Reproducible Computational Workflows with Continuous Analysis

Brett K. Beaulieu-Jones[1], and Casey S. Greene[2,+]

[1] Genomics and Computational Biology Graduate Group. Perelman School of Medicine. University of Pennsylvania.
[2] Department of Systems Pharmacology and Translational Therapeutics. Perelman School of Medicine. University of Pennsylvania

[+] Corresponding Author
email: csgreene@upenn.edu
address:
3400 Civic Center Blvd.
10-131 SCTR
Philadelphia, PA 19103

Abstract

Reproducing experiments is vital to science. Being able to replicate, validate and extend previous work also speeds new research projects. Reproducing computational biology experiments, which are scripted, should be straightforward. But reproducing such work remains challenging and time consuming. In the ideal world we would be able to quickly and easily rewind to the precise computing environment where results were generated. We'd then be able to reproduce the original analysis or perform new analyses. We introduce a process termed "continuous analysis" which provides inherent reproducibility to computational research at a minimal cost to the researcher. Continuous analysis combines Docker, a container service similar to virtual machines, with continuous integration, a popular software development technique, to automatically re-run computational analysis whenever relevant changes are made to the source code. This allows results to be reproduced quickly, accurately and without needing to contact the original authors. Continuous analysis also provides an audit trail for analyses that use data with sharing restrictions. This allows reviewers, editors, and readers to verify reproducibility without manually downloading and rerunning any code.

The Current State of Reproducibility

Leading scientific journals have targeted reproducibility to increase readers' confidence in results and reduce retractions[1–5]. In a recent survey, 90% of researchers acknowledged a reproducibility crisis[6]. Research that uses computational protocols should be particularly amenable to reproducible workflows because all of the steps are scripted into a machine-readable format. But written descriptions of computational approaches can be difficult to understand and may lack required parameters. Even when results can be reproduced, the process often requires a substantial time investment and help from the original authors. Garijo et al.[7] estimated it would take 280 hours for a non-expert to reproduce a paper describing a computational construction of a drug-target network for *Mycobacterium tuberculosis*[8]. These are the good scenarios: the results behind most computational papers are not readily reproducible[7,9–11].

The practice of "open science" has been discussed as a means to aid reproducibility[3,12]. In open science the data and source code are shared. Sharing can also extend to intermediate results and project planning[13]. Sharing data and source code is currently necessary but not sufficient to make research reproducible. Even when code and data are shared, it remains difficult to reproduce results due to differing computing environments, operating systems, library dependencies etc. It is common to use one or more open source libraries on a project, and research code quickly becomes dependent on old versions of these libraries as software advances[14]. These old or broken dependencies make it difficult for readers and reviewers to recreate the environment of the original researchers, whether to validate or extend their work.

An example of where sharing data does not automatically make science reproducible occurs in the most standard of places: differential gene expression analysis. Such analyses are routine. Our understanding of the genome, including transcriptome annotations, have improved and updated probe set definitions are available[15]. Analyses relying on unspecified probe set definitions cannot be reproduced using current definitions.

We analyzed the fifteen most recently published papers that cite Dai et al., a common source for custom chip description files (CustomCDF), that were accessible at our institution[16–31]. We identified these manuscripts using Web of Science on May 31, 2016. We recorded the number of papers that cited a version of CustomCDF, as well as which version was cited. We expect this analysis to provide an upper bound on reproducible work: these papers specifically cited the source of their CDFs. Of these fifteen papers, nine (60%) specified which version they used. These nine used versions 11, 15, 16, 17, 18, and 19 of the BrainArray CustomCDF.

This initial analysis was performed based on article recency without regard to article impact. To determine the extent to which this issue affects high impact papers, we performed a parallel evaluation for the ten most cited papers[32–41] that cite Dai et al. We determined the ten most cited papers using Web of Science on May 31, 2016. Of these ten papers, one[38] (10%) specified which version of the CustomCDF was used. That paper used version 11 of the BrainArray CustomCDF.

We sought to determine which versions were currently in use in the field. We asked three individuals who performed microarray analysis recently, and we accessed and evaluated two cluster systems used for processing data. We found that each individual had one of the three most recently released versions installed (18, 19, and 20), and versions 18 and 19 were installed on cluster systems.

To evaluate the impact of differing CDF versions, we downloaded a recently published public gene expression dataset. This experiment examined differential expression between normal HeLa cells and HeLa cells with TIA1 or TIAR knocked down[42]. We performed a parallel analysis using each of the three versions that we found installed on machines that we could access (18, 19, and 20). Each version identifies a different number of significantly altered genes (Figure 1A), demonstrating the challenge of reproducible analysis. We performed a parallel analysis for differential expression using Docker containers on mismatched machines. This process allows versions to be matched, and produces the same number and set of differentially expressed genes (Figure 1B).

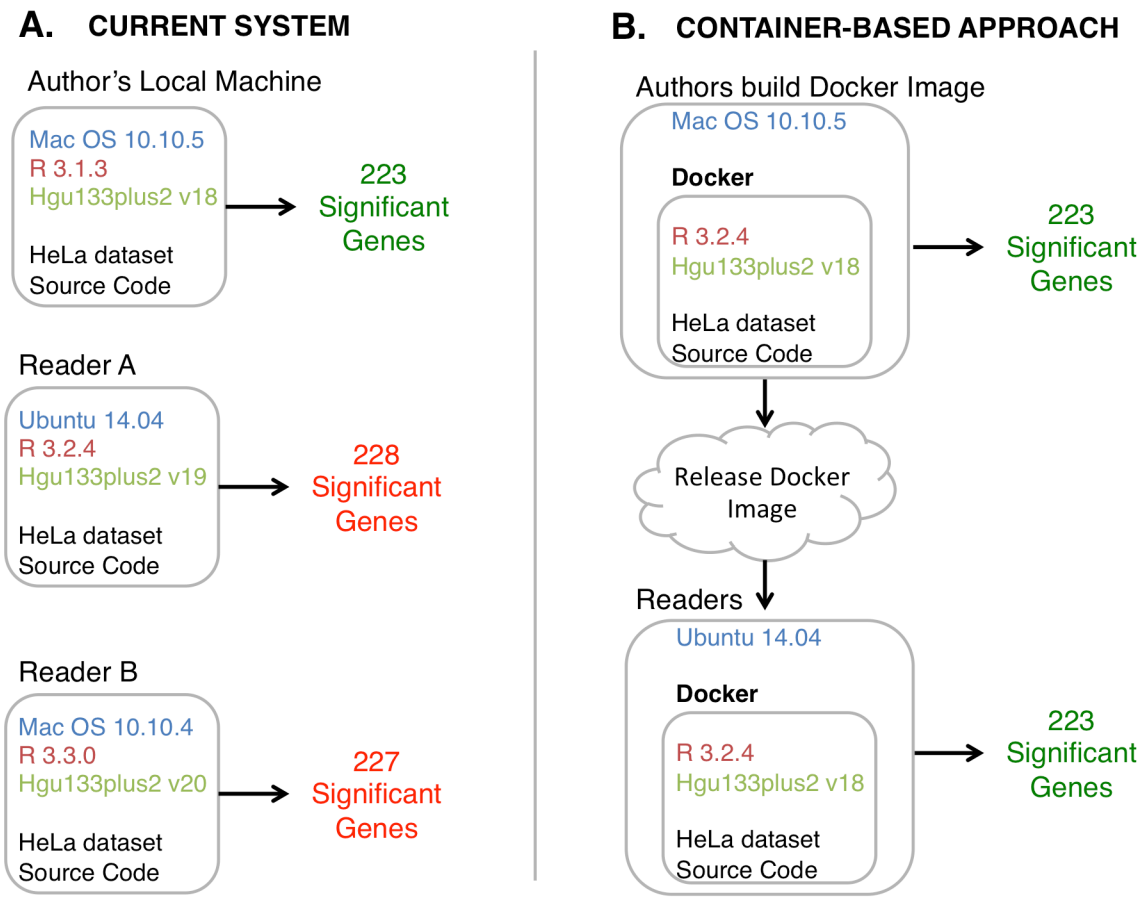

Figure 1. Current state of research computing vs. container-based approaches. **A.)** The status quo requires a reader or reviewer to find and install specific versions of dependencies. These dependencies can become difficult to find and may become incompatible with newer versions of other software packages. Different versions of packages identify different numbers of significantly differentially expressed genes from the same source code and data. **B.)** Containers define a computing environment that captures dependencies. In container-based systems, the results are the same regardless of the host system.

Continuous Analysis in Computational Workflows

We developed continuous analysis to produce a verifiable end-to-end run of computational research with minimal start-up costs. In contrast with the status quo, continuous analysis preserves the computing environment and maintains the versions of dependencies. We described the benefits of containerized approaches above, but maintaining, running and distributing Docker images manually would become time consuming. Integrating Docker into a continuous scientific analysis pipeline meets three criteria: (1) readers and reviewers can follow exactly what was done in an "audit" fashion without running code; (2) anyone can re-run code in a computing environment matching the original authors; and (3) the solution imposes

zero to minimal cost in terms of time and money on the researcher, depending on their current research process.

Continuous analysis extends continuous integration[43], a common practice in software development and deployment. Continuous integration is a software development workflow that triggers an automated build process whenever developers check their code in to a source control repository. This automated build process runs test scripts if they exist. These tests can catch bugs introduced into software. Software that passes tests is automatically deployed to remote servers.

For continuous analysis (Figure 2), we repurpose these services in order to run computational analyses, update figures, and publish changes to online repositories whenever relevant changes are made to the source code. When an author is ready to release code or publish their work they can export the most recent continuous integration run. Because this process generates results in a clean and clearly defined computing environment without manual intervention, reviewers can be confident that the analyses are reproducible.

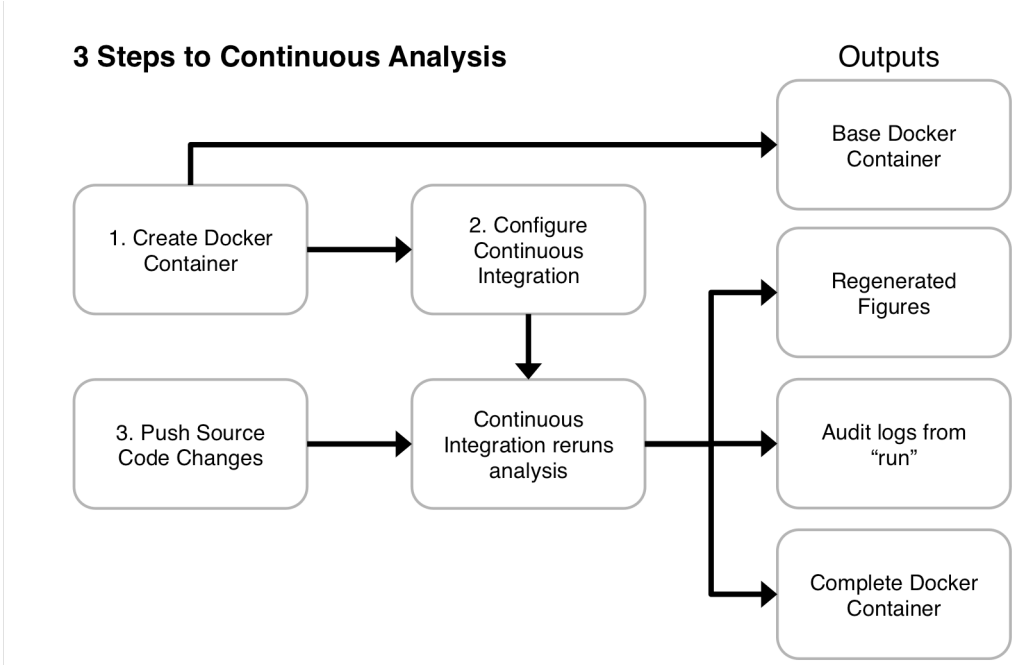

Figure 2. Continuous analysis can be set up in three primary steps (numbered 1, 2, and 3). (1) The researcher creates a Docker container with the required software. (2) The researcher configures a continuous integration service to use this Docker image. (3) The researcher pushes code that includes a script capable of running the analyses from start to finish. 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 everything that occurred. Example configurations are available at our online repository (https://github.com/greenelab/continuous_analysis). Because code is run in an independent, reproducible computing environment and produces detailed logs of what was executed, this practice reduces or eliminates the need for reviewers to re-run code to verify reproducibility.

We maintain dependencies with the free open-source software tool Docker[44] that we used in our differential gene expression analysis example. Docker defines a "container" that allows users to download and run a minimalist virtual machine with a predefined computing environment. The Docker container can be started in a matter of seconds and has minimal overhead[14].

To set up continuous analysis, a researcher needs to do three things. First they must create a Dockerfile, which specifies a list of dependencies. Second, they need to connect a continuous integration service to their version control system and provide the commands to run their analysis. Finally, they need to commit and push changes to their version control system. Many researchers already perform the first and third tasks in their standard workflow.

The continuous integration system 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, e.g. by committing to a specific 'staging' branch. For the first project, this process can be put into place in less than a day. For subsequent projects, this can be done in under an hour.

Setting up Continuous Analysis

We have created a GitHub repository with instructions for paid, local, and cloud-based continuous analysis setups[45]. Here we describe how continuous analysis can be setup using the free and open source Drone software on a researcher's personal computer and connected to the GitHub version control service. This setup is free to users.

1. Install Docker on the computer.
2. Pull the Drone image via docker:

```
sudo docker pull drone/drone:latest
```

3. Create a new application in GitHub (Figure 3).

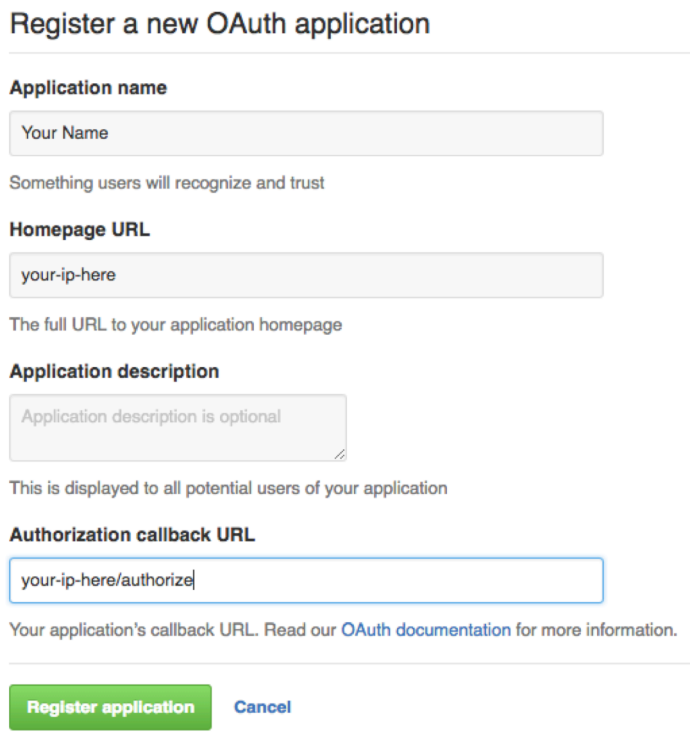

Figure 3. Register a new application for the Drone continuous integration server. Set the homepage URL to be the IP address of the Drone computer. Set the callback URL to the same IP address followed by /authorize.

4. Add a webhook to the GitHub project (Figure 4). This will notify the continuous integration server of any updates pushed to the repository.

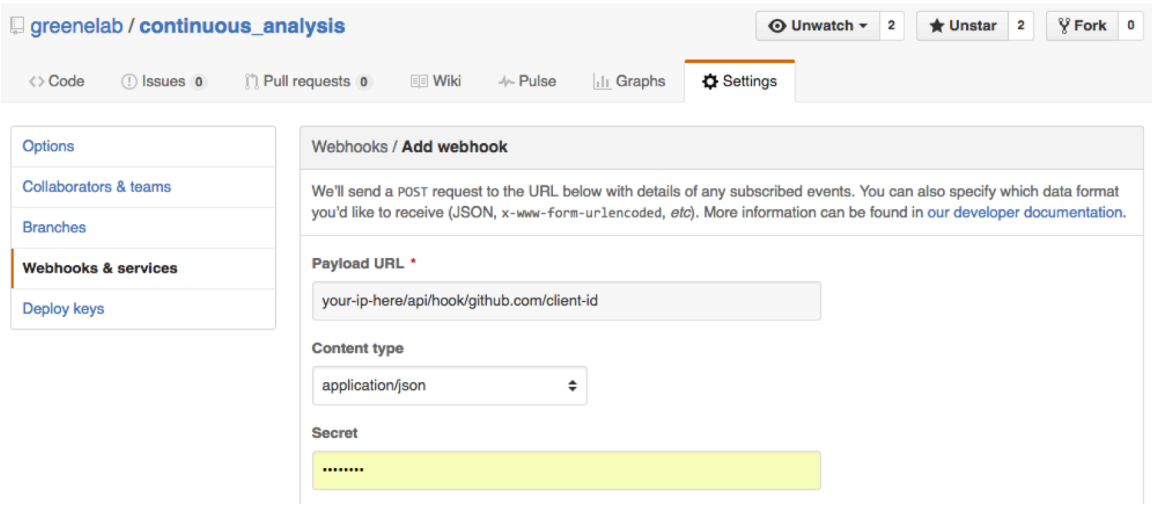

Figure 4. Register a new application for the Drone continuous integration server. The payload URL should be in the format of your-ip/api/hook/github.com/client-id

5. Create a configuration file on the Drone computer at /etc/drone/dronerc filling in the client information provided by GitHub

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

6. Run the drone container

```
sudo docker run drone/drone:latest
```

Continuous analysis can be performed with dozens of full service providers or a private installation on a local machine, cluster or cloud service[45]. Full service providers can be set up in minutes but may have 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 walks of research. With free, open-source continuous integration software[46], computing resources are the only associated costs.

Using Continuous Analysis

After setup, running continuous analysis is simple and fits into existing research workflows that use source control systems. We have used continuous analysis in our own work[47]. We have also prepared an example demonstrating

continuous analysis with kallisto[48]. The recently published software tool kallisto quantifies transcript abundance in RNA-Seq data. Our example re-runs the examples provided in kallisto with each commit to a repository.

1. Add a script file to re-run custom analysis. For Drone, this is a .drone.yml file that specifies commands to run each step of the analysis. An example configuration is available in the continuous_analysis GitHub repository.
2. Commit changes to the source control repository.
3. Push changes to GitHub.

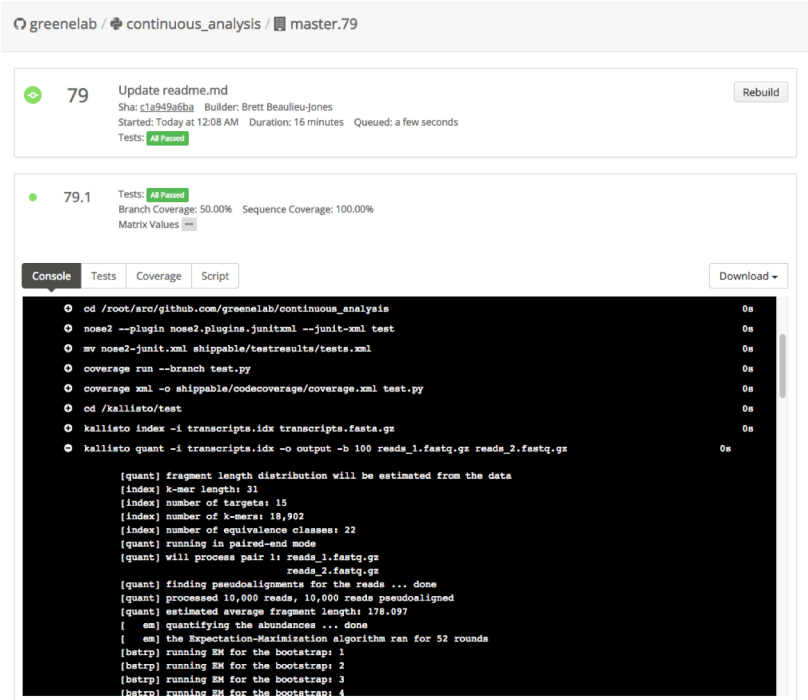

Figure 5. Audit logs from a continuous integration run with the service Shippable for the kallisto example.

The configured continuous integration service automatically runs the specified script. We configured this to rerun the analysis, regenerate the figures, and commit updated versions to the repository. The service provides a complete audit log of what was run in the clean continuous integration environment (Figure 5). By generating and pushing updated figures, this process also generates a complete change log for each result, which we label with the time (Figure 6).

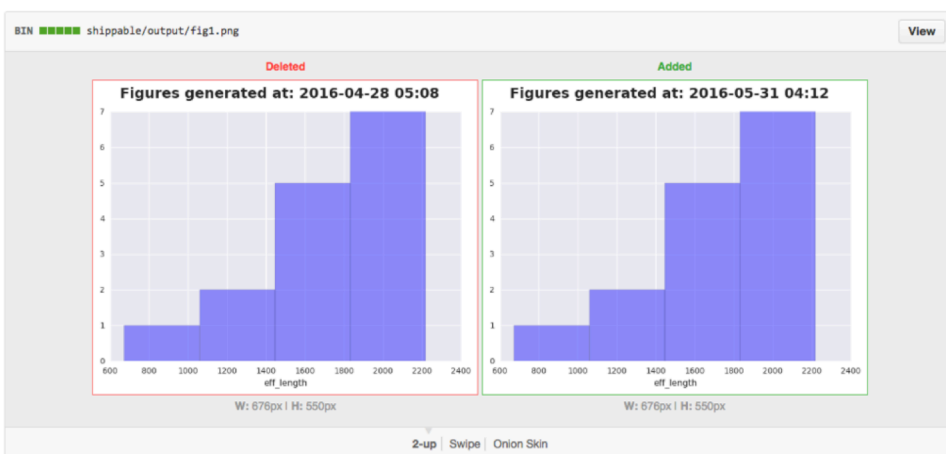

Figure 6. Resulting figures from the run are committed back to Github where changes between runs can be viewed.

In summary, continuous analysis provides the results of a verifiable end-to-end run in a "clean" environment. The audit trail provided by continuous analysis allows reviewers and editors to provide sound judgment on reproducibility without a large time commitment. If readers or reviewers would like to re-run the code on their own (e.g. to change a parameter and evaluate the impact on results), they can easily do so with the Docker container containing the final computing environment and intermediate results.

Continuous analysis provides an audit trail for reproducible analyses of closed data.

Continuous analysis can be even more powerful when working with closed data that cannot be released. Without continuous analysis, reproducing computational analyses based on closed data is dependent on the original authors completely and exactly describing each step, a process that may be an afterthought and relegated to extended methods. Readers must then diligently follow complex written instructions without intermediate confirmation they are on the right track. The containers produced during continuous analysis include a matching environment for replication as well as intermediate results. This allows readers to determine where their results diverge from the original work and to determine whether divergence is due to software-based or data-based differences.

The impact of reproducible computational research

Reproducibility can have wide-reaching benefits for the advancement of science. For authors, easily reproducible work is a sign of quality and credibility. Continuous analysis addresses the reproducibility of computationally analyses in the narrow sense: generating the same results from the same inputs. It does not solve reproducibility in the broader sense: how robust results are to parameter settings, starting conditions and partitions in the data. Continuous analysis lays the groundwork needed to address reproducibility and robustness of findings in the broad sense.

Acknowledgements
The Gordon and Betty Moore Foundation supported this work with a Data Driven Discovery Investigator Award to CSG (GBMF 4552).

References

1. Rebooting review. *Nat Biotech*. 2015;33(4):319. http://dx.doi.org/10.1038/nbt.3202.
2. Software with impact. *Nat Meth*. 2014;11(3):211. http://dx.doi.org/10.1038/nmeth.2880.
3. Peng RD. Reproducible Research in Computational Science. *Science (80-)*. 2011;334(6060):1226-1227. doi:10.1126/science.1213847.

4. McNutt M. Reproducibility. *Science (80-)*. 2014;343(6168):229. http://science.sciencemag.org/content/343/6168/229.abstract.
5. Illuminating the black box. *Nature*. 2006;442(7098):1. http://dx.doi.org/10.1038/442001a.
6. Baker M. 1,500 scientists lift the lid on reproducibility. *Nature*. 2016;533(7604):452-454. doi:10.1038/533452a.
7. Garijo D, Kinnings S, Xie L, et al. Quantifying reproducibility in computational biology: The case of the tuberculosis drugome. *PLoS One*. 2013;8(11). doi:10.1371/journal.pone.0080278.
8. Kinnings SL, Xie L, Fung KH, Jackson RM, Xie L, Bourne PE. The Mycobacterium tuberculosis drugome and its polypharmacological implications. *PLoS Comput Biol*. 2010;6(11). doi:10.1371/journal.pcbi.1000976.
9. Bell AW, Deutsch EW, Au CE, et al. A HUPO test sample study reveals common problems in mass spectrometry-based proteomics. *Nat Methods*. 2009;6(6):423-430. doi:10.1038/nmeth.1333.
10. Ioannidis JPA, Allison DB, Ball CA, et al. Repeatability of published microarray gene expression analyses. *Nat Genet*. 2009;41(2):149-155. doi:10.1038/ng.295.
11. Hothorn T, Leisch F. Case studies in reproducibility. *Brief Bioinform*. 2011;12(3):288-300. doi:10.1093/bib/bbq084.
12. Groves T, Godlee F. Open science and reproducible research. *BMJ*. 2012;344. doi:10.1136/bmj.e4383.
13. ThinkLab. https://thinklab.com/. Accessed January 1, 2016.
14. Boettiger C. An introduction to Docker for reproducible research, with examples from the R environment. *ACM SIGOPS Oper Syst Rev Spec Issue Repeatability Shar Exp Artifacts*. 2015;49(1):71-79. doi:10.1145/2723872.2723882.
15. Dai M, Wang P, Boyd AD, et al. Evolving gene/transcript definitions significantly alter the interpretation of GeneChip data. *Nucleic Acids Res*. 2005;33(20):e175. doi:10.1093/nar/gni179.
16. Kopljar I, Gallacher DJ, De Bondt A, et al. Functional and Transcriptional Characterization of Histone Deacetylase Inhibitor-Mediated Cardiac Adverse Effects in Human Induced Pluripotent Stem Cell-Derived Cardiomyocytes. *Stem Cells Transl Med*. 2016;5(5):602-612. doi:10.5966/sctm.2015-0279.
17. Karpiński P, Frydecka D, Sąsiadek M. Reduced number of peripheral natural killer cells in schizophrenia but not in bipolar disorder. *Brain, Behav*. 2016.
18. Brummelman J, Raeven R, Helm K. Transcriptome signature for dampened Th2 dominance in acellular pertussis vaccine-induced CD4+ T cell responses through TLR4 ligation. *Scientific*. 2016.
19. Bilgrau A, Eriksen P, Rasmussen J. GMCM: Unsupervised clustering and meta-analysis using gaussian mixture copula models. *J Stat*. 2016.
20. Gandin V, Masvidal L, Cargnello M, Gyenis L. mTORC1 and CK2 coordinate ternary and eIF4F complex assembly. *Nature*. 2016.
21. Killeen A, Diskin M, Morris D. Endometrial gene expression in high-and low-fertility heifers in the late luteal phase of the estrous cycle and a comparison

with midluteal gene expression. *Physiological*. 2016.
22. Colletti N, Liu H, Gower A, Alekseyev Y. Tlr3 signaling Promotes the induction of Unique human BDca-3 Dendritic cell Populations. *Front*. 2016.
23. Lee M, Huang R, Tong W. Discovery of transcriptional targets regulated by nuclear receptors using a probabilistic graphical model. *Toxicol Sci*. 2015.
24. Troy N, Hollams E, Holt P. Differential gene network analysis for the identification of asthma-associated therapeutic targets in allergen-specific T-helper memory responses. *BMC Med*. 2016.
25. Manié E, Popova T, Battistella A. Genomic hallmarks of homologous recombination deficiency in invasive breast carcinomas. *J Cancer*. 2016.
26. Dekkers B, He H, Hanson J, Willems L. The Arabidopsis DELAY OF GERMINATION 1 gene affects ABSCISIC ACID INSENSITIVE 5 (ABI5) expression and genetically interacts with ABI3 during Arabidopsis. *The Plant*. 2016.
27. Holt P, Strickland D, Bosco A, Belgrave D. Distinguishing benign from pathologic TH 2 immunity in atopic children. *J Allergy*. 2015.
28. Lück S, Westermark P. Circadian mRNA expression: insights from modeling and transcriptomics. *Cell Mol Life Sci*. 2016.
29. Bosco A, Wiehler S, Proud D. Interferon regulatory factor 7 regulates airway epithelial cell responses to human rhinovirus infection. *BMC Genomics*. 2016.
30. Fauteux F, Hill J, Jaramillo M, Pan Y, Phan S. Computational selection of antibody-drug conjugate targets for breast cancer. *Oncotarget*. 2015.
31. Napolitano F, Sirci F, Carrella D, Bernardo D di. Drug-set enrichment analysis: a novel tool to investigate drug mode of action. *Bioinformatics*. 2016.
32. Carroll J, Meyer C, Song J, Li W, Geistlinger T. Genome-wide analysis of estrogen receptor binding sites. *Nature*. 2006.
33. Lupien M, Eeckhoute J, Meyer C, Wang Q, Zhang Y. FoxA1 translates epigenetic signatures into enhancer-driven lineage-specific transcription. *Cell*. 2008.
34. Wang Q, Li W, Zhang Y, et al. Androgen receptor regulates a distinct transcription program in androgen-independent prostate cancer. *Cell*. 2009.
35. Lefterova M, Zhang Y, Steger D. PPARγ and C/EBP factors orchestrate adipocyte biology via adjacent binding on a genome-wide scale. *Genes &*. 2008.
36. Tuupanen S, Turunen M, Lehtonen R, Hallikas O. The common colorectal cancer predisposition SNP rs6983267 at chromosome 8q24 confers potential to enhanced Wnt signaling. *Nature*. 2009.
37. Obad S, Santos C dos, Petri A, Heidenblad M. Silencing of microRNA families by seed-targeting tiny LNAs. *Nature*. 2011.
38. He H, Meyer C, Shin H, Bailey S, Wei G, Wang Q. Nucleosome dynamics define transcriptional enhancers. *Nature*. 2010.
39. Ozsolak F, Song J, Liu X, Fisher D. High-throughput mapping of the chromatin structure of human promoters. *Nat Biotechnol*. 2007.
40. Zuo T, Wang L, Morrison C, Chang X, Zhang H, Li W. FOXP3 is an X-linked breast cancer suppressor gene and an important repressor of the HER-2/ErbB2 oncogene. *Cell*. 2007.
41. Enard W, Gehre S, Hammerschmidt K, Hölter S. A humanized version of Foxp2

affects cortico-basal ganglia circuits in mice. *Cell*. 2009.
42. Nunez M, Sanchez-Jimenez C, Alcalde J, Izquierdo JM. Long-term reduction of T-cell intracellular antigens reveals a transcriptome associated with extracellular matrix and cell adhesion components. *PLoS One*. 2014;9(11). doi:10.1371/journal.pone.0113141.
43. Duvall P, Matyas S, Glover A. *Continuous Integration: Improving Software Quality and Reducing Risk*.; 2007. http://portal.acm.org/citation.cfm?id=1406212.
44. Docker. Docker. https://www.docker.com.
45. Beaulieu-Jones BK, Greene CS. Continuous Analysis. GitHub repository. https://github.com/greenelab/continuous_analysis. Published 2016.
46. Drone.io. https://drone.io/.
47. Beaulieu-Jones BK. Denoising Autoencoders for Phenotype Stratification (DAPS): Preprint Release. Zenodo. January 2016. doi:10.5281/zenodo.46165.
48. Bray NL, Pimentel H, Melsted P, Pachter L. Near-optimal probabilistic RNA-seq quantification. *Nat Biotechnol*. 2016;34(5):525-527. doi:10.1038/nbt.3519.