Paths to computational fluency for natural resource educators, researchers, and managers

Richard A. Erickson1 | Jessica L. Burnett2 | Mark T. Wiltermuth2 | Edward A. Bulliner3 | Leslie Hsu2

1Upper Midwest Environmental Sciences Center, U.S. Geological Survey, La Crosse, Wisconsin, USA
2Science Analytics and Synthesis, Denver Federal Center, U.S. Geological Survey, Denver, Colorado, USA
3Columbia Environmental Research Center, U.S. Geological Survey, Columbia, Missouri, USA

Correspondence
Richard A. Erickson, Upper Midwest Environmental Sciences Center, U.S. Geological Survey, 2630 Fanta Reed Rd, La Crosse, WI 54601, USA.
Email: rerickson@usgs.gov

Funding information
US Geological Survey Community for Data Integration, Grant/Award Number: US Geological Survey Community for Data Integration

Abstract
Natural resource management and supporting research teams need computational fluency in the data and model-rich 21st century. Computational fluency describes the ability of practitioners and scientists to conduct research and represent natural systems within the computer’s environment. Advancement in information synthesis for natural resource management requires more sophisticated computational approaches, as well as reproducible, reusable, extensible, and transferable methods. Despite this importance, many new and current natural resource practitioners lack computational fluency and no common set of recommended resources and practices exist for learning these skills. Broadly, attaining computational fluency entails moving beyond the simple use of computers to applying sound computational principles and methods and including computational experts (such as computer scientists) on research teams. Our path for computational fluency includes using open-source tools when possible; reproducible data management, statistics, and modeling; understanding and applying the benefits of basic computer programming to carry out more complex procedures; tracking code with version control; working in...
controlled computer environments; and using advanced computing resources.

**Considerations for Resource Managers**

- Natural resource management increasingly uses computer-generated results to inform and guide decision making. Open science requires that these data and software be reproducible. In turn, open science promotes computational fluency among natural resource managers, researchers, and educators.
- Based upon our experiences and perspectives working to support natural resource professionals, many natural resource managers would like to attain computationally fluency, yet lack formal training.
- We provide a path to computational fluency that emphasizes: using open-source tools when possible; reproducible data management, statistics, and modeling; understanding and applying the basic computer programming philosophy to carry out both simple and more complex procedures; tracking code with version control; working in controlled computer environments; and using advanced computing resources.
- Natural resource managers, educators, and researchers can use their computationally fluency skills to promote open science in their own work.

**KEYWORDS**
natural resource education, open science, reproducible results

## 1 | INTRODUCTION

Natural resource management and supporting research are increasingly computationally intense fields. In part, this is because the environment and ecosystems are inherently complex (Valle & Berdanier, 2012). Capturing this complexity requires computer programming to implement advanced statistical methods and mathematical models (Bolker, 2008; Ellison & Dennis, 2010). Furthermore, new technologies generate large data sets, oftentimes gigabytes or larger, which require advanced computing resources and tools. Examples of uses of these data include the use of light detection and ranging (LIDAR) for tree inventory in forestry (e.g., Popescu, 2007), the application of high-throughput genetic sequencing to monitor and survey for wildlife and fish management (e.g., Bohmann et al., 2014), the use of satellite-generated land-cover data to track and monitor land use (e.g., Loveland & Dwyer, 2012), and the collection of high-resolution telemetry data of fish movements as part of new management tool
These massive influxes of data and subsequent transformation with analyses and models create challenges. People who use data to make decisions and conduct research can be “drowning in information” while still struggling to make management decisions (Text box 1).

Whereas computer literacy is the ability to use a computer, computational fluency (borrowing from Ellison & Dennis’s, 2010 definition of statistical fluency) is an understanding of computational concepts and the ability to apply them appropriately within an individual’s role (see definition in Text box 2). Computational fluency can help with synthesis because it provides not only the tools to work with large and messy data sets, but also the knowledge necessary to evaluate complex models and data sets (see Dobson et al., 2020; Salk, 2020 for perspectives on messy data and tidying messing data for conservation). For example, a fish biologist might use a hydrology-based model to decide when to implement management actions for an invasive fish species (e.g., applying a model, such as Embke et al., 2019). The fish biologist may not be an expert on hydrology or modeling but will be evaluating what model best

**TEXT BOX 1. E. O. Wilson on information**

“We are drowning in information, while starving for wisdom. The world henceforth will be run by synthesizers, people able to put together the right information at the right time, think critically about it, and make important choices wisely.” E. O. Wilson in *Consilience: The Unity of Knowledge* (1998)

**TEXT BOX 2. Contrasting computational fluency with computer literacy**

*Computer literacy* has several formal definitions (e.g., Simonson et al., 1987). Simonson et al. (1987) presents a formal definition, describing computer literacy as “an understanding of computer characteristics, capabilities, and applications, as well as an ability to implement this knowledge in the skillful, productive use of computer applications suitable to individual roles in society.” Less formally, Nataraj (2014) highlights computer literacy as the ability to “use computer technology effectively in education and work environment” and for most computer users, this is the ability to use point-and-click graphical user interfaces (GUIs).

*Computational fluency* requires computer literacy as well as an understanding of the underlying theory of scientific information synthesis within a computer environment combined with an ability to apply new concepts that are suitable to an individual’s role. Thus, the concepts are important and not the specific technologies implementing the concepts.

As an example, computer literacy would be the ability to use a scripting language, such as R or Python to process, analyze, and visualize data. Computational fluency would be not only developing and using the computer script, but also understanding the data structure and analytical methods as well as ensuring the data workflow, analytical methods, and results are reproducible and trustworthy.
addresses their management questions. In these cases, the natural resource manager making decisions needs to be computationally fluent to the extent they are comfortable working with the team using the model as part of their decision-making process. Additionally, decisions made using these data and models require transparency and reproducibility. Lastly, computational fluency will help natural resource managers better understand modeling “rules of thumb” used to simplify and communicate about complex natural systems.

Increasing data availability and computational complexity coincides with an emerging need for reproducible and open science, especially in fields like natural resource management where science informs decision making and policies (Ellison & Dennis, 2010; Powers & Hampton, 2019; Valle & Berdanier, 2012). Many organizations currently encourage open science. Groups, such as GoFAIR (https://www.go-fair.org/, accessed October 29, 2020), promote open science through the Findability, Accessibility, Interoperability, and Reuse (FAIR Guiding Principles; Wilkinson et al., 2016). Scientific societies whose members include natural resource managers also recognize the need for reproducible and open science (Table 1). However, not all “open standards” are open science. For example, the Conservation Measures Partnership created Open Standards for the Practice of Conservation (https://cmp-openstandards.org/, accessed October 29, 2020). These standards are openly published methods for the conservation decision-making process, but their case studies are not open science nor reproducible given the limited materials shared publicly as part of their case studies.

Working with large and complex data sets, using advanced statistical and mathematical models, and reproducible and open research all require computational fluency. The diversity of data, information, and interpretation often require expertise in diverse fields; hence contribution and computational fluency might depend upon a person’s role within a team. For example, a research bioinformatician may require different skills than a regulator, but both could benefit from understanding how to assess the reproducibility of code, the former from the aspect of application to new data and the latter from being able to evaluate if the code is reproducible and trustable.

In our experience working closely with natural resource managers and the researchers supporting these managers, we have observed that many scientists and managers want to increase their computational fluency. Likewise, Barraquand et al. (2014) conducted a survey of early career ecologists and found many wanted more quantitative training. Historically, natural resource curricula provided minimal exposure to quantitative fluency other than data management and basic scientific programing (Barraquand et al., 2014; Ellison & Dennis, 2010; Valle & Berdanier, 2012). Outside of the field of natural resource management groups, broader groups have promoted increasing quantitative and computer literacy in education and practice. As one example, the U.S. National Academies report on undergraduate biology education (BIO2010, 2003) presents a vision and path for training future research biologists. Likewise, computer science and mathematical groups also promote educational standards (e.g., Rüde et al., 2018), yet this study is not always read or accessible to many resource managers or students. As another example for undergraduate education, the U.S. National Academies published a report promoting a vision for undergraduate data science education (Committee on Envisioning the Data Science Discipline, 2021). Together these reports provide thorough and detailed recommendations for how to increase computational fluency, yet also highlight the gap between natural resource managers and quantitative educators simply by the omission of this target audience. Additionally, none of the author and contributor affiliations included state, tribal, or federal natural resource management agencies (e.g., state Departments of Natural Resources, state Natural History Surveys, National Oceanic and Atmospheric
### Table 1: Example of organizations (including governments) supporting or requiring open science and link to webpage documentation action

| Organization | Organization action or group and citation or webpage (and access date) |
|--------------|------------------------------------------------------------------------|
| ESA          | Open Science Section (https://www.esa.org/openscience/, October 29, 2020) |
| SETAC        | Declaration on Open Science (https://www.setac.org/page/open-science-declaration, October 29, 2020) |
| AGU          | Enabling FAIR Data Project (http://www.copdess.org/enabling-fair-data-project, October 29, 2020) |
| WCRI         | Singapore Statement on Research Integrity (https://wcrif.org/guidance/singapore-statement, October 29, 2020) |
| ASA          | Reproducible Research Recommendations for Funding Agencies (https://www.amstat.org/asa/files/pdfs/POL-ReproducibleResearchRecommendations.pdf, October 29, 2020) |
| ALLEA        | Code of conduct (https://allea.org/code-of-conduct/, October 29, 2020) |
| OECD         | Open data in their standards (http://www.oecd.org/gov/digital-government/open-government-data.htm, October 29, 2020) |
| US           | Open Government Data Act of 2018 (H.R.1770) |
| US           | Geospatial Data Act of 2018 (H.R. 302, P.L. 115–254) |
| USA          | 2016 Memoranda on Federal Source Code require open code (https://www.whitehouse.gov/sites/whitehouse.gov/files/omb/memoranda/2016/m_16_21.pdf, October 29, 2020) |
| USGS         | Fundamental of Scientific Practices (U.S. Geological Survey, 2011) |
| USGS         | Open Data (https://www.usgs.gov/products/data-and-tools/data-management October 29, 2020) |
| USGS         | Open Software (https://www.usgs.gov/products/software/software-management/, October 29, 2020) |
| USEPA        | Open Science, (Environmental Protection Agency, 2021) Federal Registry Number 2020-29179 |
| US NSF       | Public access (https://www.nsf.gov/news/special_reports/public_access/index.jsp, October 29, 2020) |
| UA           | Australian Code for the Responsible Conduct of Research (National Health and Medical Research Council Australia, 2018). |
| OCPAC        | Scientific Integrity, draft policy (http://www.ic.gc.ca/eic/site/052.nsf/eng/00010.html, October 29, 2020) |
| EC           | Open Science (https://ec.europa.eu/research/openscience/, October 29, 2020) |

Abbreviations: AGU, American Geophysical Union; ALLEA, All European Academies; ASA, American Statistical Association; EC, European Commission; ESA, Ecological Society of America; FAIR, Findability, Accessibility, Interoperability, and Reuse; OCPAC, Office of Chief Policy Advisor-Canada; OECD, Organization for Economic Cooperation and Development; SETAC, Society for Environmental Toxicology and Chemistry; UA, Universities Australia; US NSF, U.S. National Science Foundation; US, U.S. Federal law; USEPA, U.S. Environmental Protection Agency; USGS, U.S. Geological Survey; WCRI, World Conferences on Research Integrity.

Recent activities, such as a December 2020 Workshop at the National Institute for Mathematical and Biological Synthesis (http://www.nimbios.org/workshops/WS_quantedu,
Open science principles, such as FAIR, can also help with transparency and scientific integrity. Although open science will not prevent scientific fraud and misconduct (especially if done for personal gain), transparent workflows and open science allow other people to examine and recreate the data, analysis, and underlying assumptions. In turn, this can help others detect fraud or errors. For example, the U.S. Environmental Protection Agency’s open science policies discuss scientific integrity (Environmental Protection Agency, 2021). Outside of natural resource management, a high-profile example of open science catching scientific misconduct occurred when Keith Baggerly and others attempted to recreate high-throughput biology results and uncovered sloppy science and fraudulent science being used to guide clinical cancer trials (Baggerly & Berry, 2011). Within natural resource management, groups, such as Conservation Measures Partnership, have created Open Standards for the Practice of Conservation (https://cmp-openstandards.org/, accessed October 29, 2020) to enable more transparent decision making.

Despite the recognized importance of computational fluency, no common definition exists for natural resource management and closely related fields, such as applied ecology and the environmental sciences (see Text box 2 for our working definition of computational fluency). This lack of a definition makes obtaining computational fluency challenging for current practitioners or students because they may not know why concepts are important or what concepts to learn. Others have provided suggestions and discussed the skills necessary for computational and quantitative fluency in ecology (Barraquand et al., 2014; Valle & Berdanier, 2012) as well as broader science (Wilson et al., 2017). We go a step further and provide an overview of these concepts as well as a path to fluency for both students and practitioners of applied ecology and natural resource management. We have based these recommendations on our experiences working with students, scientists, decision makers, and managers within government agencies as well as the skills and concepts we train new and current practitioners to use.

Our recommendations for achieving computational fluency approximately follow the American Statistical Association’s (ASA’s) 2017 guidelines on reproducible results (ASA, 2017). We also share the ASA’s recognition that computational fluency and reproducible results are not all-or-nothing. Instead, there will be different levels of understanding and application depending upon one’s role and specific project goals. Furthermore, different levels of fluency will be present on teams of resource managers and researchers. Consider a field biologist collecting data, a graduate student analyzing those data, a primary investigator leading the study, and a resource manager identifying interpretive management information gaps to be addressed by the team and ultimately applying that knowledge to action. Each has separate yet integrated roles within the team that require varying degrees of computational fluency. The field biologist collecting data may enter observations into a computer system but may benefit
from understanding the context of the data to be able to identify potentially confounding field observations. The graduate student analyzing the data will be creating reproducible analysis as a primary product but likely also benefits from understanding the larger context of the data collection to ensure data quality/applicability of the data set. The primary investigator would ensure (and ideally verify) reproducibility of the graduate student’s analysis as well as describe the frequency and accuracy limits of the data collection. The resource manager would be able to identify the details of the requested information that underpins the investigation, in addition to obtaining a general understanding of the data analyses to make the work understandable by stakeholders and the general public. Additionally, this example illustrates another need for the science of team science in natural resources (Hall et al., 2018).

We recognize this diversity and provide different levels of guidance for the concepts described below. We present these concepts by describing the importance and utility of open-source software when possible. Next, we provide an overview of reproducible science, including data management, statistics, and modeling. Then, we describe the benefits of basic computer programming to complete more repetitive or complex procedures. Next, we discuss tracking code development with version control. Then, we describe working in controlled computer environments. Lastly, we describe how the previous concepts can be applied to using advanced computing resources (e.g., high-performance computing, HPC; high-throughput computing, HTC). For the broader context of quantitative education, we refer the reader to more detailed reports (e.g., BIO2010, 2003; Committee on Envisioning the Data Science Discipline, 2021; Rüde et al., 2018).

2  |  CONCEPTS

2.1  |  Open source

2.1.1  |  Overview

We start by describing the benefits of open-source software. Although no official definition exists, the Open Source Initiative provides key principles of open-source software (Text box 3). Open-source software can be beneficial for many reasons, including access, transparency, and often continuous development. Additionally, some governments either require or encourage open-source software development for work they fund (e.g., United States, https://www.cio.gov/2016/08/11/peoples-code.html, accessed October 29, 2020; United Kingdom, https://www.gov.uk/guidance/be-open-and-use-open-source, accessed October 29, 2020; European Union, https://ec.europa.eu/info/departments/informatics/open-source-software-strategy_en, accessed October 29, 2020; also see Table 1). We describe two benefits of open-source software for natural resource practitioners.

First, if methods are reproducible then you and others can reuse and build upon them. For example, Ellison (2010) described how he could not recreate ecological estimations because previous authors either did not report software used or used commercial programs that were no longer available. More recent studies have found a lack of reproducibility to be a broader trend in wildlife management journals (e.g., Archmiller et al., 2020) and ecological journals (Culina et al., 2020). Open-source software helps prevent this lack of reproducibility because the software is free to redistribute or share for any purpose. Additionally, most major open-source software projects make a commitment to maintain access to old versions of their software (archiving). The free-of-cost-to-use benefit
removes monetary software costs as a barrier to entry for economically disadvantaged groups or natural resource agencies without funding for proprietary software. However, the computational resources (e.g., random access memory [RAM], central processor unit [CPU] power) to run the open-source software may still limit access.

Second, open-source software provides access to the source code. This allows advanced users access to see how code works, identify bugs, expand upon or alter the code, or simply understand how software works. This second strength of open-source software highlights that many individuals and organizations support and contribute to open-source software for practical and altruistic reasons: to add, enhance, and support features they need or want or to give back to the broader community. Hence, based upon these two characteristics, using open-source tools for computational activities may be beneficial.

2.1.2 | Example implementations

Open-source software exists for most research activities, including operating systems (e.g., Linux), statistical software (e.g., R and Python), word processing (e.g., LibreOffice and Apache), and geospatial tools (e.g., QGIS).

2.1.3 | Where can data creators and users start?

Whether driven by a need (e.g., you cannot afford commercial software, commercial software does not exist for your need) or desire to choose open-source software, many options exist. One of the easiest ways to learn new technologies is to begin by using open-source software. For example, if you are learning how to apply statistical models to natural resource problems, an open-source language, such as R or Python, may be a good starting point. Searching the

---

**TEXT BOX 3.** The 10 principles of open-source software from the Open Source Initiative (https://opensource.org/osd, accessed October 29, 2020, published under CC by 4.0 license) include

1. Free redistribution
2. Source code
3. Derived works
4. Integrity of the author's source code
5. No discrimination against persons or groups
6. No discrimination against fields of endeavor
7. Distribution of license
8. License must not be specific to a product
9. License must not restrict other software
10. License must be technology-neutral
internet for your application (e.g., “statistics programming”) and using the keyword “open source” will often identify open-source software options.

2.1.4 | How can educators help?

Educators can teach using open-source software to inform about the existence and utility of open-source tools. For example, R might be used for a statistics course, Python might be used for a numerical methods course, or QGIS for geospatial information system (GIS) courses.

2.2 | Reproducible data, code, statistics, and modeling

2.2.1 | Overview

Computational fluency includes understanding and applying principles of reproducible science that, at the most basic level, allows other people to recreate results. This includes the data, the metadata (“data about data,” such as the data collection protocols, precision, and accuracy), the code used to process the data, and the underlying models. We have identified three principles to follow. First, reproducible methods consist of sharing and publishing data with exceptions made for sensitive data (e.g., endangered species locations, culturally sensitive locations for fields such archeology, medical records with personal information, ASA, 2017) regardless of the other elements of reproducibility. When restrictions on data access and use are well justified, the other principles of reproducibility still apply. Second, reproducible methods entail documenting what methods and software versions were used to process, analyze, and interpret data. Computationally fluent scientists use computer code scripts for documenting and exercising these steps. Computer scripts are preferable to writing out the steps and methods because documenting every detail becomes mundane and difficult (e.g., how data were transformed in a spreadsheet, what specific statistical model options were used, which specific settings were used for a plot, etc.). Instead, the ASA (2017) recommends “end-to-end scripting of research, including data processing and cleaning, statistical analyses, visualizations, and report and/or manuscript generation, with the full workflow made available to others.” Computational methods are using computer programming scripts, written in platforms, such as R or Python, that execute procedures for data processing (e.g., transforming variables and formatting checks), data analysis and modeling (e.g., statistical tests and simulations), and data presentation (e.g., generating plots and data summaries). Third, reproducible methods include sharing the computational methods used to manipulate data and generate results. In addition to documenting methods, using these repositories provides a place for acknowledging contributions and recommending citation form.

In our observations, openly sharing data and code at or before time of publication might be the most important of these three principles because many fields of research have not yet prioritized sharing of data and code (as noted by ASA, 2017). Positive incentives to encourage the sharing of data and code include earning research badges (e.g., the Center for Open Science; https://www.cos.io/initiatives/badges, accessed October 29, 2020); whose members include this journal’s publisher Wiley (https://authorservices.wiley.com/open-research/open-recognition-and-reward/open-research-badges.html, accessed October 29, 2020). Additionally, funding agencies and publishers (as described in Section 1) are increasingly requiring researchers to
share their data (e.g., the U.S. Geological Survey, USGS) currently requires authors to share data with only narrow exceptions for security, privacy, confidentiality, or other constraints (https://www.usgs.gov/about/organization/science-support/survey-manual/5026-fundamental-science-practices-scientific-data, accessed March 16, 2021). We have observed that many researchers are already using open-source scripting languages for their research even if the researchers are not sharing their code (e.g., Erickson & Rattner, 2020 found a plurality of authors in the 2019 volume of Environmental Toxicology and Chemistry used the R program for scripting their research). Some of these explicit efforts and requirements are needed to realize the greater long-term benefit to science.

2.2.2 | Example implementations

Open data require sharing data and code to a public repository. Currently, Trusted Digital Repositories exist as a long-term option (https://www.oclc.org/content/dam/research/activities/trustedrep/repositories.pdf, accessed October 29, 2020). Some organizations, such as USGS, have their own data repositories (e.g., sciencebase.gov, accessed October 29, 2020), and nongovernmental groups exist to broadly support data sharing, such as Dryad (https://datadryad.org/stash, accessed October 29, 2020). Both of these repositories contain many examples of open data. Adding and using data from open repositories is a key aspect of producing reproducible science.

Data management and analysis through code is an approach that uses scripting languages within computer files to record and execute computational procedures. For example, a script file might read in data, transform or format the data, run a statistical model, and then save the statistical output as new files. Multiple benefits exist for “end-to-end” scripting. First, documented procedures are repeatable with minimal effort. This might be done when more data become available or a model requires updates. Second, this approach alleviates data versioning challenges because original data are preserved. The ideal of this approach is that all subsequent data sets beyond the original data source are generated by code, and code connects all steps between original data and final outputs. Third, other people can see what was done with the data and modeling efforts.

Currently, available open-source scripting languages include R, Python, or Julia. New scripting languages are emerging, and the most commonly used languages are changing, but the core concepts of scripting are constant across technologies.

2.2.3 | Where can data creators and users start?

Open data principles include sharing of data as a foundational practice. Key concepts include putting data in a long-term stable online location (preferably a Trusted Digital Repository) and describing those data with metadata. Further, minting data and information with digital object identifiers (DOIs) provides a stable reference for accessing the data and is a method for crediting data creators in publication.

To start scripting data processing and analysis methods, it may be beneficial to use a programming language your friends, colleagues, or fellow students use. They will be able to help you more than any other resource when initially stymied (Carey & Papin, 2018). Although there are syntax differences among languages, programming languages are all used to develop
procedures. To begin programming, we often find it helpful to organize and outline our code before we start programming. As part of this process, we organize the steps within the procedures and begin writing the easy steps first. Some people give up GUIs abruptly and stop using all GUls, although gradually transitioning may be helpful. For example, someone who currently edits their data in a spreadsheet and uses point-and-click programs for statistics and plotting could start off using a scripting language for statistics. Then, on a later project, they could incorporate plotting and transforming their data with the scripting language into their workflow. When learning to program your statistical methods, it may be helpful to use the point-and-click program to compare results.

For people at organizations (e.g., universities and resource management agencies) without formal guidance or training, organizations, such as GoFAIR (https://www.go-fair.org/, accessed October 29, 2020) and Data Carpentry (https://datacarpentry.org/, accessed October 29, 2020), provide resources and training for reproducible data management, analysis, and sharing.

2.2.4 | Key points for decision makers

When using data and model results, ensure the data and scripts are readily available and include documentation so that others may use them. Even if decision makers do not have the technical knowledge to rerun or reuse the script and data, evaluating the availability, accessibility, code review process (e.g., peer review), and understandability of data and code can be useful to determine if the quality of the information reported is sufficient for use in decision support. We propose these three criteria are essential for high-quality information production and sharing. To break down those three criteria:

1. **availability** is the degree to which all data and code are made available to the public,
2. **accessibility** is the degree of ease to which those items are obtained and used, and
3. **understandability** is the degree to which a person with or without technical knowledge can follow the basic approach and meaning by looking at a data set or reading through a code script or documentation.

Metadata of a data set and annotations within a code script should fulfill these criteria for most applications.

2.2.5 | How can educators help?

Educators can use open-source languages in their classes. For example, we know of some professors who compare different statistics programs for some classes and have students compare and contrast results (e.g., one group of students uses R, another Python, and a third SAS). Educators who are also researchers can demonstrate open science with their own research and that of students’ conducting research whom they mentor. Educators can use the volumes of available data on various topics within their classes to not only reproduce science results but also begin exploring new uses and interpretation of those data with their students. Such an approach is a great demonstration of how the open science community operates. Good locations for discovering data include government sites, such as data.gov, https://data.gov.uk/, and https://data.europa.eu/euodp/en/data/, nongovernment sites, such as DataOne.org, and
commercial search engines, such as https://www.google.com/publicdata/ (all URLs accessed October 29, 2020). Locations we have used for discovering relevant disciplinary and open science course materials include public repositories, such as github.com, gitlab.com, The Carpentries (https://carpentries.org/), and Open Science Foundation (https://osf.io accessed May 19, 2021).

2.3 Basics of programming

2.3.1 Overview

Moving beyond simple scripting, often people find themselves doing more programming and repetitive tasks. For this, familiarity with some basic concepts from programming and computer science helps. These basic concepts can be applied to any programming language. First, writing a script using a ‘clean’ and consistent style helps both you and others read and reuse your code (Martin, 2009). For example, snake_case or CamelCase make code easier to read, much like using spaces, or header styles in text. Second, using the do not repeat yourself (or DRY) principle helps improve coding efficiency and usability. For example, when conducting a statistical analysis, as part of Waller, Bartsch, and Erickson (2020), Erickson had to use the same analyses and create similar figures for multiple treatments and species. Rather than creating one long script file, Erickson used the DRY principle putting the analysis code into a series of for loops to rerun the analysis Waller, Bartsch, Lord et al. (2020). Taking this a step further, code can repeatedly be used if defined as functions. Multiple functions can then be placed into packages, which help other people use the code with new data or in different contexts. Using functions and for loops reduces the amount of code and makes it easier to update and fix code (e.g., rather than fixing 1 typo that was copy-and-pasted five times, there would only be 1 typo to fix). Third, testing can be used to make sure functions work correctly (Hunt & Thomas, 2011). Fourth, as people become better at programing, they can find methods to automate more parts of their coding and code checking. For example, unit testing lets people automatically check code (Hunt & Thomas, 2011) and linting is a way to automate checking code style, errors, and bugs. Fifth, using terminals, such as the BASH shell or Windows Power Shell, rather than GUIs, such as Windows Explorer, allows people to more efficiently interact with their operating system and give people the option of scripting their operating system. Although we first found terminals to be overwhelming, they have helped us move other advanced computational methods.

2.3.2 Example implementations

Code style guides exist for different organizations (e.g., Google has their own style guides, https://developers.google.com/style/, accessed October 29, 2020, for many languages, and NASA has a C++ style guide, https://ntrs.nasa.gov/citations/20080039927, accessed October 29, 2020). If your organization does not have a style guide, simply being internally consistent will help make code readable for both yourself and others. The basic programming concepts, such as for loops and functions, we described appear in most modern programming languages. Also, automation tools for coding exist for most modern computer languages. For example, both Python and R have linting packages (e.g., pylint and flake8 for Python; lintr for R) and
testing packages (e.g., unittest in Python; testthat in R). Last, all modern computer systems come with command prompt terminal emulators. It may be beneficial for users to start with the BASH shell because it comes with both Linux and macOS and is easily installed on Windows operating systems.

### 2.3.3 Key points for decision makers

If using somebody else’s code to support your decision, ask them about their coding style and quality control checks. Also, examine the source code. Even if you cannot read the programming language, the “clean” code should still have documentation and an intuitive structure. Additionally, ask people about their coding practices. For example, do they write tests for their code? Testing might not be necessary for one-off, simple scripts (e.g., a script running a t-test), but the programmer’s response will provide insight into their coding approach. Or, to put the question more directly, ask a person writing code “why should we trust your code?” Ideally, a user will be able to understand code based only upon documentation without reaching out to the programmer.

### 2.3.4 Where can data creators and users start?

For people seeking to learn about these programming concepts, groups, such as Software Carpentry, offer online tutorials (https://software-carpentry.org/, accessed October 29, 2020) and introductory programming books, such as Hunt and Thomas (2011), cover these topics. Language-specific books and resources also often cover these topics, such as Wickham and Bryan’s (2021) online R package book (https://r-pkgs.org/, accessed March 23, 2021) or the Python documentation (https://docs.python.org/3/tutorial/, accessed October 29, 2020). Organizations, such as The Carpentries (parent organization of Software Carpentry), also offer discipline-specific training, such as Ecology, Genomic, and Geospatial focused training (e.g., https://carpentries.org/workshops-curricula/, accessed October 29, 2020).

### 2.3.5 How can educators help?

Educators can help students by encouraging good coding practice when teaching how to program for coursework and capstone projects (e.g., thesis and dissertations). Educators can also use automated testing to reduce their grading time for coding-related projects, such as applied statistics. Educators may also include introductory computer science courses as part of natural resource curriculums to formally teach these concepts.

### 2.4 Version control

#### 2.4.1 Overview

Version control keeps track of when code was changed in a repository (Tichy, 1985). For example, if a person creates a script file today, but then adds new lines of code tomorrow,
version control would track changes. Likewise, version control can also be used to track changes when people collaborate. A single user might only use version control on their local computer with a single repository, but most users will also use remote repositories. Remote repositories allow for code to be backed up remotely and shared across users. The importance of version control is that the software keeps track of when and how code was changed. This allows people to more easily fix bugs or incorporate new features. Making use of version control as a standard practice (or habit) is a good metric of computational fluency.

2.4.2 Example implementations

Currently, multiple open-source version control software programs exist, including Git, Mercurial, Subversion (SVN), and Concurrent Versions System (CVS). Of these, Git is the most popular with 72% of all repositories online, SVN has 23%, and Mercurial and CVS each have about 1% (https://www.openhub.net/repositories/compare, accessed September 28, 2020). When using version control, people often use remote repositories to share their code and collaborate. For example, the USGS has its own Git repository, code.usgs.gov (accessed October 29, 2020). Commercial companies also host free Git repositories. Major commercial repositories include GitHub, GitLab, and Bitbucket. Each of these repositories (and others that we likely missed) allows people to create their own public repositories and have both free and paid options.

2.4.3 Where can data creators and users start?

The Git documentation (https://git-scm.com/, accessed October 29, 2020) contains tutorials on version control, as does Software Carpentry (https://swcarpentry.github.io/git-novice/, accessed October 30, 2020). As one begins to use version control, they may only want to use it to share final versions of their software and code. However, the path to computational fluency includes using version control throughout the project to keep track of changes to the data and code, in addition to sharing it. Also, public Git repositories (such as the three commercial vendors previously listed) allow one to create portfolios demonstrating their skills. This can be helpful for people looking for employment, especially those early in their careers (Robinson & Nolis, 2020).

2.4.4 Key points for decision makers

When using code for decision making, ask the code author if they used version control to track changes both pre- and postrelease. If not, ask them why not. If so, ask if they have made their code repository publicly available. When looking at the repository, check whether the repository file structure is disorganized or is organized and easy to follow. For example, is there a “readme” file describing the repository or other metadata about the code? Are there 10 data files with names like “data_final_2.csv?” Is the original, unmanipulated data (or “raw data”) provided? A lack of repository structure may also indicate a disorganized and irreproducible workflow.
2.4.5 | How can educators help?

Educators can help by teaching students how to use version control. Educators can also help by encouraging students to use public repositories as part of student portfolios and using these programs for grading group assignments (e.g., educators can see who added what content). Besides serving as a learning opportunity and teaching method, these portfolios also help students showcase their skills for future employers (Robinson & Nolis, 2020).

2.5 | Controlled computer environments

2.5.1 | Overview

A computer's environment is the collection of software installed on a given specific hardware (e.g., using R version 4.0.2 on Windows 10.0.18363 on a 2020 Surface Pro 7). More formally, Schmidt (2013) provided the definition that “the computing environment involves the collection of computer machinery, data storage devices, work stations, software applications, and networks that support the processing and exchange of electronic information demanded by the software solution.” For most computer users, using the default installation of software works well (e.g., installing the current version of R or Python and using the programs to analyze data or run models). However, even the slightest change in a version or configuration of software can produce different results. Hence, a controlled computer environment intentionally specifies the versions of software used, and the software “containers” are the tool for controlling the environment. For example, rather than using the locally installed version of R on a computer, one might choose to use R version 4.0.2 with ggplot2 version 3.0 in a container running Ubuntu 18.04 for the operating system. We have observed three situations where controlled computer environments help us.

First, controlled computer environments specify what versions of software are used and may alleviate problems caused by changes in the software between versions. For example, we have had papers go through the peer review process and be required to revise figures. However, a software update broke our original code and we had to update our plotting code. As a more extreme example, some USGS colleagues have had hydrology models that produce slightly different numerical results due to numerical rounding differences between operating systems (e.g., the computer’s representation of numbers varies at the last decimal place, usually the 8th or 16th digit; Michael N. Fienen, USGS, written commun., March, 23 2021). Due to possible version errors and lack of reproducibility, the ASA (2017) recommendations include using controlled computer environments.

Second, we also use controlled environments because they allow us to work in locked-down workplace computer environments. For example, we can currently update and install the newest versions of R in a Docker Container (https://www.rocker-project.org/, accessed October 30, 2020), but cannot upgrade R in our Windows computer environment. Controlled computer environments can also help with “dependency hell” where different applications require different versions of programs (Merkel, 2014). For example, R package A might require version 1.3 of R package Z, but R package B might require version 2.1 of package Z. Using this example, switching between projects that require package A or B is difficult unless using a controlled environment.

Third, we find controlled computer environments helpful when we change computers on a regular basis or work with collaborators that have difficulty installing software programs.
For example, Person A might develop a model on their macOS laptop while working with Person B, who is running Windows 10 on a Desktop. The model might then be used to run large simulations on a server running Ubuntu Linux but rerun on a second server using RedHat Linux. In this example, four different computers would have to have the same programs installed and configured. Controlled computer environments would help the team make sure all four computers access the same versions of software installed.

### 2.5.2 Example implementations

Multiple tools can be used for controlled environments, including containers and Conda environments. Conda environments allow people to specify specific versions of Python, R, and required packages for both programs (https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html, accessed October 29, 2020). Containers run a lightweight, mini-operating system and use specified software versions (Merkel, 2014). Current container options include Docker, and for scientific computing Singularity (https://sylabs.io/docs/, accessed October 29, 2020) and other options exist. For R, the Rocker project (https://www.rocker-project.org/, accessed March 23, 2021) can be helpful because it allows us to use RStudio with Docker. Additionally, other tools exist, such as Packrat for R (https://rstudio.github.io/packrat/, accessed March 23, 2021) or the use of virtual machines.

### 2.5.3 Where can data creators and users start?

In our experience, people start to use controlled computer environments when they have the three problems we previously described: (1) a need for highly reproducible models with specific software versions; (2) a need to install software, but no ability to locally install the software other than in containers; (3) a need to install otherwise difficult-to-install software on multiple machines. For Python users, Conda is an easy-to-use program for controlling package version. For R users, the Rocker project may be an easy way to start using Docker with R. Scripting can be used to create controlled environments, which makes the controlled environments more reproducible (e.g., using environment.yml files for Conda and Dockerfiles for Docker).

### 2.5.4 Key points for decision makers

Ask people developing software if they use a controlled computer environment. Also, ask if the statistical or model results change if different versions of software or operating systems are used, and if so, how. Details on the controlled computer environment, software versions, and operating systems should be documented in the repository.

### 2.5.5 How can educators help?

Educators can introduce the concept of controlled computer environments in advanced statistics and modeling courses. They might also use controlled computer environments to help students install difficult-to-install software for their classes.
2.6 | Advanced computing resources

2.6.1 | Overview

A scientific computer user may find that a personal computer (e.g., desktop, workstation, and laptop) no longer meets their computational requirements. When this occurs, the person requires advanced computing. We use the term “advanced computing” because research universities use similar terms (e.g., the University of Wisconsin-Madison hosts an “Advanced Computing Initiative,” https://it.wisc.edu/it-projects/projects-initiatives/advanced-computing-initiative/, accessed October 29, 2020) and the USGS has an Advanced Computing Cooperative Community of Practice (https://my.usgs.gov/confluence/display/cdi/Advanced+Computing+Cooperative, accessed October 29, 2020). Advanced computing can include HPC, HTC, and cloud computing. HPC works best with large computer tasks that are often broken down into tightly linked smaller tasks. In contrast, HTC works best with large computer tasks that can be broken down into many smaller tasks and run independently (Erickson et al., 2018). Historically, HPC required large, onsite computers, such as the super-computers, found on major research universities or national laboratories. In contrast, HTC could either scavenge unused resources, such as idle desktops computers at night or use dedicated resources, such as servers. Cloud computing uses offsite resources and can use HTC- or HPC-based software and computing approaches. Cloud computing can also allow for “cloud bursting” to expand computer resources for HTC or HPC (e.g., Requa et al., 2016).

Using advanced computing resources usually requires concepts described in previous sections of this paper. For example, most advanced computing resources require the use of scripting languages to run programs, the use of terminals to start programs and move data on and off the resources, and customized computer environments on the advanced computing resources. In fact, the basic computer concepts used for advanced computing are similar across HPC, HTC, and cloud computing. Advanced computing platforms are sufficiently complex that it may be helpful to foster collaborations with experts in the use of these platforms if they are necessary to the problem being addressed.

2.6.2 | Example implementations

HPC is usually limited to larger organizations due to the high overhead, although some smaller universities may have advanced computing resources. In contrast, HTC was created to operate on smaller budgets by using existing, idle resources and open-source software to run the HTC systems. Additionally, collaborative groups, such as the Open Science Grid (https://opensciencegrid.org/, accessed October 29, 2020), offer free HTC resources for large-scale computing projects. Cloud-based computing usually requires paying for services, although some cloud-based providers have provided research grants for academic researchers.

2.6.3 | Where can data creators and users start?

If you are fortunate, your organization has “facilitators,” such as those found at the Center for High Throughput Computing at the University of Wisconsin-Madison (http://chtc.cs.wisc.edu/, accessed October 29, 2020) who can guide you through your computing problems. If not, first
identify your computing problem, or reasons you need the advanced computing resources. For example, use code profiling to identify slow portions of bottlenecks in your code. For R users, Wickham (2019) describes how to do this in Chapter 23 of the online Advanced R Programming book (https://adv-r.hadley.nz/perf-measure.html, accessed October 29, 2020). For Python users, the Python documentation describes how to profile code (https://docs.python.org/3/library/profile.html, accessed October 29, 2020). For example, are you limited by processor speed or memory? Can your job be broken down into many smaller jobs? Can you optimize your code and possibly even avoid the need for HPC or HTC (e.g., switching the slow parts of your code from slower languages like Python or R to faster languages, such as C++)? Broadly, where is your code slowest and where is the best use of your resources (both time and money) to speed it up either through improved programming or using advanced computational resources? Then, start investigating resources, solutions, and potential collaborators to make incremental improvements to your code or to use HPC or HTC.

2.6.4 | Key points for decision makers

When funding computer-based resources, check to see if the project’s computer requirements seem reasonable given their described methods. For example, a bioinformatics study proposing to sequence the DNA of 1000s of species will require more computer resources than a basic laptop. Conversely, research requiring a simple linear regression on a small data set will rarely require computing resources beyond a modern laptop.

2.6.5 | How can educators help?

If applicable, help the students you mentor access advanced computing resources at your home institution and encourage basic training in using these resources. Also, educators may help students work through the optimization profiling process described above when resources are not available. Educators can also develop exercises that require optimization (or improved efficiency) to demonstrate the principles of advanced computing even on a single computer. Additionally, agencies can help educators access advanced computing resources for their students, especially when the education examples tie into the agencies’ missions.

3 | CONCLUSION

Understanding natural resources requires computational fluency among both researchers and management professionals. We provided a definition of computational fluency and contrasted it with computational literacy (Text box 2). Computational fluency requires moving beyond simply using a computer to understand how computers facilitate open and reproducible research including topics, such as computer environments, software carpentry, and coding. We also provided a path for computational fluency for researchers, educators, and management professionals. This path includes using open-source tools when possible; applying reproducible data management, statistics, and modeling; understanding and applying the basic computer programming philosophy to carry out
both simple and more complex procedures; tracking code with version control; working in controlled computer environments; and using advanced computing resources.

ACKNOWLEDGMENTS
We thank the USGS Community for Data Integration for funding this project. Any use of trade, firm, or product names is for descriptive purposes only and does not imply endorsement by the U.S. Government. We thank Erica L. Mize for her feedback and perspective on previous drafts of this manuscript. We thank anonymous reviewers, Barnett A. Rattner, Janet M. Carter, Jay E. Diffendorfer, Mark P. Gaikowski, and Jon J. Amberg for providing feedback on earlier versions of this manuscript.

AUTHOR CONTRIBUTIONS
Richard A. Erickson: Conceptualization (lead); funding acquisition (lead); project administration (lead); writing original draft (lead); writing review and editing (lead). Jessica L. Burnett: Conceptualization (equal); writing original draft (equal); writing review and editing (equal). Mark T. Wiltermuth: Conceptualization (equal); writing original draft (equal); writing review and editing (equal). Edward A. Bulliner: Funding acquisition (equal); writing review and editing (supporting). Leslie Hsu: Project administration (equal); writing review and editing (supporting).

DATA AVAILABILITY STATEMENT
Data sharing not applicable—no new data generated.

ORCID
Richard A. Erickson https://orcid.org/0000-0003-4649-482X
Jessica L. Burnett https://orcid.org/0000-0002-0896-5099
Mark T. Wiltermuth https://orcid.org/0000-0002-8871-2816
Edward A. Bulliner https://orcid.org/0000-0002-2774-9295
Leslie Hsu https://orcid.org/0000-0002-5353-807X

REFERENCES
American Statistical Association (ASA). (2017). Recommendations to funding agencies for supporting reproducible research. Retrieved from https://www.amstat.org/asa/files/pdfs/POL-ReproducibleResearchRecommendations.pdf. Accessed October 29, 2020.

Archmiller, A. A., Johnson, A. D., Nolan, J., Edwards, M., Elliott, L. H., Ferguson, J. M., Iannarilli, F., Vélez, J., Vitense, K., Johnson, D. H., & Fieberg, J. (2020). Computational reproducibility in the Wildlife Society’s flagship journals. The Journal of Wildlife Management, 84(5), 1012–1017. https://doi.org/10.1002/jwmg.21855

Baggerly, K. A., & Berry, D. A. (2011). Reproducible research. Amstat News. January 2011:16–17.

Barraquand, F., Ezard, T. H. G., Jorgensen, P. S., Zimmerman, N., Chamberlain, S., Salguero-Gómez, R., Curran, T. J., & Poisot, T. (2014). Lack of quantitative training among early-career ecologists: A survey of the problem and potential solutions. PeerJ, 2, e285. https://doi.org/10.7717/peerj.285

BIO2010. (2003). Transforming undergraduate education for future research biologists. National Academies Press. Retrieved from http://www.nap.edu/catalog/10497. Accessed January 26, 2021.

Bohmann, K., Evans, A., Gilbert, M. T. P., Carvalho, G. R., Creer, S., Knapp, M., Yu, D. W., & de Bruyn, M. (2014). Environmental DNA for wildlife biology and biodiversity monitoring. Trends in Ecology and Evolution (Personal Edition), 29(6), 358–367. https://doi.org/10.1016/j.tree.2014.04.003

Bolker, B. M. (2008). Ecological models and data in R. Princeton University Press.

Carey, M. A., & Papin, J. A. (2018). Ten simple rules for biologists learning to program. PLOS Computational Biology, 14, e1005871141. https://doi.org/10.1371/journal.pcbi.1005871
Committee on Envisioning the Data Science Discipline: The Undergraduate Perspective, Computer Science and Telecommunications Board, Board on Mathematical Sciences and Analytics, Committee on Applied and Theoretical Statistics, Division on Engineering and Physical Sciences, Board on Science Education, Division of Behavioral and Social Sciences and Education, National Academies of Sciences, Engineering, and Medicine. (2018). Envisioning the data science discipline: The undergraduate perspective: Interim report. National Academies Press. Retrieved from https://www.nap.edu/catalog/24886. Accessed January 26, 2021.

Culina, A., van den Berg, I., Evans, S., & Sánchez-Tójar, A. (2020). Low availability of code in ecology: A call for urgent action. PLOS Biology, 18(7), e3000763. https://doi.org/10.1371/journal.pbio.3000763

Cupp, A. R., Erickson, R. A., Fredricks, K. T., Swers, N. M., Hatton, T. W., & Amberg, J. J. (2017). Responses of invasive silver and bighead carp to a carbon dioxide barrier in outdoor ponds. Canadian Journal of Fisheries and Aquatic Sciences, 74, 297–305. https://doi.org/10.1139/cjfas-2015-0472

Dobson, A. D. M., Milner-Gulland, E. J., Aebischer, N. J., Beale, C. M., Brozovic, R., Coals, P., Critchlow, R., Dancer, A., Greve, M., Hinsley, A., Ibbett, H., Johnston, A., Kuiper, T., Le Comber, S., Mahood, S. P., Moore, J. F., Nilsen, E. B., Pocock, M. J. O., Quinn, A., ... Keane, A. (2020). Making messy data work for conservation. One Earth, 2(5), 455–465. https://doi.org/10.1016/j.oneear.2020.04.012

Ellison, A. M. (2010). Repeatability and transparency in ecological research. Ecology, 91(9), 2536–2539.

Ellison, A. M., & Dennis, B. (2010). Paths to statistical fluency for ecologists. Frontiers in Ecology and the Environment, 8, 362–370. https://doi.org/10.1890/080209

Embke, H. S., Kocovsky, P. M., Garcia, T., Mayer, C. M., & Qian, S. S. (2019). Modeling framework to estimate spawning and hatching locations of pelagically spawned eggs. Canadian Journal of Fisheries and Aquatic Sciences, 76(4), 597–607. https://doi.org/10.1139/cjfas-2018-0047

Environmental Protection Agency. (2021). Strengthening transparency in pivotal science underlying significant regulatory actions and influential scientific information. Retrieved from https://beta.regulations.gov/document/EPA-HQ-OA-2018-0259-22422

Erickson, R. A., Fienen, M. N., McCalla, S. G., Weiser, E. L., Bower, M. L., Knudson, J. M., & Thain, G. (2018). Wrangling distributed computing for high-throughput environmental science: An introduction to HTCondor. PLOS Computational Biology, 14, e1006468. https://doi.org/10.1371/journal.pcbi.1006468

Erickson, R. A., & Rattner, B. A. (2020). Moving beyond p < 0.05 in ecotoxicology: A guide for practitioners. Environmental Toxicology and Chemistry, 39(9), 1657–1669. https://doi.org/10.1002/etc.4800

Hall, K. L., Vogel, A. L., Huang, G. C., Serrano, K. J., Rice, E. L., Tsakraklides, S. P., & Fiore, S. M. (2018). The science of team science: A review of the empirical evidence and research gaps on collaboration in science. American Psychologist, 73(4), 532–548. https://doi.org/10.1037/amp0000319

Hsu, L., & Colasuonno, L. (2019). Community for data integration 2018 annual report. (US Geological Survey Open-File Report 2019–1123). https://doi.org/10.3133/ofr20191123

Hunt, A., & Thomas, D. (2011). The pragmatic programmer: From journeyman to master (Vol. 26). Addison-Wesley.

Loveland, T. R., & Dwyer, J. L. (2012). Landsat: Building a strong future. Remote Sensing of the Environment, 122, 22–29. https://doi.org/10.1016/j.rse.2011.09.022

Martin, R. C. (2009). Clean code: A handbook of agile software craftsmanship. Pearson Education.

Mertel, D. (2014). Docker: Lightweight Linux containers for consistent development and deployment. Linux Journal, 2014, 2.

Nataraj, S. (2014). The need for an introductory computer literacy course at the university level. International Journal of Business and Economics Research, 5(4), 71–73.

National Health and Medical Research Council (Australia). (2018). Australian code for the responsible conduct of research. NHMRC.

Popescu, S. C. (2007). Estimating biomass of individual pine trees using airborne LIDAR. Biomass & Bioenergy, 31(9), 646–655. https://doi.org/10.1016/j.biombioe.2007.06.022

Powers, S. M., & Hampton, S. E. (2019). Open science, reproducibility, and transparency in ecology. Ecological Applications, 29(1), 01822. https://doi.org/10.1002/eap.1822

Requa, M., Vaughan, G., David, J., & Cotton, B. (2016). Using cloud bursting to count trees and shrubs in Sub-Saharan Africa. In 2016 IEEE International Conference on Big Data (Big Data) (pp. 2960–2963). Retrieved from http://ieeexplore.ieee.org/document/7840947/. Accessed March 2, 2021.

Robinson, E., & Nolis, J. (2020). Build a career in data science. Safari.
Rüde, U., Willcox, K., McInnes, L. C., & Sterck, H. D. (2018). Research and education in computational science and engineering. *SIAM Review, 60*(3), 707–754. https://doi.org/10.1137/16M1096840

Salk, C. F. (2020). Tidying up conservation with messy data. *One Earth, 2*(5), 413–414. https://doi.org/10.1016/j.oneear.2020.04.015

Schmidt, R. (2013). Computing environment relationships and dependencies. In *Software engineering* (pp. 43–54). Morgan Kaufmann. https://www.sciencedirect.com/topics/computer-science/computing-environment

Simonson, M. R., Maurer, M., Montag-Torardi, M., & Whitaker, M. (1987). Development of a standardized test of computer literacy and a computer anxiety index. *Journal of Educational Computing Research, 3*(2), 231–247.

Tichy, W. F. (1985). RCS—A system for version control. *Software Practice and Experience, 15*(7), 637–654. https://doi.org/10.1002/spe.4380150703

U.S. Geoslo Survey. (2011). *U.S. Geological Survey fundamental science practices*. U.S. Geological Survey. https://doi.org/10.3133/cir1367

Valle, D., & Berdanier, A. (2012). Computer programming skills for environmental sciences. *Bulletin of the Ecological Society of America, 93*(4), 373–389.

Waller, D., Bartsch, M., & Erickson, R. (2020). Temperature-related responses of invasive (*Dreissena polymorpha*) and native mussels (Order: Unionida) to elevated carbon dioxide data. *US Geological Survey data release*. Retrieved from https://doi.org/10.5066/P9FMIHJM. Accessed October 29, 2020.

Waller, D., Bartsch, M. R., Lord, E. G., & Erickson, R. A. (2020). Temperature-related responses of an invasive mussel and 2 unionid mussels to elevated carbon dioxide. *Environmental Toxicology and Chemistry, 39*(8), 1546–1557. https://doi.org/10.1002/etc.4743

Wickham, H. (2019). *Advanced R* (2nd). CRC Press. Retrieved from https://adv-r.hadley.nz/. Accessed March 23, 2021.

Wickham, H., & Bryan, R. (2021). *R packages: Organize, test, document, and share your code*, (2nd ed.) (online only). O’Reilly. https://r-pkgs.org/

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., Bouwman, J., Brookes, A. J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C. T., Finkers, R., ... Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data, 3*(1), 1–9.

Wilson, E. O. (1998). *Consilience: The unity of knowledge* (1st Vintage books ed.). Vintage Books.

Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., & Teal, T. K. (2017). Good enough practices in scientific computing. *PLOS Computational Biology, 13*(6), e1005510. https://doi.org/10.1371/journal.pcbi.1005510

**How to cite this article:** Erickson, R. A., Burnett, J. L., Wiltermuth, M. T., Bulliner, E. A., & Hsu, L. (2021). Paths to computational fluency for natural resource educators, researchers, and managers. *Natural Resource Modeling*. 34, e12318. https://doi.org/10.1111/nrm.12318