nature biotechnology

Reproducibility of computational workflows is automated using continuous analysis

Brett K Beaulieu-Jones[1] & Casey S Greene[2]

Replication, validation and extension of experiments are crucial for scientific progress. Computational experiments are scriptable and should be easy to reproduce. However, computational analyses are designed and run in a specific computing environment, which may be difficult or impossible to match using written instructions. We report the development of continuous analysis, a workflow that enables reproducible computational analyses. Continuous analysis combines Docker, a container technology akin to virtual machines, with continuous integration, a software development technique, to automatically rerun a computational analysis whenever updates or improvements are made to source code or data. This enables researchers to reproduce results without contacting the study authors. Continuous analysis allows reviewers, editors or readers to verify reproducibility without manually downloading and rerunning code and can provide an audit trail for analyses of data that cannot be shared.

Leading scientific journals have highlighted a need for improved reproducibility to increase confidence in results and reduce the number of retractions[1–5]. In a recent survey, 90% of researchers acknowledged that there 'is a reproducibility crisis'[6]. Computational reproducibility is the ability to exactly reproduce results given the same data, as opposed to replication, which requires an independent experiment. Computational protocols used for research should be readily reproducible, because all of the steps are scripted into a machine-readable format. However, results can often be reproduced only with help from the original authors, and reproducing results requires a substantial time investment. Garijo *et al.*[7] estimated that it would take 280 h for a non-expert to reproduce the computational construction of a drug-target network for *Mycobacterium tuberculosis*[8]. Written descriptions of computational approaches can be difficult to understand, and may lack sufficient detail, including information about data pre-processing, model parameter selection and software versions, which are crucial for reproducibility. Indeed, Ioannidis *et al.*[9] showed that the outputs of 56% of microarray gene expression experiments could not be reproduced, and another 33% could be reproduced only with discrepancies. Additionally, Hothorn and Leisch found that more than 80% of manuscripts did not report software versions[10].

It has been proposed that open science could aid reproducibility[3,11]. In open science, the data and source code are shared. Intermediate results and project planning are sometimes also shared (as, for example, with Thinklab, https://thinklab.com/). Sharing data and source code is necessary, but not sufficient, to make research reproducible. Even when code and data are shared, variability in computing environments, operating systems and the software versions used during the original analysis make it difficult to reproduce results. It is common to use one or more software libraries during a project. Using these libraries creates a dependency on a particular version of the library; research code often works only with old versions of these libraries[12]. Developers of newer versions may rename functions, resulting in broken code, or change the way a function works to yield a slightly different result without returning an error. For example, Python 2 would perform integer division by default, so 5/2 would return 2. Python 3 performs floating-point division by default, so the same 5/2 command now returns 2.5. In addition, old or broken dependencies can mean that it is not possible for readers or reviewers to recreate the computational environment used by the authors of a study. In this case it becomes impossible to validate or extend results.

We first illustrate, using a practical example, the problem of reproducibility of computational studies. Then we describe the development and validation of a method named continuous analysis that can address this problem.

RESULTS

One example illustrating how data sharing does not automatically make science reproducible can be found in routine analyses of differential gene expression. Differential expression analyses are performed first by quantifying RNA in two or more conditions and then identifying transcripts whose expression levels are altered by the experiment. When a DNA microarray is used to measure transcript expression levels, positions on the array correspond to oligonucleotides of certain sequences, termed probes. A certain set of probes is used to estimate the expression level of each gene or transcript. As understanding of the genome changes, the optimal mapping of probes to genes or transcripts can change as well.

Dai *et al.*[13] publish and maintain BrainArray Custom CDF, a popular source of probe set description files that is routinely updated (http://brainarray.mbni.med.umich.edu/brainarray/Database/CustomCDF/genomic_curated_CDF.asp). Analyses that fail to report the probe set version or that were performed with probe set definitions that are now missing can never be reproduced. We set out to ascertain the extent

[1]Genomics and Computational Biology Graduate Group, Perelman School of Medicine, University of Pennsylvania, Philadelphia, Pennsylvania, USA. [2]Department of Systems Pharmacology and Translational Therapeutics, Perelman School of Medicine, University of Pennsylvania, Philadelphia, Pennsylvania, USA. Correspondence should be addressed to C.S.G. (csgreene@upenn.edu).

Received 10 June 2016; accepted 22 December 2016; published online 13 March 2017; doi:10.1038/nbt.3780

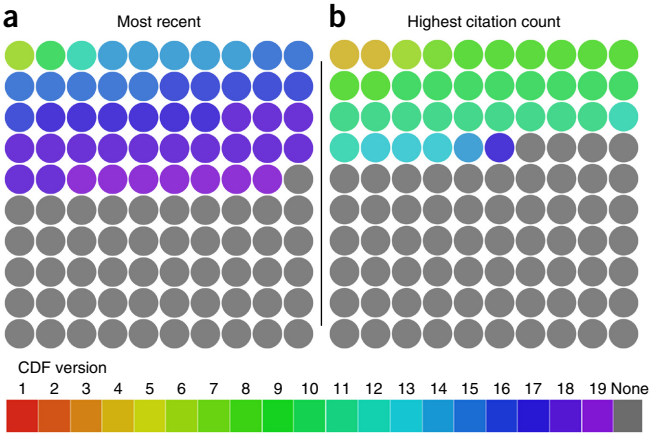

Figure 1 Reporting of Custom CDF file descriptors in published papers. (**a**,**b**) CDF version reporting in the 100 most recent (**a**) and the 100 most cited (**b**) papers citing Dai *et al.*[13] that use Custom CDF. Each circle represents one manuscript; color coding indicates the Custom CDF version used.

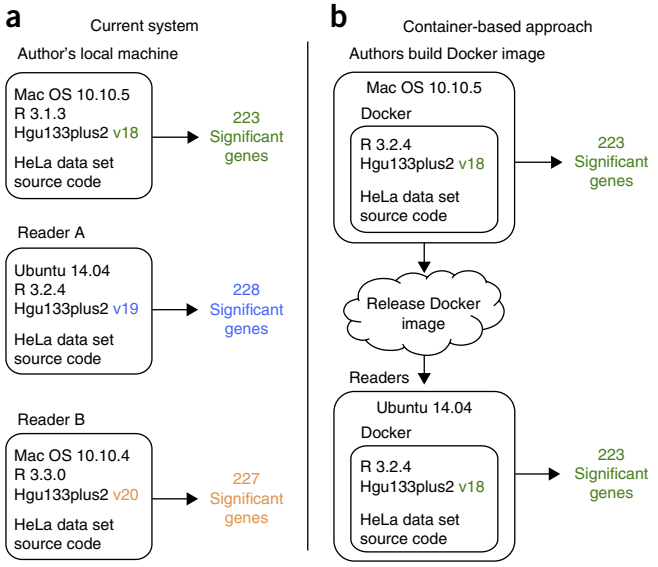

Figure 2 Research computing versus container-based approaches for differential gene expression analysis of HeLa cells. (**a**,**b**) Numbers of significantly differentially expressed genes identified using different versions of software packages (**a**) and a container-based approach with a defined computing environment (**b**). $n = 3$ biological replicates per group (wild-type or double-knockdown HeLa cells).

of this problem through a literature search. We analyzed the 100 most recently published papers citing Dai *et al.* that were accessible at our institution (**Supplementary Data 1**). We identified these manuscripts using Web of Science on 14 November 2016. We recorded the number of papers that cited a version of Custom CDF, including which version was cited. These articles adhered to expectations of citing methods appropriately, because they cited the sources of their probe set definitions. Of these 100 papers, 49 (49%) specified which version was used (**Fig. 1a**). These manuscripts reported the use of versions 6, 10, 12, 14, 15, 16, 17, 18 and 19 of the BrainArray Custom CDF. As of 14 November 2016, versions 6 and 12 were no longer available for download on the BrainArray website. To determine the extent to which a lack of version reporting of the probe set affects high-impact papers, we analyzed the 100 most cited papers that cite Dai *et al.*[13] (**Supplementary Data 2**), determined using Web of Science on 14 November 2016. Of these 100 papers, 36 (36%) specified which version of the Custom CDF was used (**Fig. 1b**). These manuscripts used versions 4, 6, 7, 8, 10, 11, 12, 13, 14 and 17. Versions 4, 6, 7, 11 and 12 were not available for download as of 14 November 2016.

In order to evaluate how different versions of BrainArray Custom CDF affect the outcomes of standard analyses, we downloaded a recently published gene expression data set (GEO GSE47664). The experiment using this data set measured gene expression in normal HeLa cells and HeLa cells in which *TIA1* and *TIAR* (*TIAL1*) were knocked down[14]. We ran the same source code using the same data set with three versions of the BrainArray Custom CDF library (versions 18, 19 and 20). Each version identified a different number of significantly altered genes (**Fig. 2a**). Fifteen genes identified as significant in version 19 were not identified in version 18, and ten genes identified as significant in version 18 were not identified in version 19. Eighteen genes identified as significant in version 20 were not found in version 18, and 14 genes identified as significant in version 18 were not identified in version 20. These results indicate that study outcomes are not reproducible without an accurate version number.

Using Docker containers improves reproducibility

To improve reproducibility, researchers can maintain dependencies using the free open-source software tool Docker[12,15]. Docker can be used to create an 'image' that allows users to download and run a container, which is a minimalist virtual machine with a predefined computing environment. Docker images can be several gigabytes in size but, once downloaded, can be started in a matter of seconds and have minimal overhead[12]. This technology has been widely adopted and is now supported by many popular cloud providers, including Amazon, Google and Microsoft.

Docker wraps software into a container that includes everything the software needs to run (operating system, system tools, installed software libraries and so on). This allows the software to run the exact same way in any environment. Boettiger introduced Docker containers as a path to reproducible research by eliminating dependency management, remediating issues caused by imprecise documentation, limiting the effects of code rot (dependencies to specific software library versions) and removing barriers to software reuse[12]. In addition, Docker images can be tagged to coincide with software releases and paper revisions. This means that even as software is updated, the exact computing environment of an older version can be available through the tag of the container's revision history.

In order to assess whether using Docker containers could improve reproducibility of the same experiment, we also carried out an analysis of differential gene expression using Docker containers on mismatched machines (available at http://doi.org/10.5281/zenodo.59892). This process enabled versions to be matched and produced the same number and set of differentially expressed genes (**Fig. 2b**).

Docker is a useful starting point for reproducible workflows. Although it helps to match the computing environment, users must manually rebuild the environment, rerun the analysis pipeline and update the container after each relevant change. In addition, Docker does not produce logs of what preprocessing steps and analyses were performed in the computing environment. It also does not automatically track results alongside the specific versions of the code and data that generated them. In summary, Docker can provide manual reproducibility when it is used appropriately.

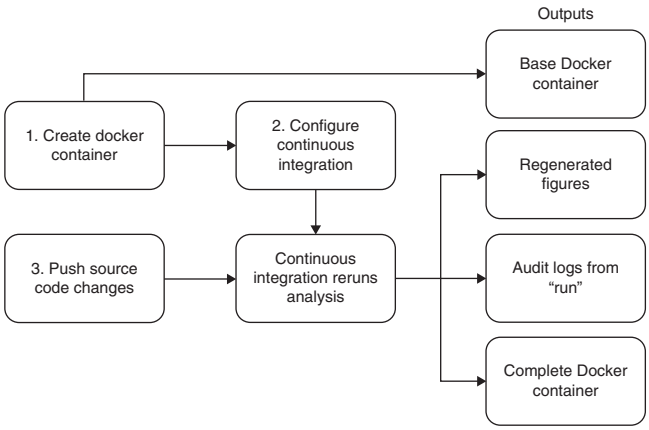

Figure 3 Setting up continuous analysis. Continuous analysis can be set up in three steps. First, the researcher creates a Docker container with the required software (1). The researcher then configures a continuous integration service to use this Docker image (2) then pushes code that includes a script capable of running the analyses from start to finish (3). The continuous integration provider runs the latest version of code in the specified Docker environment without manual intervention. This generates a Docker container with intermediate results that allows anyone to rerun analysis in the same environment, produces updated figures and stores logs describing what occurred. Example configurations are available in Online Methods and at https://github.com/greenelab/continuous_analysis.

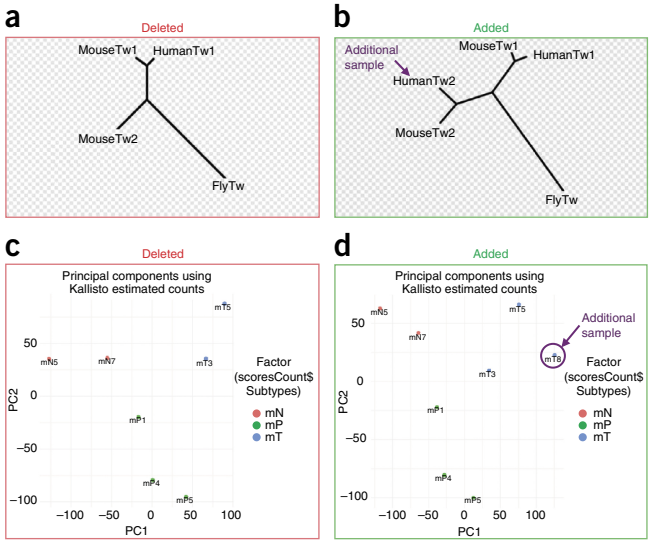

Figure 4 Reproducible workflows with continuous analysis. (**a**,**b**) Phylogenetic tree building with four mRNA samples (MouseTw1, HumanTw1, MouseTw2 and FlyTw) (**a**) and an additional gene (HumanTw2) (**b**). (**c**,**d**) RNA-seq differential expression experiment principal component (PC) analysis before (**c**) and after (**d**) addition of a sample (mT8).

Continuous analysis

Our goal in developing continuous analysis was to produce an automatic and verifiable end-to-end run for computational analyses with minimal startup costs. The status quo requires researchers to describe each step of an analysis and communicate exact versions of software library dependencies used, which can be hundreds or thousands of packages for modern operating systems. To avoid requiring readers and reviewers to download and install multiple software packages and data sets, continuous analysis preserves the exact computing environment used for the original analysis. A Docker container is built at the time of original analysis and thus includes the versions used by the original authors without the risk of packages later becoming inaccessible. The 'continuous' aspect refers to an analysis being rerun, the results saved in version control and the container being automatically updated after any relevant changes to the software script or data.

Continuous analysis is an extension of continuous integration[16], which is widely used in software development. In continuous integration, whenever developers update code in a source control repository, an automated build process is triggered. This automated build process first runs any existing test scripts in an attempt to catch bugs introduced into software. If there are no tests or if the software passes the tests, the software is automatically sent to remote servers so that users worldwide can access it.

Our continuous analysis workflows (**Fig. 3**) use continuous integration services to run computational analyses, update figures and publish changes to online repositories whenever changes are made to the source code used in an experiment. We provide continuous analysis workflows for popular continuous integration systems that can be used with multiple types of computing environments, including local and cloud computing.

In our continuous analysis workflows, a continuous integration service is used to monitor the source code repository. Whenever a change is made to a user-specified branch of the repository, the service reruns the scientific analysis. Workflows are defined in files written in YAML that specify which configuration parameters and commands should be run. The YAML language is widely used by continuous integration services to specify a human-readable set of instructions. The continuous analysis YAML files that we developed enable users to employ local computing, cloud-based computing or commercial service providers.

Each workflow begins by specifying a base Docker image to replicate the researcher's computing environment. The YAML files that we developed provide a place for researchers to choose a base Docker image. Using Docker enables other researchers to rerun code in a computing environment that matches that used by the original authors, even if they do not duplicate the original authors' continuous integration configuration. Next, the continuous analysis workflow YAML files specify one or more shell commands required to perform the analysis. Researchers can replace the commands in our examples with their own analytical code. Executing these steps generates the relevant figures from the analysis. Our YAML implementation of continuous analysis then updates the remote source code repository by adding figures and results generated during the run. Finally, a Docker container with the final computing environment is automatically updated. This continuous analysis process allows changes to be tracked automatically as a project proceeds and pairs each result with the source code, data and Docker container used to generate it.

Using continuous integration in this fashion automatically generates a log of exactly what code was run that is synchronized to the code, data and computing environment (**Supplementary Fig. 1**). Version control systems enable images to be easily compared, which provides users with the ability to observe results before and after changes (**Fig. 4**). Interactive development tools such as Jupyter[17,18], Rmarkdown[19,20] and Sweave[21] can be incorporated to present the code and analysis in a logical graphical manner. For example, we recently used Jupyter with continuous analysis[22] and a corresponding repository (available at https://doi.org/10.5281/zenodo.46165 (archival version) and at https://github.com/greenelab/daps). Reviewers can follow what was done in an audit-like fashion without having to install and run software and with confidence that analyses are reproducible.

When authors are ready to publish their work, they should archive their repositories, which contain the automatically generated results alongside the analytical source code and scripted commands for data retrieval. With the continuous analysis workflow, authors can use the 'docker save' command to export the latest static container, which should also be archived. An increasing number of services allow digital artifacts to be archived and distributed, including Figshare or Zenodo. Journals may also allow authors to upload these files as supplementary elements. If the archiving service used by the authors provides a digital object identifier (DOI), future users can easily cite the computing environment and source code. For example, our continuous analysis environment and results are available (Online Methods) in this fashion with results, and our source code is provided as **Supplementary Source Code**.

Continuous analysis is set up once per project and will then run automatically for the entire project. We provide example YAML workflows for commonly used services (https://github.com/greenelab/continuous_analysis). Researchers can replace the steps in our example analyses with their own commands to enable automatic reproducibility of their own projects.

Setting up continuous analysis

Three steps are required to set up continuous analysis. First is the creation of a Dockerfile, which specifies the software required for their analysis. Second is connection of a continuous integration service to the version control system and addition of a continuous analysis command script to run the analysis. Finally, changes must be saved to the version control system. Many researchers already perform the first and third tasks in the course of standard procedures for computational research.

The continuous integration system (**Fig. 3**) will automatically rerun the specified analysis with each change, precisely matching the source code and results. It can also be set to listen and run only when changes are marked in a specific way, for example, by committing to a specific 'staging' branch. For the first project, continuous analysis can be set up in less than a day. For subsequent projects, the continuous analysis protocol can be amended in less than an hour.

We have set up continuous analysis using the free and open-source Drone software (https://github.com/drone/drone) on a PC and connected it to the GitHub version control service (Online Methods). This method is free to users. Our GitHub repository and **Supplementary Source Code** include continuous analysis YAML scripts for local, cloud-based and full-service paid configurations (http://doi.org/10.5281/zenodo.178613 (archival) and https://github.com/greenelab/continuous_analysis). However, it is important to note that although full-service providers can be set up in minutes, they may impose computational resource limits or monthly fees. Private installations require configuration but can scale to a local cluster or cloud service to match the computational complexity of all types of research. With free, open-source continuous integration software, computing resources are the only associated costs.

We suggest a development workflow in which continuous analysis runs only on a selected branch (**Supplementary Fig. 2**). Our example setup configures a staging branch for this purpose. Researchers can push to this branch whenever they would like to generate results files and figures. If the updates to this branch succeed, the changes—along with the results of analyses—are then automatically carried over to the master or production branch and released.

Reproducible workflows

After the initial setup, continuous analysis can be adopted into existing workflows that use source control systems. We used continuous analysis in our work on neural networks to stratify patients on the basis of their electronic health records (http://doi.org/10.5281/zenodo.46165). In addition, we ran two example analyses using continuous analysis: a phylogenetic tree–building analysis and an RNA-seq differential expression analysis.

The phylogenetic tree–building example (Online Methods) aligned four mRNA sequences (two mouse sequences, one human sequence and one *Drosophila* sequence) using MAFFT[23] and built a phylogenetic tree with these alignments using PHYLIP[24] (**Fig. 4a**). After we added an additional sample (a human sequence labeled HumanTw2), continuous analysis rebuilt the tree (**Fig. 4b**).

The RNA-seq example (Online Methods) demonstrates differential expression analysis between three organoid models of pancreatic cancer in mice based on Boj *et al.*[25] (GEO GSE63348), reusing source code from D. Balli (http://web.archive.org/web/20150923171026/https://benchtobioinformatics.wordpress.com/). This analysis quantified transcript counts using kallisto[26], limma[27,28] and sleuth[29]; performed principal component analysis and ran a differential expression analysis. The analysis was initially performed with seven samples (**Fig. 4c**). An eighth sample was added to show how continuous analysis tracks results (**Fig. 4d**). This example also demonstrated the ability of continuous analysis to scale to the analysis of large data sets—the GEO accession we used includes 150 GB of data (approximately 480 million reads).

DISCUSSION

Continuous analysis provides a verifiable end-to-end run of scientific software in a fully specified environment, enabling true reproduction of computational analyses. Because continuous analysis runs automatically, it can be set up at the start of any project to provide an audit trail that allows reviewers, editors and readers to assess reproducibility without a large time commitment. If readers or reviewers need to rerun the code on their own (for example, to change a parameter and evaluate the impact on results), they can easily do so with a Docker container in which the final computing environment and results have been automatically kept up to date. Version control systems enable automatic notification of code updates and new runs to users who 'star' or 'watch' a repository on services such as GitHub, GitLab or Bitbucket. Wide adoption of continuous analysis could conceivably be linked with the peer review and publication process, allowing interested parties to be notified of updates.

Continuous analysis can also be applied to closed data that cannot be released, such as patient data. Without continuous analysis, reproducing or replicating computational analyses based on closed data is dependent on complete description of each step, which is often relegated to supplementary information. Readers and reviewers must then carefully follow complex written instructions without confirmation of whether they are on the right track. The pairing of automatically updated containers, source code and results with the audit log provides readers with confidence that results would be replicable if the data were available. This allows independent researchers to attempt to replicate findings in their own non-public data sets without worrying that failure to replicate could be caused by source code or environment differences.

Continuous analysis has limitations. It may be impractical to use continuous analysis at every commit for generic pre-processing steps involving very large data sets or analyses requiring particularly high computational costs. In particular, steps that take days to run or incur substantial costs in computational resources (for example, genotype imputation) may be too expensive for existing providers[30]. We recommend the use of continuous analysis whenever a single machine can be used. For workflows that require cluster computing, continuous analysis is technically feasible but requires substantial

expertise in systems administration, because the cluster must be provisioned automatically.

For small data sets and less intensive computational workflows, it is easiest to use a full-service continuous integration service. These services have the shortest setup times—often requiring only that the user enable the service and add a single file to their source code. With private data or for analyses that include large data sets or require significant computing, cloud-based or locally hosted continuous integration servers can be employed.

Stodden *et al.*[31] highlight the importance of capturing and sharing data, software and the computational environments. Continuous analysis addresses reproducibility in this narrow sense by automatically capturing the computational reagents needed to generate the same results from the same inputs. It does not address reproducibility in the broader sense: such as the robustness of results to parameter settings, starting conditions and partitions in the data.

METHODS

Methods, including statements of data availability and any associated accession codes and references, are available in the online version of the paper.

Note: Any Supplementary Information and Source Data files are available in the online version of the paper.

ACKNOWLEDGMENTS

We would like to thank D. Balli (University of Pennsylvania) for providing the RNA-seq analysis design, K. Siewert (University of Pennsylvania) for providing the phylogenetic analysis design and A. Whan (Commonwealth Scientific and Industrial Research Organization) for contributing a Travis-CI implementation. We also thank M. Paul, Y. Park, G. Way, A. Campbell, J. Taroni and L. Zhou for serving as usability testers during the implementation of continuous analysis. This work was supported by the Gordon and Betty Moore Foundation under a Data Driven Discovery Investigator Award to C.S.G. (GBMF 4552). B.K.B.-J. was supported by a Commonwealth Universal Research Enhancement (CURE) Program grant from the Pennsylvania Department of Health and by US National Institutes of Health grants AI116794 and LM010098.

AUTHOR CONTRIBUTIONS

B.K.B.-J. and C.S.G. conceived the study and designed the solution. B.K.B.-J. implemented continuous analysis. B.K.B.-J. and C.S.G. wrote the manuscript.

COMPETING FINANCIAL INTERESTS

The authors declare no competing financial interests.

1. Anonymous. Rebooting review. *Nat. Biotechnol.* **33**, 319 (2015).
2. Anonymous. Software with impact. *Nat. Methods* **11**, 211 (2014).
3. Peng, R.D. Reproducible research in computational science. *Science* **334**, 1226–1227 (2011).
4. McNutt, M. Reproducibility. *Science* **343**, 229 (2014).
5. Anonymous. Illuminating the black box. *Nature* **442**, 1 (2006).
6. Baker, M. 1,500 scientists lift the lid on reproducibility. *Nature* **533**, 452–454 (2016).
7. Garijo, D. *et al.* Quantifying reproducibility in computational biology: the case of the tuberculosis drugome. *PLoS One* **8**, e80278 (2013).
8. Kinnings, S.L. *et al.* The *Mycobacterium tuberculosis* drugome and its polypharmacological implications. *PLoS Comput. Biol.* **6**, e1000976 (2010).
9. Ioannidis, J.P.A. *et al.* Repeatability of published microarray gene expression analyses. *Nat. Genet.* **41**, 149–155 (2009).
10. Hothorn, T. & Leisch, F. Case studies in reproducibility. *Brief. Bioinform.* **12**, 288–300 (2011).
11. Groves, T. & Godlee, F. Open science and reproducible research. *Br. Med. J.* **344**, e4383 (2012).
12. Boettiger, C. An introduction to Docker for reproducible research, with examples from the R environment. *ACM SIGOPS Oper. Syst. Rev.* **49**, 71–79 (2015).
13. Dai, M. *et al.* Evolving gene/transcript definitions significantly alter the interpretation of GeneChip data. *Nucleic Acids Res.* **33**, e175 (2005).
14. Núñez, M., Sánchez-Jiménez, C., Alcalde, J. & Izquierdo, J.M. Long-term reduction of T-cell intracellular antigens reveals a transcriptome associated with extracellular matrix and cell adhesion components. *PLoS One* **9**, e113141 (2014).
15. Docker v.1.12.5, build 7392c3b (Docker, 2016).
16. Duvall, P., Matyas, S. & Glover, A. *Continuous Integration: Improving Software Quality and Reducing Risk* (Addison-Wesley Professional, 2007).
17. Pérez, F. & Granger, B.E. IPython: a system for interactive scientific computing. *Comput. Sci. Eng.* **9**, 21–29 (2007).
18. Jupyter v.4.1.0 (Project Jupyter, 2016).
19. RStudio: Integrated Development for R: v.0.98.1083 (RStudio Inc., 2015).
20. Baumer, B., Cetinkaya-Rundel, M., Bray, A., Loi, L. & Horton, N.J.R. Markdown: integrating a reproducible analysis tool into introductory statistics. *Technol. Innov. Stat. Educ.* **8**, uclastat_cts_tise_20118 (2014).
21. Friedrich Leisch. Sweave: dynamic generation of statistical reports using literate data analysis. *Proc. Comput. Stat.* **2002**, 575–580 (2002).
22. Beaulieu-Jones, B.K. & Greene, C.S. Semi-supervised learning of the electronic health record for phenotype stratification. *J. Biomed. Inform.* **64**, 168–178 (2016).
23. Katoh, K., Misawa, K., Kuma, K. & Miyata, T. MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Res.* **30**, 3059–3066 (2002).
24. Felsenstein, J. PHYLIP—phylogeny inference package (version 3.2). *Cladistics* **5**, 164–166 (1989).
25. Boj, S.F. *et al.* Organoid models of human and mouse ductal pancreatic cancer. *Cell* **160**, 324–338 (2015).
26. Bray, N.L., Pimentel, H., Melsted, P. & Pachter, L. Near-optimal probabilistic RNA-seq quantification. *Nat. Biotechnol.* **34**, 525–527 (2016).
27. Ritchie, M.E. *et al.* limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Res.* **43**, e47 (2015).
28. Smyth, G.K. Linear models and empirical bayes methods for assessing differential expression in microarray experiments. *Stat. Appl. Genet. Mol. Biol.* **3**, e3 (2004).
29. Pimentel, H.J., Bray, N., Puente, S., Melsted, P. & Pachter, L. Differential analysis of RNA-seq incorporating quantification uncertainty. Preprint at *bioRxiv* https://doi.org/10.1101/058164 (2016).
30. Souilmi, Y. *et al.* Scalable and cost-effective NGS genotyping in the cloud. *BMC Med. Genomics* **8**, 64 (2015).
31. Stodden, V. *et al.* Enhancing reproducibility for computational methods. *Science* **354**, 1240–1241 (2016).

ONLINE METHODS

Assessment of reporting of custom CDF versions. We performed a literature analysis of the 104 most recently published articles citing Dai *et al.*[13] that were accessible at our institution using the Web of Science on 14 November 2016. We aimed to capture 100 articles that included expression analysis and used this resource. We excluded four articles that did not perform expression analysis using the Custom CDF and thus would not be expected to cite a version. We repeated this process for the 116 most cited articles that cited Dai *et al.*[13] and were accessible at our institution using the Web of Science on 14 November 2016. We again aimed to capture 100 articles that included expression analysis. We excluded 16 articles that did not quantify expression with the Custom CDF and would not be expected to cite a version. For the 100 articles in each set that included expression analysis, we searched for the citation or mention of Custom CDF and Dai *et al.* and examined the methods section and any supplementary materials to determine which, if any, Custom CDF version was specified.

HeLa cell differential expression analysis using BrainArray Custom CDF. We compared the results of a differential expression using versions 18, 19 and 20 of the BrainArray Custom CDF files[13]. We performed the differential expression analysis between wild-type HeLa cells and HeLa cells with a double-knockdown of T cell intracellular antigen 1 (TIA1) and TIA1-related or TIAL1-like (TIAR/TIAL1) proteins (GEO GSE47664). The experiment included 3 biological replicates of the wild-type HeLA cells and 3 biological replicates of the HeLa cells with double knockdown.

Statistical analysis. Differential expression analysis was performed using the Bioconductor multtest R package[32] to perform a two-sided *t*-test. Bonferroni adjustment was used for multiple testing correction. The complete *P* values for Custom CDF versions 18, 19 and 20 are included (**Supplementary Data 3–5**). For this analysis, an α threshold of 10^{-5} was used. Continuous analysis was used for this example and is available at https://github.com/greenelab/continuous_analysis_brainarray.

Setting up continuous analysis. *Installing and configuring Drone.* We used a private local continuous integration server with Drone 0.4 for the examples in this work. A detailed walkthrough is available at http://doi.org/10.5281/zenodo.178613. The first step in setting up a local continuous integration server is to install Docker on the local machine. We pulled the Drone image via the Docker command 'sudo docker pull drone/drone:0.4'. We created a new application within GitHub at https://github.com/settings/developers (note this requires users to register for a free account). The homepage URL was the local machine's IP address and the callback URL was the local machine's IP address followed by '/authorize'. After creating the application, we noted the Client ID and Client Secret generated by GitHub. Next, we created the Drone configuration file on our local machine (at '/etc/drone/dronerc'). To do this, we first created a directory ('sudo mkdir/etc/drone'). Then, we created a new file named dronerc in the new directory with the client information in the following format, filling in the Client ID and Secret with the information obtained from GitHub:

```
REMOTE_DRIVER=github
REMOTE_CONFIG=https://github.com?client_id=….&client_secret= ….
```

The drone instance was ready to run:

```
        sudo docker run \
–volume /var/lib/drone:/var/lib/drone \
–volume /var/run/docker.sock:/var/run/docker.sock \
–env-file /etc/drone/dronerc \
–restart=always \
–publish=80:8000 \
–detach=true \
–name=drone \
drone/drone:0.4
```

The continuous integration server was accessed at the login page (IP-address followed by '/login'). We clicked the login button and chose the repository we wished to activate for continuous analysis. This automatically created a webhook in GitHub to notify Drone when new changes were made to the repository.

Creating a Docker image to replicate the research environment. We used a Dockerfile to define a Docker image with the software required by the analysis (examples are provided below). This file was named Dockerfile. The first line of the file specified a base image to pull from. We used Ubuntu 14.04, so we set the first line to 'FROM Ubuntu:14.04'. We specified the maintainer with contact information on the next line as: MAINTAINER "Brett Beaulieu-Jones" brettbe@med.upenn.edu. Next, commands to install the appropriate packages were added to the file:

```
RUN apt-get install –y git python-numpy wget gcc python-dev python-setuptools python-dev build-essential
```

After constructing the Dockerfile, we built the Docker image using the Docker build command: 'sudo docker build –t username/image_name'. After building the image, we pushed it to Docker Hub in order to share the exact computing environment. We used the following commands, replacing 'username' and 'image_name' with the appropriate values: 'sudo docker login' and 'sudo docker push username/image_name'. In addition to pushing the tagged docker images to Docker Hub, we saved static versions of the docker images before and after analysis ('docker save brettbj/continuous_analysis_base > continuous_analysis_base' and 'docker save brettbj/continuous_analysis > continuous_analysis/). These are available at Figshare (http://dx.doi.org/10.6084/m9.figshare.3545156.v1).

Running a continuous analysis. After configuration, the analysis was run using a file called '.drone.yml'. When this file was added to a repository, Drone performed all of the commands within it (**Supplementary Fig. 3**). Whenever changes are made to the source code, Drone repeats these steps. Example '.drone.yml' files are available at GitHub (see below). Within continuous analysis, all commands were executed within a build section of the .drone.yml file. First, the base Docker image created previously was pulled ('build: image: username/image_name'). Then shell scripting was used to rerun the appropriate analysis.

commands: ('- mkdir –p output - …'). To run continuous analysis only on the staging branch, a final section was added: 'branches: - staging'.

Phylogenetic tree–building example. This example aligned five mRNA sequences and uses these alignments to build simple phylogenies. We used five mRNA sequences accessible from the NCBI nucleotide database: Twist, fly (NM_079092, splice form A); Twist1, human (NM_000474); Twist1, mouse (NM_011658); Twist2, human (NM_057179); Twist2, mouse (NM_007855). The first analysis did not include Twist2, human. The second analysis included all five sequences (**Supplementary Fig. 4**). The sequences were aligned using MAFFT[23] and converted to PHYLIP interleaved format using EMBOSS Seqret[33]. A maximum parsimony tree was generated for the sequences using PHYLIP[24] DNAPARS and a representation of this tree was drawn with PHYLIP drawtree. PHYLIP Seqboot was used to assess robustness of the generated tree and PHYLIP consense to determine the consensus tree form the bootstrapped trees. The complete continuous analysis runs are available at https://github.com/greenelab/continuous_analysis_phylo and Docker images before and after analysis are available at http://dx.doi.org/10.6084/m9.figshare.3545156.v1.

RNA-seq differential analysis of mouse models for pancreatic cancer. This workflow was based on work by D. Balli (http://web.archive.org/web/20150923171026/https://benchtobioinformatics.wordpress.com/) and uses mouse organoid cultures generated by Boj *et al.*[25]. Boj *et al.* generated organoids from three different tissues: normal pancreas (mN), early stage lesions (mP) and pancreatic adenocarcinoma (mT). The authors then performed RNA-seq on these organoids (GEO GSE63348; SRA SRP049959). We performed differential expression analysis initially on seven samples: two normal, three mP and two mT (150 GB FASTQ format, 480 million reads. An eighth sample (mT) was added in the second run to demonstrate the differences under continuous analysis. Four pre-processing steps were performed before beginning continuous analysis. These steps were performed to reduce the amount of run time for continuous analysis and to limit the necessary bandwidth. Pre-processing steps should be performed only with time- and/or resource-heavy tasks that follow a standard workflow. First, the samples were

downloaded from Sequence Read Archive (SRA SRP049959). The SRA[29] toolkit was used to split the SRA files in to FASTQ files. The mouse reference genome (mm10) assembly was downloaded (http://hgdownload.soe.ucsc.edu/goldenPath/mm10/bigZips/refMrna.fa.gz). Finally, these files were stored in a folder accessible to local FTP so they would not need to be re-downloaded with each run. Within the continuous analysis process, kallisto[29] was used to generate an index file from the reference mouse genome and then to quantify abundance of transcripts for each RNA-seq sample. Next, lowly expressed genes were filtered out (<1) and a principal components plot was generated (two components) (**Supplementary Fig. 5**). Finally, a limma[27] linear model was fit for differential gene expression analysis, and the results were plotted in the form of a volcano plot (**Supplementary Fig. 6**).

The complete continuous analysis runs (https://github.com/greenelab/continuous_analysis_rnaseq and Docker images before and after analysis were uploaded and made available (http://dx.doi.org/10.6084/m9.figshare.3545156.v1).

Code availability. The Docker images for all three experiments are available on Figshare (http://dx.doi.org/10.6084/m9.figshare.3545156.v1). The continuous analysis examples and instructions accompanying this manuscript are available on Zenodo (http://doi.org/10.5281/zenodo.178613). Instructions and examples will be periodically updated on GitHub, and contributions are welcomed (https://github.com/greenelab/continuous_analysis).

Data availability. HeLa cell data (wild-type and double-knockout) are available through GEO (accession number GSE47664). The genes used in the phylogenetic tree–building example are available via the NCBI nucleotide database (NM_079092, NM_000474, NM_011658, NM_057179 and NM_007855). The RNA-seq data used for the larger differential expression analysis are available in the Sequence Read Archive (SRA SRP049959). The continuous analysis RNA-seq example is available at https://github.com/greenelab/continuous_analysis_rnaseq; the BrainArray example is available at https://github.com/greenelab/continuous_analysis_brainarray; the phylogenetic tree–building example is at https://github.com/greenelab/continuous_analysis_phylo. Analysis of differential gene expression using Docker containers on mismatched machines is available at http://doi.org/10.5281/zenodo.59892.

32. Pollard, K.S., Dudoit, S. & van der Laan, M.J. Multiple testing procedures: the multtest package and applications to genomics. in *Bioinformatics and Computational Biology Solutions Using R and Bioconductor* (eds. Gentleman, R. *et al*.) (Springer New York, 2005).
33. Rice, P., Longden, I. & Bleasby, A. EMBOSS: the European Molecular Biology Open Software Suite. *Trends Genet.* **16**, 276–277 (2000).

doi:10.1038/nbt.3780