Data Science and Advanced Programming — Lecture 3b

Python Start, Git

Simon Scheidegger
Department of Economics, University of Lausanne, Switzerland

September 29th, 2025 | 12:30 - 16:00 | Internef 263

Outline

- Replicable results
- Git repositories

Replicable results

`See e.g. Stodden et. al (2014) --- Implementing Reproducible Research`

Implementing reproducible research

- Computational tools are at the core of modern research.
- In addition to experiments & theory, the notions of **simulation** and **data-intensive discovery** are often referred to as **"third" (and fourth) pillars of science.**
- **For all its importance, computing receives perfunctory attention in training of new scientists and in the conduct of everyday research.**
- It is treated as an inconsequential task that students and researchers learn "on the go" with little consideration for ensuring computational results are trustworthy, comprehensible, and ultimately a secure foundation for **reproducible outcomes.**

Implementing reproducible research

- Computational tools are at the core of modern research.
- In addition to experiments & theory, the notions of **simulation** and **data-intensive discovery** are often referred to as **"third" (and fourth) pillars of science.**
- **For all its importance, computing receives perfunctory attention in training of new scientists and in the conduct of everyday research.**
- It is treated as an inconsequential task that students and researchers learn "on the go" with little consideration for ensuring computational results are trustworthy, comprehensible, and ultimately a secure foundation for **reproducible outcomes.**

Implementing reproducible research

- Computational tools are at the core of modern research.
- In addition to experiments & theory, the notions of **simulation** and **data-intensive discovery** are often referred to as **"third" (and fourth) pillars of science.**
- **For all its importance, computing receives perfunctory attention in training of new scientists and in the conduct of everyday research.**
- It is treated as an inconsequential task that students and researchers learn "on the go" with little consideration for ensuring computational results are trustworthy, comprehensible, and ultimately a secure foundation for **reproducible outcomes.**

Implementing reproducible research II

- Software and data are often stored with poor organization.
- Little documentation.
- Few tests.
- **Haphazard patchwork of software tools** is used with limited attention paid to capturing the complex work flows that emerge.
- **The evolution of code is not tracked over time**, making it difficult to understand what iteration of the code was used to obtain any specific result.
- Many of the software packages used by scientists in research are proprietary and closed source, preventing complete understanding and control of the final scientific results.
- **One node hour on a HPC system costs around** > 1 **CHF**

Implementing reproducible research II

- Software and data are often stored with poor organization.
- Little documentation.
- Few tests.
- **Haphazard patchwork of software tools** is used with limited attention paid to capturing the complex work flows that emerge.
- **The evolution of code is not tracked over time**, making it difficult to understand what iteration of the code was used to obtain any specific result.
- Many of the software packages used by scientists in research are proprietary and closed source, preventing complete understanding and control of the final scientific results.
- **One node hour on a HPC system costs around** > 1 **CHF**

Implementing reproducible research II

- Software and data are often stored with poor organization.
- Little documentation.
- Few tests.
- **Haphazard patchwork of software tools** is used with limited attention paid to capturing the complex work flows that emerge.
- **The evolution of code is not tracked over time**, making it difficult to understand what iteration of the code was used to obtain any specific result.
- Many of the software packages used by scientists in research are proprietary and closed source, preventing complete understanding and control of the final scientific results.
- **One node hour on a HPC system costs around** > 1 **CHF**

Implementing reproducible research II

- Software and data are often stored with poor organization.
- Little documentation.
- Few tests.
- **Haphazard patchwork of software tools** is used with limited attention paid to capturing the complex work flows that emerge.
- **The evolution of code is not tracked over time**, making it difficult to understand what iteration of the code was used to obtain any specific result.
- Many of the software packages used by scientists in research are proprietary and closed source, preventing complete understanding and control of the final scientific results.
- **One node hour on a HPC system costs around > 1 CHF**

Implementing reproducible research II

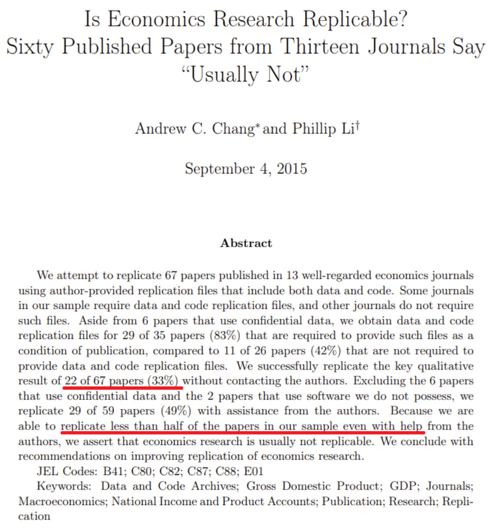
- Software and data are often stored with poor organization.
- Little documentation.
- Few tests.
- **Haphazard patchwork of software tools** is used with limited attention paid to capturing the complex work flows that emerge.
- **The evolution of code is not tracked over time**, making it difficult to understand what iteration of the code was used to obtain any specific result.
- Many of the software packages used by scientists in research are proprietary and closed source, preventing complete understanding and control of the final scientific results.
- **One node hour on a HPC system costs around** > 1 **CHF**

Implementing reproducible research II

- Software and data are often stored with poor organization.
- Little documentation.
- Few tests.
- **Haphazard patchwork of software tools** is used with limited attention paid to capturing the complex work flows that emerge.
- **The evolution of code is not tracked over time**, making it difficult to understand what iteration of the code was used to obtain any specific result.
- Many of the software packages used by scientists in research are proprietary and closed source, preventing complete understanding and control of the final scientific results.
- **One node hour on a HPC system costs around > 1 CHF**

Result — status of the community

http://www.federalreserve.gov/econresdata/feds/2015/files/2015083pap.pdf



Is Economics Research Replicable?
Sixty Published Papers from Thirteen Journals Say
"Usually Not"

Andrew C. Chang[*] and Phillip Li[†]

September 4, 2015

Abstract

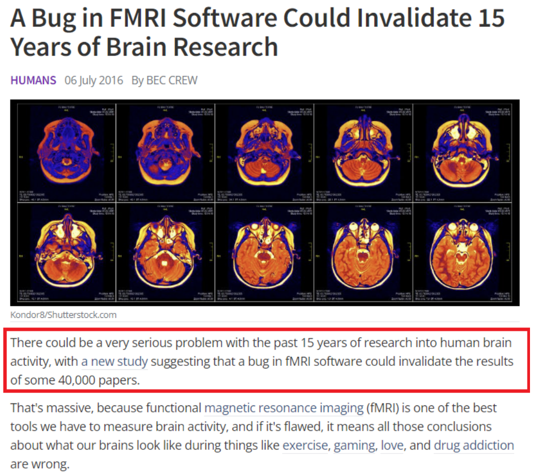
We attempt to replicate 67 papers published in 13 well-regarded economics journals using author-provided replication files that include both data and code. Some journals in our sample require data and code replication files, and other journals do not require such files. Aside from 6 papers that use confidential data, we obtain data and code replication files for 29 of 35 papers (83%) that are required to provide such files as a condition of publication, compared to 11 of 26 papers (42%) that are not required to provide data and code replication files. We successfully replicate the key qualitative result of 22 of 67 papers (33%) without contacting the authors. Excluding the 6 papers that use confidential data and the 2 papers that use software we do not possess, we replicate 29 of 59 papers (49%) with assistance from the authors. Because we are able to replicate less than half of the papers in our sample even with help from the authors, we assert that economics research is usually not replicable. We conclude with recommendations on improving replication of economics research.

JEL Codes: B41; C80; C82; C87; C88; E01

Keywords: Data and Code Archives; Gross Domestic Product; GDP; Journals; Macroeconomics; National Income and Product Accounts; Publication; Research; Replication

Flaws in other fields

http://www.pnas.org/content/113/28/7900.abstract



A Bug in FMRI Software Could Invalidate 15 Years of Brain Research

HUMANS 06 July 2016 By BEC CREW

Kondor8/Shutterstock.com

There could be a very serious problem with the past 15 years of research into human brain activity, with a new study suggesting that a bug in fMRI software could invalidate the results of some 40,000 papers.

That's massive, because functional magnetic resonance imaging (fMRI) is one of the best tools we have to measure brain activity, and if it's flawed, it means all those conclusions about what our brains look like during things like exercise, gaming, love, and drug addiction are wrong.

Common computational research workflow

- Matlab, Python etc. for prototyping.
- Developing high-performance code in C++, ...
- Postprocessing (e.g. Matlab - manually twiddling data & Figs.)
- → What if a couple of months later the researcher realizes there is a problem with the results?Will he be able to remember what buttons he clicked to reproduce the work flow to generate updated plots, paper manuscripts, etc.
- → Publish-or-perishmindset encourages to charge forward chasing the goal of an accepted manuscript.
- → “reproducibility” implies repetition and thus the requirement also to move back, to retrace one’s steps.

Common computational research workflow

- Matlab, Python etc. for prototyping.
- Developing high-performance code in C++, ...
- Postprocessing (e.g. Matlab - manually twiddling data & Figs.)
- $\rightarrow$ What if a couple of months later the researcher realizes there is a problem with the results?Will he be able to remember what buttons he clicked to reproduce the work flow to generate updated plots, paper manuscripts, etc.
- $\rightarrow$ Publish-or-perishmindset encourages to charge forward chasing the goal of an accepted manuscript.
- $\rightarrow$ “reproducibility” implies repetition and thus the requirement also to move back, to retrace one’s steps.

Computational research life cycle

Open source community has cultivated tools and practices that, if embraced and adapted by the scientific community, will greatly enhance the ability to achieve reproducible outcomes.

- Individual exploration: a single investigator tests an idea, algorithm, or question, likely with a small-scale test, dataset, or simulation.
- Collaboration: if the initial exploration appears promising, more often than not some kind of collaborative effort ensues to bring together complementary expertise from colleagues.
- Production-scale execution: large datasets and complex simulations often require the use of clusters, supercomputers, or cloud resources in parallel.
- Publication: whether as a paper or an internal report for discussion with colleagues, results need to be presented to others in a coherent form.
- Education: ultimately, research results become part of the corpus of a discipline that is shared with students and colleagues, thus seeding the next iteration in the cycle of research.

list from Millman and Pérez (2014)

Individual Work

- For individual work, researchers use various interactive computing environments: Excel, Python, MATLAB, Mathematica, R,...
- These environments combine interactive, high-level programming languages with a rich set of numerical and visualization libraries. The impact of these environments cannot be overstated; researchers use them for rapid prototyping, interactive exploration, and data analysis, as well as visualization.
- However, they have limitations:
  - some of them are proprietary and/or expensive (Excel, MATLAB, Mathematica)
  - most are focused on coding in a single, relatively slow, programming language.
- While the use of proprietary tools is not a problem per se and may be a good solution in industry, it is a barrier to scientific collaboration and to the construction of a common scientific heritage where anyone can validate the work of others and build upon it.
- Scientists cannot share work unless all colleagues can purchase the same package.
- Students are forced to work with black boxes they are legally prevented from inspecting. Furthermore, because of their limitations in performance and handling large, complex code bases, these tools are mostly used for prototyping: researchers eventually have to switch tools for building production systems.

Individual Work

- For individual work, researchers use various interactive computing environments: Excel, Python, MATLAB, Mathematica, R,...
- These environments combine interactive, high-level programming languages with a rich set of numerical and visualization libraries. The impact of these environments cannot be overstated; researchers use them for rapid prototyping, interactive exploration, and data analysis, as well as visualization.
- However, they have limitations:
  - some of them are proprietary and/or expensive (Excel, MATLAB, Mathematica);
  - most are focused on coding in a single, relatively slow, programming language.
- While the use of proprietary tools is not a problem per se and may be a good solution in industry, it is a barrier to scientific collaboration and to the construction of a common scientific heritage where anyone can validate the work of others and build upon it.
- Scientists cannot share work unless all colleagues can purchase the same package.
- Students are forced to work with black boxes they are legally prevented from inspecting. Furthermore, because of their limitations in performance and handling large, complex code bases, these tools are mostly used for prototyping: researchers eventually have to switch tools for building production systems.

Individual Work

- For individual work, researchers use various interactive computing environments: Excel, Python, MATLAB, Mathematica, R,...
- These environments combine interactive, high-level programming languages with a rich set of numerical and visualization libraries. The impact of these environments cannot be overstated; researchers use them for rapid prototyping, interactive exploration, and data analysis, as well as visualization.
- However, they have limitations:
  - some of them are proprietary and/or expensive (Excel, MATLAB, Mathematica);
  - most are focused on coding in a single, relatively slow, programming language.
- While the use of proprietary tools is not a problem per se and may be a good solution in industry, it is a barrier to scientific collaboration and to the construction of a common scientific heritage where anyone can validate the work of others and build upon it.
- Scientists cannot share work unless all colleagues can purchase the same package.
- Students are forced to work with black boxes they are legally prevented from inspecting. Furthermore, because of their limitations in performance and handling large, complex code bases, these tools are mostly used for prototyping: researchers eventually have to switch tools for building production systems.

Individual Work

- For individual work, researchers use various interactive computing environments: Excel, Python, MATLAB, Mathematica, R,...
- These environments combine interactive, high-level programming languages with a rich set of numerical and visualization libraries. The impact of these environments cannot be overstated; researchers use them for rapid prototyping, interactive exploration, and data analysis, as well as visualization.
- However, they have limitations:
  - some of them are proprietary and/or expensive (Excel, MATLAB, Mathematica);
  - most are focused on coding in a single, relatively slow, programming language.
- While the use of proprietary tools is not a problem per se and may be a good solution in industry, it is a barrier to scientific collaboration and to the construction of a common scientific heritage where anyone can validate the work of others and build upon it.
- Scientists cannot share work unless all colleagues can purchase the same package.
- Students are forced to work with black boxes they are legally prevented from inspecting. Furthermore, because of their limitations in performance and handling large, complex code bases, these tools are mostly used for prototyping: researchers eventually have to switch tools for building production systems.

Individual Work

- For individual work, researchers use various interactive computing environments: Excel, Python, MATLAB, Mathematica, R,...
- These environments combine interactive, high-level programming languages with a rich set of numerical and visualization libraries. The impact of these environments cannot be overstated; researchers use them for rapid prototyping, interactive exploration, and data analysis, as well as visualization.
- However, they have limitations:
  - some of them are proprietary and/or expensive (Excel, MATLAB, Mathematica);
  - most are focused on coding in a single, relatively slow, programming language.
- While the use of proprietary tools is not a problem per se and may be a good solution in industry, it is a barrier to scientific collaboration and to the construction of a common scientific heritage where anyone can validate the work of others and build upon it.
- Scientists cannot share work unless all colleagues can purchase the same package.
- Students are forced to work with black boxes they are legally prevented from inspecting. Furthermore, because of their limitations in performance and handling large, complex code bases, these tools are mostly used for prototyping: researchers eventually have to switch tools for building production systems.

Collaboration

- For collaboration, researchers tend to use a mix of e-mail, version control systems (**VCSs**) and shared network folders (Dropbox, etc.).
- **VCSs are critically important** in making research collaborative and reproducible.
- They allow groups to work collaboratively on documents and track how they evolve over time. Ideally, all aspects of computational research would be hosted on publicly available version control repositories, such as GitHub, Bitbucket or Google Code.
- Unfortunately, the common approach is for researchers to e-mail documents to each other with ad hoc naming conventions that provide a poor man's version control.
- While a small group can make it work, this approach most certainly does not scale beyond a few collaborators, as painfully experienced by anyone who has participated in the madness of a flurry of e-mail attachments with oddly named files such as `paper-final-v2-REALLY-FINAL-john-OCT9.doc`.

Collaboration

- For collaboration, researchers tend to use a mix of e-mail, version control systems (**VCSs**) and shared network folders (Dropbox, etc.).
- **VCSs are critically important** in making research collaborative and reproducible.
- They allow groups to work collaboratively on documents and track how they evolve over time. Ideally, all aspects of computational research would be hosted on publicly available version control repositories, such as GitHub, Bitbucket or Google Code.
- Unfortunately, the common approach is for researchers to e-mail documents to each other with ad hoc naming conventions that provide a poor man's version control.
- While a small group can make it work, this approach most certainly does not scale beyond a few collaborators, as painfully experienced by anyone who has participated in the madness of a flurry of e-mail attachments with oddly named files such as `paper-final-v2-REALLY-FINAL-john-OCT9.doc`.

Collaboration

- For collaboration, researchers tend to use a mix of e-mail, version control systems (**VCSs**) and shared network folders (Dropbox, etc.).
- **VCSs are critically important** in making research collaborative and reproducible.
- They allow groups to work collaboratively on documents and track how they evolve over time. Ideally, all aspects of computational research would be hosted on publicly available version control repositories, such as GitHub, Bitbucket or Google Code.
- Unfortunately, the common approach is for researchers to e-mail documents to each other with ad hoc naming conventions that provide a poor man's version control.
- While a small group can make it work, this approach most certainly does not scale beyond a few collaborators, as painfully experienced by anyone who has participated in the madness of a flurry of e-mail attachments with oddly named files such as `paper-final-v2-REALLY-FINAL-john-OCT9.doc`.

Collaboration

- For collaboration, researchers tend to use a mix of e-mail, version control systems (**VCSs**) and shared network folders (Dropbox, etc.).
- **VCSs are critically important** in making research collaborative and reproducible.
- They allow groups to work collaboratively on documents and track how they evolve over time. Ideally, all aspects of computational research would be hosted on publicly available version control repositories, such as GitHub, Bitbucket or Google Code.
- Unfortunately, the common approach is for researchers to e-mail documents to each other with ad hoc naming conventions that provide a poor man's version control.
- While a small group can make it work, this approach most certainly does not scale beyond a few collaborators, as painfully experienced by anyone who has participated in the madness of a flurry of e-mail attachments with oddly named files such as `paper-final-v2-REALLY-FINAL-john-OCT9.doc`.

Collaboration

- For collaboration, researchers tend to use a mix of e-mail, version control systems (**VCSs**) and shared network folders (Dropbox, etc.).
- **VCSs are critically important** in making research collaborative and reproducible.
- They allow groups to work collaboratively on documents and track how they evolve over time. Ideally, all aspects of computational research would be hosted on publicly available version control repositories, such as GitHub, Bitbucket or Google Code.
- Unfortunately, the common approach is for researchers to e-mail documents to each other with ad hoc naming conventions that provide a poor man's version control.
- While a small group can make it work, this approach most certainly does not scale beyond a few collaborators, as painfully experienced by anyone who has participated in the madness of a flurry of e-mail attachments with oddly named files such as `paper-final-v2-REALLY-FINAL-john-OCT9.doc`.

Production-scale execution

- For production-scale execution, researchers typically turn away from the convenience of interactive computing environments to compiled code (C,C++, Fortran) and libraries for distributed and parallel processing.
- These tools are specialized enough that their mastery requires a substantial investment of time. We emphasize, that before production-scale computations begin, the researchers already have a working prototype in an interactive computing environment.
- Therefore, turning to new parallel tools means starting over and maintaining at least two versions of the code moving forward. Furthermore, data produced by the compiled version are often imported back into the interactive environment for visualization and analysis.
- The resulting back-and-forth work-flow is nearly impossible to capture and put into VCSs, making the computational research difficult to reproduce.
- Obviously the alternative, taken by many, issimply to run the slow serial code for as long as it takes. This is hardly a solution to the reproducibility problem, as run times in the weeks or months become in practice single-shot efforts that no one will replicate.

Production-scale execution

- For production-scale execution, researchers typically turn away from the convenience of interactive computing environments to compiled code (C,C++, Fortran) and libraries for distributed and parallel processing.
- These tools are specialized enough that their mastery requires a substantial investment of time. We emphasize, that before production-scale computations begin, the researchers already have a working prototype in an interactive computing environment.
- Therefore, turning to new parallel tools means starting over and maintaining at least two versions of the code moving forward. Furthermore, data produced by the compiled version are often imported back into the interactive environment for visualization and analysis.
- The resulting back-and-forth work-flow is nearly impossible to capture and put into VCSs, making the computational research difficult to reproduce.
- Obviously the alternative, taken by many, issimply to run the slow serial code for as long as it takes. This is hardly a solution to the reproducibility problem, as run times in the weeks or months become in practice single-shot efforts that no one will replicate.

Production-scale execution

- For production-scale execution, researchers typically turn away from the convenience of interactive computing environments to compiled code (C,C++, Fortran) and libraries for distributed and parallel processing.
- These tools are specialized enough that their mastery requires a substantial investment of time. We emphasize, that before production-scale computations begin, the researchers already have a working prototype in an interactive computing environment.
- Therefore, turning to new parallel tools means starting over and maintaining at least two versions of the code moving forward. Furthermore, data produced by the compiled version are often imported back into the interactive environment for visualization and analysis.
- The resulting back-and-forth work-flow is nearly impossible to capture and put into VCSs, making the computational research difficult to reproduce.
- Obviously the alternative, taken by many, issimply to run the slow serial code for as long as it takes. This is hardly a solution to the reproducibility problem, as run times in the weeks or months become in practice single-shot efforts that no one will replicate.

Production-scale execution

- For production-scale execution, researchers typically turn away from the convenience of interactive computing environments to compiled code (C,C++, Fortran) and libraries for distributed and parallel processing.
- These tools are specialized enough that their mastery requires a substantial investment of time. We emphasize, that before production-scale computations begin, the researchers already have a working prototype in an interactive computing environment.
- Therefore, turning to new parallel tools means starting over and maintaining at least two versions of the code moving forward. Furthermore, data produced by the compiled version are often imported back into the interactive environment for visualization and analysis.
- The resulting back-and-forth work-flow is nearly impossible to capture and put into VCSs, making the computational research difficult to reproduce.
- Obviously the alternative, taken by many, issimply to run the slow serial code for as long as it takes. This is hardly a solution to the reproducibility problem, as run times in the weeks or months become in practice single-shot efforts that no one will replicate.

Production-scale execution

- For production-scale execution, researchers typically turn away from the convenience of interactive computing environments to compiled code (C,C++, Fortran) and libraries for distributed and parallel processing.
- These tools are specialized enough that their mastery requires a substantial investment of time. We emphasize, that before production-scale computations begin, the researchers already have a working prototype in an interactive computing environment.
- Therefore, turning to new parallel tools means starting over and maintaining at least two versions of the code moving forward. Furthermore, data produced by the compiled version are often imported back into the interactive environment for visualization and analysis.
- The resulting back-and-forth work-flow is nearly impossible to capture and put into VCSs, making the computational research difficult to reproduce.
- Obviously the alternative, taken by many, issimply to run the slow serial code for as long as it takes. This is hardly a solution to the reproducibility problem, as run times in the weeks or months become in practice single-shot efforts that no one will replicate.

Punclication & Education

- For publications and education, researchers use tools such as LaTeX, Google Docs, or Microsoft Word, PowerPoint,...
- The most important attribute of these tools in this context is that, LaTeX excepted, they integrate poorly with VCSs and are ill-suited for work flow automation.
- Digital artefacts (code, data, and visualizations) are often manually pasted into these documents, which easily leads to a divergence between the computational outcomes and the publication.
- The lack of automated integration requires manual updating, something that is errorprone and easy to forget.

Lessons learned out of those issues

- The common approaches and tools used today introduce discontinuities between the different stages of the scientific work flow. Forcing researchers to switch tools at each stage, which in turn makes it difficult to move fluidly back and forth.
- A key element of the problem is the gap that exists between what we view as "final outcomes" of the scientific effort (papers and presentations that contain artefacts such as figures, tables, and other outcomes of the computation) and the pipeline that feeds these outcomes. Because most work flows involve a **manual transfer of information** (often with unrecorded changes along the way), the chances that these final outcomes match what the computational pipeline actually produces at any given time are low.
- The problems listed earlier are both **technical and social.** It is critical to understand that at the end of the day, only when researchers make a conscious decision to adopt improved work habits will we see substantial improvements on this problem. Obviously, higher-quality tools will make it easier and more appealing to adopt such changes; but other factors — from the inertia of ingrained habits to the pressure applied by the incentive models of modern research — are also at play.

Lessons learned out of those issues

- The common approaches and tools used today introduce discontinuities between the different stages of the scientific work flow. Forcing researchers to switch tools at each stage, which in turn makes it difficult to move fluidly back and forth.
- A key element of the problem is the gap that exists between what we view as "final outcomes" of the scientific effort (papers and presentations that contain artefacts such as figures, tables, and other outcomes of the computation) and the pipeline that feeds these outcomes. Because most work flows involve a **manual transfer of information** (often with unrecorded changes along the way), the chances that these final outcomes match what the computational pipeline actually produces at any given time are low.
- The problems listed earlier are both **technical and social.** It is critical to understand that at the end of the day, only when researchers make a conscious decision to adopt improved work habits will we see substantial improvements on this problem. Obviously, higher-quality tools will make it easier and more appealing to adopt such changes; but other factors — from the inertia of ingrained habits to the pressure applied by the incentive models of modern research — are also at play.

Lessons learned out of those issues

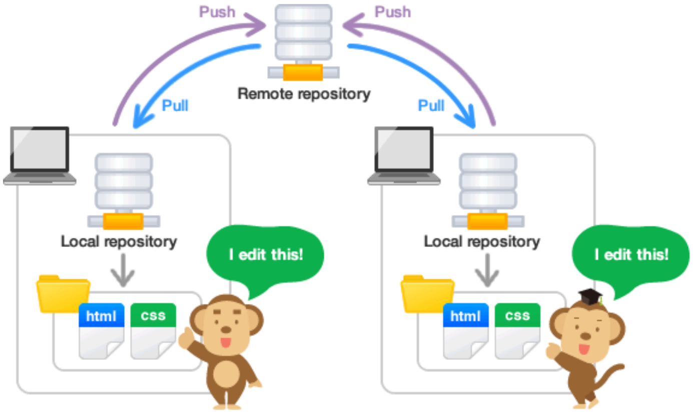
- The common approaches and tools used today introduce discontinuities between the different stages of the scientific work flow. Forcing researchers to switch tools at each stage, which in turn makes it difficult to move fluidly back and forth.
- A key element of the problem is the gap that exists between what we view as "final outcomes" of the scientific effort (papers and presentations that contain artefacts such as figures, tables, and other outcomes of the computation) and the pipeline that feeds these outcomes. Because most work flows involve a **manual transfer of information** (often with unrecorded changes along the way), the chances that these final outcomes match what the computational pipeline actually produces at any given time are low.
- The problems listed earlier are both **technical and social.** It is critical to understand that at the end of the day, only when researchers make a conscious decision to adopt improved work habits will we see substantial improvements on this problem. Obviously, higher-quality tools will make it easier and more appealing to adopt such changes; but other factors — from the inertia of ingrained habits to the pressure applied by the incentive models of modern research — are also at play.

Simple possible remedies

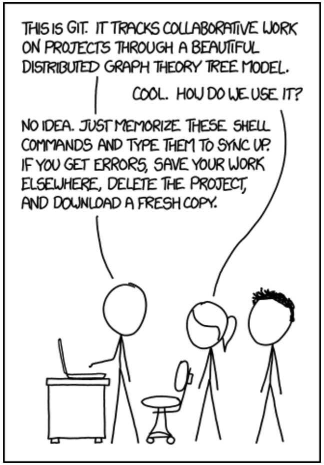
- $\rightarrow$ use open source tools (I don't claim to make all code open source).
- $\rightarrow$ Version control with clear commit messages.
- $\rightarrow$ Execution automation (use makefiles, scripts to plot,...).
- $\rightarrow$ Testing (including unit tests).
- $\rightarrow$ Readability (well commented framework - write README,...)

Git repositories

- The official website: https://git-scm.com/
- Free git book: https://
//progit2.s3.amazonaws.com/en/2016-03-22-f3531/progit-en.1084.pdf
I put it into `/materials`



How I use git :)


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

https://xkcd.com/1597/

Git overview

- As you develop software and make changes, add features, fix bugs, etc. it is often useful to have a mechanism to keep track of changes and to ensure that your code base and artefacts are well-protected by being stored on a reliable server (or multiple servers).
- This allows you access to historic versions of your application's code in case something breaks or to "roll-back" to a previous version if a critical bug is found.
- The solution is to use a revision control system that allows you to "check-in" changes to a code base.
- It keeps track of all changes and allows you to "branch" a code base into a separate copy so that you can develop features or enhancements in isolation of the main code base (often called the "trunk" in keeping with the tree metaphor).
- Once a branch is completed (and well-tested and reviewed), it can then be merged back into the main trunk and it becomes part of the project.

Git overview

- As you develop software and make changes, add features, fix bugs, etc. it is often useful to have a mechanism to keep track of changes and to ensure that your code base and artefacts are well-protected by being stored on a reliable server (or multiple servers).
- This allows you access to historic versions of your application's code in case something breaks or to "roll-back" to a previous version if a critical bug is found.
- The solution is to use a revision control system that allows you to "check-in" changes to a code base.
- It keeps track of all changes and allows you to "branch" a code base into a separate copy so that you can develop features or enhancements in isolation of the main code base (often called the "trunk" in keeping with the tree metaphor).
- Once a branch is completed (and well-tested and reviewed), it can then be merged back into the main trunk and it becomes part of the project.

Git overview

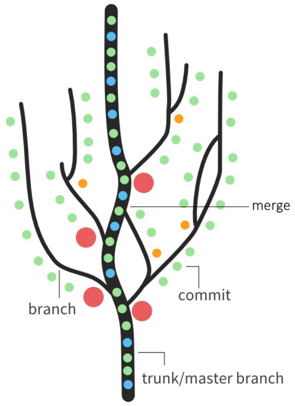
- As you develop software and make changes, add features, fix bugs, etc. it is often useful to have a mechanism to keep track of changes and to ensure that your code base and artefacts are well-protected by being stored on a reliable server (or multiple servers).
- This allows you access to historic versions of your application's code in case something breaks or to "roll-back" to a previous version if a critical bug is found.
- The solution is to use a revision control system that allows you to "check-in" changes to a code base.
- It keeps track of all changes and allows you to "branch" a code base into a separate copy so that you can develop features or enhancements in isolation of the main code base (often called the "trunk" in keeping with the tree metaphor).
- Once a branch is completed (and well-tested and reviewed), it can then be merged back into the main trunk and it becomes part of the project.

Git overview

- As you develop software and make changes, add features, fix bugs, etc. it is often useful to have a mechanism to keep track of changes and to ensure that your code base and artefacts are well-protected by being stored on a reliable server (or multiple servers).
- This allows you access to historic versions of your application's code in case something breaks or to "roll-back" to a previous version if a critical bug is found.
- The solution is to use a revision control system that allows you to "check-in" changes to a code base.
- It keeps track of all changes and allows you to "branch" a code base into a separate copy so that you can develop features or enhancements in isolation of the main code base (often called the "trunk" in keeping with the tree metaphor).
- Once a branch is completed (and well-tested and reviewed), it can then be merged back into the main trunk and it becomes part of the project.

Version control system

- Git essentially keeps track of all changes made to a project and allows users to work in large teams on very complex projects while minimizing the conflicts between changes.
- These systems are not only used for organizational and backup purposes, but are absolutely essential when developing software as part of a team.
- Each team member can have their own working copy of the project code without interfering with other developer's copies or the main trunk.
- Only when separate branches have to be merged into the trunk do conflicting changes have to be addressed.
- Otherwise, such a system allows multiple developers to work on a very complex project in an organized manner.



Git — decentralized

- Git is a decentralized system.
- Multiple servers can act as repositories, but each copy on each developer's own machine is also a complete revision copy.
- Code commits are committed to the local repository.
- Merging a branch into another requires a push/pull request.
- Decentralizing the system means that anyone's machine can act as a code repository and can lead to wider collaboration and independence since different parties are no longer dependent on one master repository.
- Git itself is a version control system that can be installed on any server.

Getting Access to a Repository

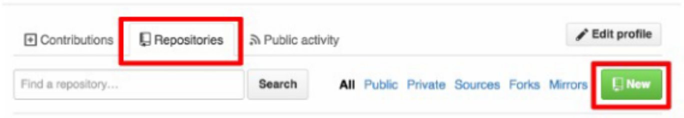
Various commercial providers:

- `https://github.com`
- `https://bitbucket.org`
- `https://gitlab.org`

Be careful: Some repositories are public (versus private) $\rightarrow$ various user choices (10CHF $\rightarrow$ more functionality).

Creating a repository on github

- You will eventually want to publish ("push") your project code to Github.
- To do this you'll first need tocreate a repository on Github's site.
  1. Login to Github (https://github.com/) and click on the "repositories" tab.
  2. Create a new repository with the name that will match your project folder (the names do not have to match, but it keeps things organized). Provide a short description and choose whether or not to make it **public** or **private** depending on whether or not you are allowed to share your code with your Peers.
- You may choose to include a README file and/or establish a license (which creates a LICENSE file). However, for this lecture we will assume that you start with an empty repository on github. If you choose to create these files some extra steps may be necessary.



Cloning an Existing Repository

- There are a variety of WYSIWYG tools. We focus here on the command line interface utilities (What You See Is What You Get).
- **Advantage**: quick, straightforward access to git.
- **Disadvantage**: requires good working knowledge of the command line; proficiency takes longer.
-
- Which version:
  - `git --version` which may output something like git version 2.7.4
- Clone a repository:
  - $\rightarrow$ Move to the directory where you want the project files to be placed.
  - $\rightarrow$ Execute the following command:
  - `git clone https://github.com/project/url`
  - e.g. `git clone https://github.com/sischei/global_solution_yale19`
- $\rightarrow$ now you can start to work with/editing the files of the project.

Creating & sharing your own project I

- Before continuing we need to create a repository on Github, as mentioned earlier
- Setup your local repository from the command line by going to your Project directory and execute the following commands (approximate outputs have been included below).
- Initialize your directory by using: `git init`
- This should result in an output like `Initialized empty Git repository in/your/directory/foo/.git/`
- Add all files, directories and subdirectories to your `git index` using `git add --all`
- Commit your files using the command here. The `-m` specified a commit message follows.

  Output should resemble:

```
[master (root-commit) 7a3fb99] Initial Commit
2 files changed, 24 insertions(+)
create mode 100644 README.md
create mode 100755 hello.c
```

Creating & sharing your own project II

- Associate your repository with the repo on GitHub using the following command: `git remote add origin https://github.com/login/PROJECT.git`, where the URL is replaced with the URL for your project.
- Push your commit to the remote repository using the following command: `git push -u origin master`
- Output should resemble something like:

```
Counting objects: 4 , done.
Delta compression using up to 8 threads.
Compressing objects: 100%(4 / 4), done.
Writing objects: 100%(4 / 4), 577 bytes | 10 bytes/s, done.
Total 4 (delta 0 ), reused 0 (delta 0 )
To https://github.com/login/PROJECT. git
* [new branch] master -> master

Branch master set up to track remote branch master from origin.
```

Committing & Pushing Changes

- Now that your code is committed to Github's servers, you'll eventually want to make changes to current files and/or add/remove files and commit these changes.
- Once you have made your changes, you can essentially repeat part of the process above:

```
git add --all
git commit -m "Update Message"
git push -u origin master
```

- The "Update Message" should be more descriptive: it is used to document the changes you've made for this commit. It is best practice to be as descriptive as possible as to your changes.
- The `git add --all` command adds all files in the current directory as well as all of its subdirectories to the commit index. If you want to be more precise and intentional, you can add individual files using `git add foo.txt`, etc.

Collaboration on a project

- You can then grant read/write access to your others by making them collaborators on the project. You can easily do this in Github by following the instructions at this link: https://help.github.com/articles/adding-collaborators-to-a-personal-repository/
- Once you've all been added, everyone should be able to push/pull from the same repository.
- Git has many more options like branching etc. $\rightarrow$ check the man pages...
- Git Reference: http://gitref.org/
- Git Glossary: https://help.github.com/articles/github-glossary/
- GitHub's walkthrough for both Windows and Mac: https://help.github.com/articles/set-up-git/ https://services.github.com/kit/downloads/github-git-cheat-sheet.pdf

Example “hello world”

- Let’s create a new directory, `~/tmp/test1`, for our first git project.

```
cd
mkdir tmp
cd tmp
mkdir test1
cd test1
```

- Put the directory under git revision control:

```
git init
```

- Let’s start our programming project. Write `hello.py` with your editor:

```
print("hello world")
```

Example “hello world” (2)

- Let’s see what git thinks about what we’re doing: `git status`
- The git status command reports that hello.py is “Untracked.”
- We can have git track `hello.py` by adding it to the “staging” area (more on this later): `git add hello.c` (or `git stage`, which is simply an alias — has the same functionality)
- Run `git status` again. It now reports that `hello.py` is a new file to be Committed.
- Let’s commit it: `git commit`
- Git opens up your editor for you to type a commit message. A commit message should describe what you’re committing in the first line. If you have more to say, Follow the first line with a blank line, and then with a more through multi-line description.
- For now, type in the following one-line commit message, save, and exit the editor: `Added hello-world program.`
- Run `git status` again. When git says nothing about a file, it means that it is being tracked, and that it has not changed since it has been last committed.

Example “hello world” (3)

- Modify hello.py to print “bye world” instead, and run `git status`.
- It reports that the file is “Changed but not updated.”
- This means that the file has been modified since the last commit, but it is still not ready to be committed because it has not been moved into the staging area.
- In git, a file must first go to the staging area before it can be committed.
- Before we move it to the staging area, let’s see what we changed in the file: `git diff`
- The output should tell you that you took out the “hello world'' line, and added a “bye world” line, like this:

```
-print("hello world")
+-print("bye world")
```

- We move the file to the staging area with the `git add` command: `git add hello.py`

Example “hello world” (3)

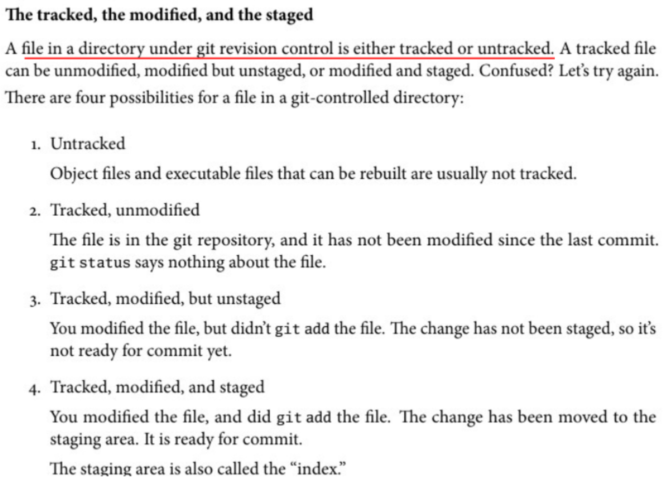
- In git, “add” or “stage” means this: move the change you made to the staging area. The change could be a modification to a tracked file, or it could be a creation of a brand new file.
- This is a point of confusion for those of you who are familiar with other version control systems such as “subversion”.
- At this point, git diff will report no change. Our change—from hello to bye-has been moved into staging already. So this means that git diff reports the difference between the staging area and the working copy of the file.
- To see the difference between the last commit and the staging area, add `--cached` option: `git diff --cached`
- Let’s commit our change. If your commit message is a one-liner, you can skip the editor by giving the message directly as part of the git commit command: `git commit -m "changed hello to bye"`

Example “hello world” (5)

- To see your commit history: `git log`
- You can add a brief summary of what was done at each commit: `git log --stat --summary`
- Or you can see the full diff at each commit: `git log -p`

Summary on file tracking

from Lee & Edwards (2013)



The tracked, the modified, and the staged

A file in a directory under git revision control is either tracked or untracked. A tracked file can be unmodified, modified but unstaged, or modified and staged. Confused? Let's try again. There are four possibilities for a file in a git-controlled directory:

1. Untracked

   Object files and executable files that can be rebuilt are usually not tracked.

2. Tracked, unmodified

   The file is in the git repository, and it has not been modified since the last commit. `git status` says nothing about the file.

3. Tracked, modified, but unstaged

   You modified the file, but didn't `git add` the file. The change has not been staged, so it's not ready for commit yet.

4. Tracked, modified, and staged

   You modified the file, and did `git add` the file. The change has been moved to the staging area. It is ready for commit.

   The staging area is also called the "index."

More useful commands

- To rename a tracked file:
  `git mv old-filename new-filename`
- To remove a tracked file from the repository:
  `git rm filename`
- The `mv` or `rm` actions are automatically staged for you, but you still need to git commit your actions.
  `git mv old-filename new-filename`
- Sometimes you make some changes to a file, but regret it, and want to go back to the version last committed. If the file has not been staged yet, you can do:
  `git checkout --filename`

More useful commands

- If the file has been staged, you must first unstage it:
  ```
  git reset HEAD filename
  ```
- There are two ways to display a manual page for a git command. For example, for the git status command, you can type one of the following two commands:
  ```
  git help status
  man git-status
  ```
- Lastly, git grep searches for specified patterns in all files in the repository. To see all places you called print():
  ```
  git grep print
  ```

Questions?

1. Advice — RTFM `https://en.wikipedia.org/wiki/RTFM`
2. Advice — `http://lmgtfy.com/` `http://lmgtfy.com/?q=git+repository`