable version of this file.

Figure 3. Example of a Jupyter notebook. This example contains code (in the Python programming language) for generating random numbers and plotting them in a graph within a Jupyter notebook. Importantly, the code and output object (graph) are contained within the same document. See Additional file 3 for an executable version of the notebook.

Figure 4. Example of a document created using knitr. This example contains code (in the R language) for generating random numbers and plotting them on a graph. The knitr tool was used to generate the document, which combines the code and the output object (figure). See Additional file 4 for an executable version of this document.

Figure 5. Architecture of virtual machines. Virtual machines encapsulate analytical software and dependencies within a “guest” operating system, which may be different to

1
2
3
4 the main (“host”) operating system. A virtual machine executes in the context of
5
6 virtualization software, which runs alongside other software installed on the computer.
7
8

9
10 **Figure 6. Architecture of software containers.** Software containers encapsulate
11
12 analytical software and dependencies. In contrast to virtual machines, containers execute
13
14 within the context of the computer’s main operating system.
15
16
17

18 **Figure 7. Example of a Docker container for genomics research.** This container would
19
20 enable researchers to preprocess various types of molecular data, using tools from
21
22 Bioconductor and Galaxy, and to analyze the resulting data within a Jupyter notebook. Each
23
24 box within the container represents a distinct Docker image. These images are layered such
25
26 that some images depend on others (for example, the Bioconductor image depends on R).
27
28 At its base, the container includes operating system libraries, which may not be present (or
29
30 may be configured differently) on the computer’s main operating system.
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

```
#!/bin/bash

# Download software, reference genome, and FASTQ files
wget http://downloads.sourceforge.net/project/bio-bwa/bwakit/bwakit-0.7.12_x64-
linux.tar.bz2
wget -O refGenomeFiles.zip https://ndownloader.figshare.com/files/4841809
wget -O FASTQ.zip https://ndownloader.figshare.com/articles/3114454/versions/1

# Extract software and reference genome files
tar -jxvf bwakit-0.7.12_x64-linux.tar.bz2
unzip refGenomeFiles.zip
unzip FASTQ.zip

# Align FASTQ data to reference genome and save to SAM file
bwa.kit/bwa mem KJ660346.fa SRR1972917_1.fastq SRR1972917_1.fastq >
SRR1972917_aligned.sam

# Convert SAM file to BAM file
bwa.kit/samtools view -bS SRR1972917_aligned.sam > SRR1972917_aligned.bam

# Sort BAM file
bwa.kit/samtools sort SRR1972917_aligned.bam SRR1972917_aligned_sorted

# Index BAM file
bwa.kit/samtools index SRR1972917_aligned_sorted.bam
```



```
all: SRR1972917_aligned_sorted.bam

SRR1972917_aligned_sorted.bam: SRR1972917_aligned.sam
# Convert SAM file to BAM file
bwa.kit/samtools view -bS SRR1972917_aligned.sam > SRR1972917_aligned.bam

# Sort BAM file
bwa.kit/samtools sort SRR1972917_aligned.bam SRR1972917_aligned_sorted

# Index BAM file
bwa.kit/samtools index SRR1972917_aligned_sorted.bam

SRR1972917_aligned.sam: bwa.kit/bwa KJ660346.fa SRR1972917_1.fastq
# Align FASTQ data to reference genome and save to SAM file
bwa.kit/bwa mem KJ660346.fa SRR1972917_1.fastq SRR1972917_1.fastq > SRR1972917_aligned.sam

bwa.kit/bwa:
# Download and unpack BWA software
wget http://downloads.sourceforge.net/project/bio-bwa/bwakit/bwakit-0.7.12_x64-linux.tar.bz2
tar -jxvf bwakit-0.7.12_x64-linux.tar.bz2

KJ660346.fa:
# Download and unzip reference genome
wget -O refGenomeFiles.zip https://ndownloader.figshare.com/files/4841809
unzip refGenomeFiles.zip

SRR1972917_1.fastq:
# Download and unzip FASTQ files
wget -O FASTQ.zip https://ndownloader.figshare.com/articles/3114454/versions/1
unzip FASTQ.zip

clean:
rm -rfv KJ660346.fa* FASTQ.zip refGenomeFiles.zip bwa.kit* bwakit-0.7.12_x64-linux.tar.bz2 SRR1972917*
```

Load Python libraries

```
In [1]: get_ipython().magic('matplotlib inline')

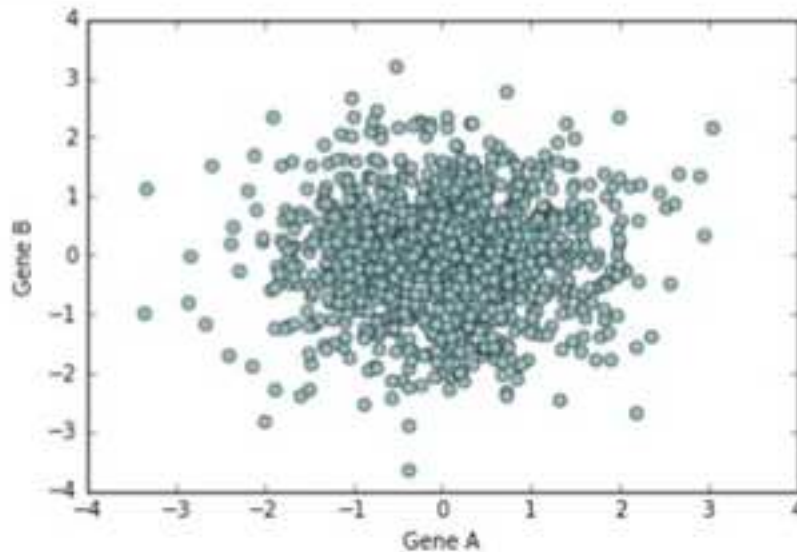
import matplotlib.pyplot as plt
from numpy.random import normal
```

Generate random numbers simulating gene-expression values

```
In [2]: geneA = normal(size=1000)
geneB = normal(size=1000)
```

Plot the values

```
In [4]: plt.plot(geneA, geneB, "o", color="#99C1C2")
plt.xlabel("Gene A")
plt.ylabel("Gene B")
plt.show()
```



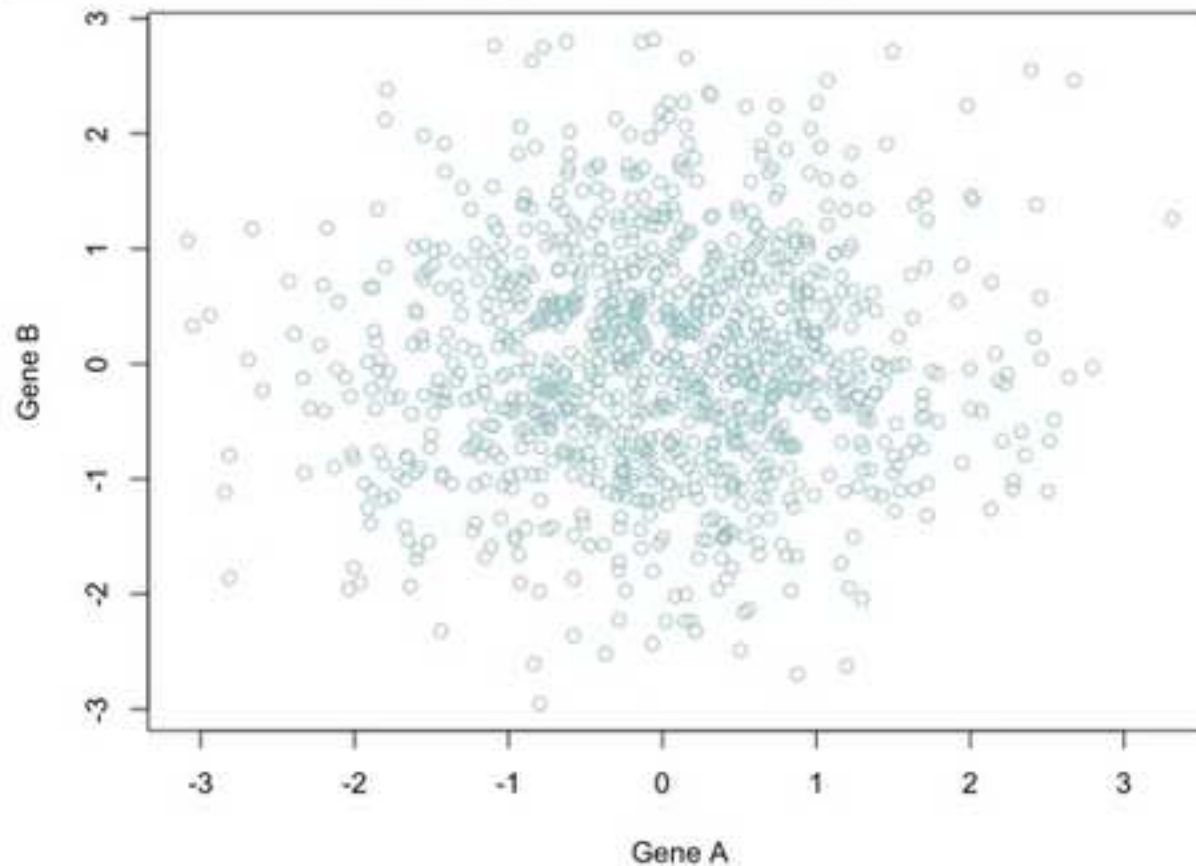
R Markdown Example

Generate random numbers simulating gene-expression values

```
geneA <- rnorm(1000)  
geneB <- rnorm(1000)
```

Plot the numbers as a histogram

```
# Set the margins so there won't be too much white space  
par(mar=c(4.1, 4.1, 0.1, 0.1))  
  
plot(geneA, geneB, col="#99C1C2", xlab="Gene A", ylab="Gene B", main="")
```



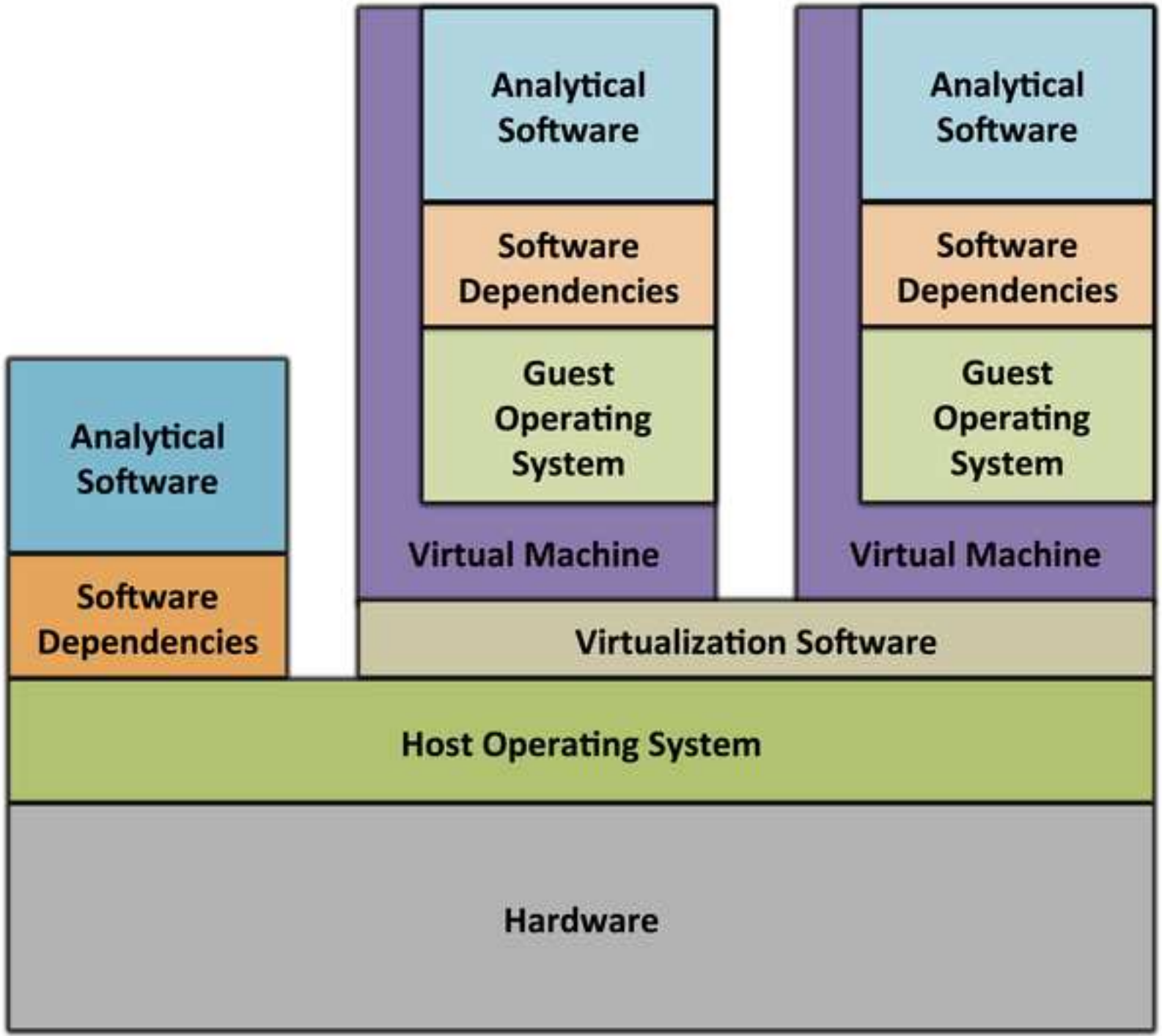
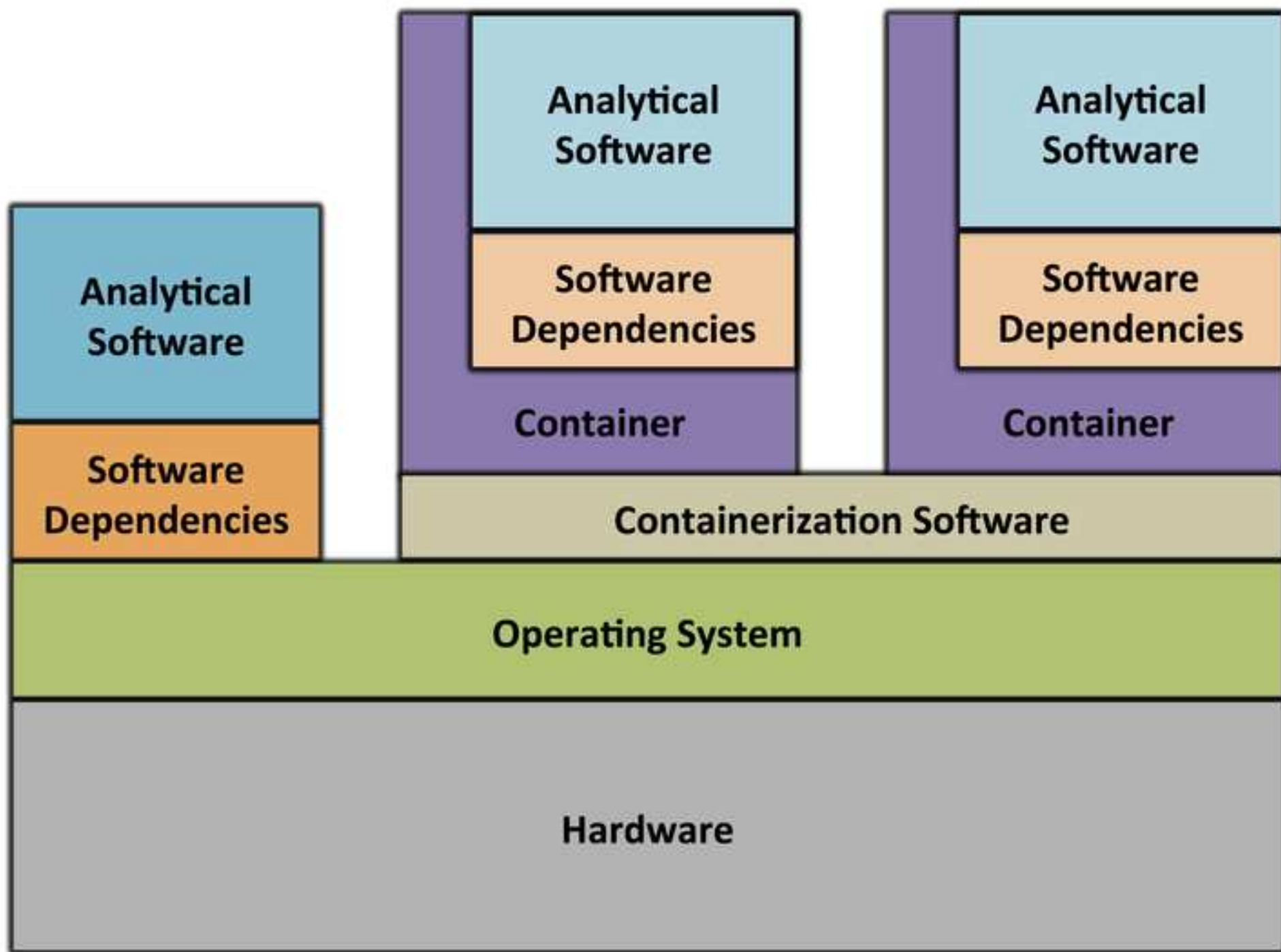
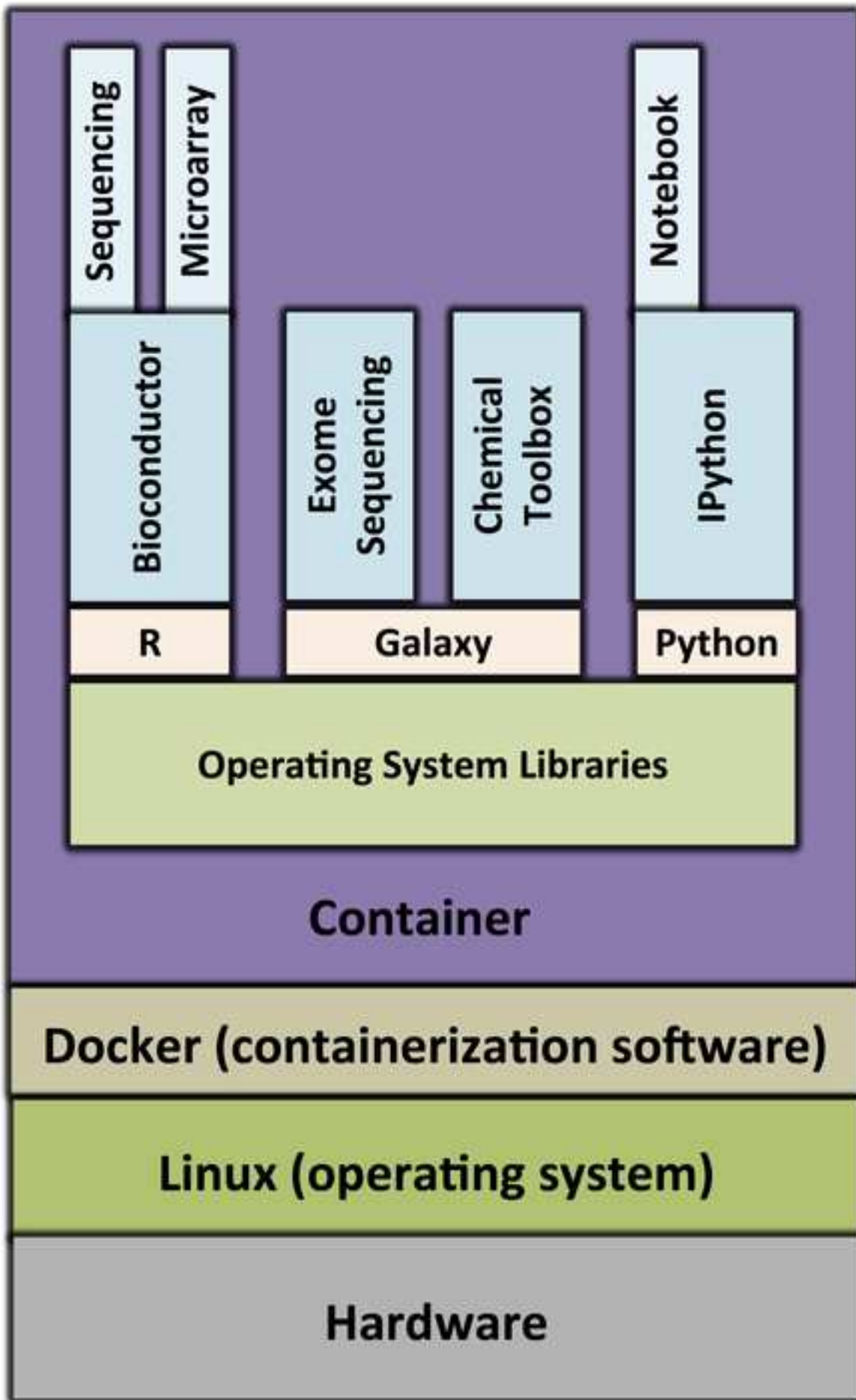
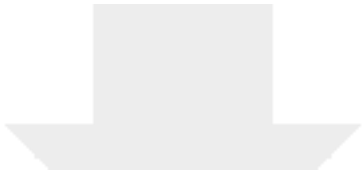


Figure 6







Click here to access/download
Supplementary Material
Script_Example.sh





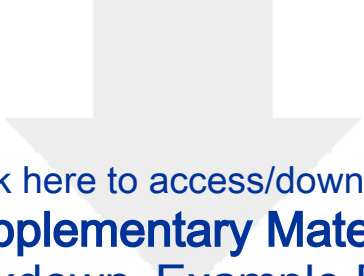
Click here to access/download
Supplementary Material
Makefile_Example





Click here to access/download
Supplementary Material
Jupyter_Example.ipynb





Click here to access/download
Supplementary Material
Markdown_Example.Rmd

