3 Lecture 3, Jan 12

Announcements

- HW1 posted. Due Thu Jan 21 11:59PM.
- Quiz 1 Thu Jan 21 in class.

Last time

- Computer arithmetics: fixed-point numbers and floating-point numbers.
- Consequences of computer arithmetics: range and precision of single/double precision numbers, cancellations, ...

Today

- Computer languages.
- Comparison of R, MATLAB, and JULIA.
- R, RStudio, RMarkdown, version control.

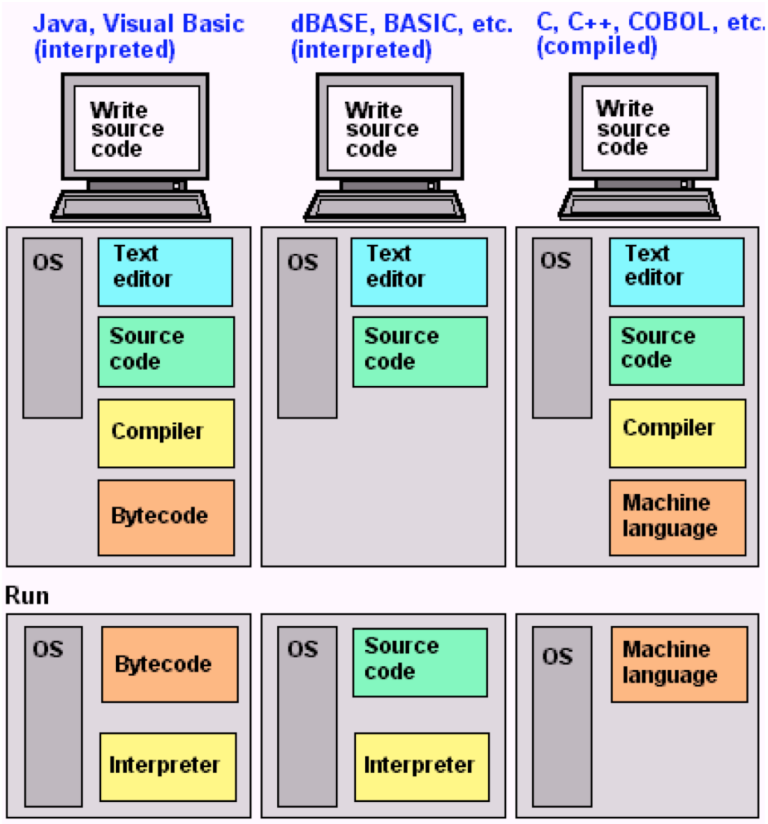
Computer Languages

> **工欲善其事，必先利其器**
>
> *To do a good job, an artisan needs the best tools.*
>
> *The Analects* by Confucius (about 500 BC)

- What features are we looking for in a language?
  - Efficiency (in both run time and memory) for handling big data.
  - IDE support (debugging, profiling).
  - Open source.
  - Legacy code.

– Tools for generating dynamic report (reproducibility).

– Adaptivity to hardware evolution (parallel and distributed computing).

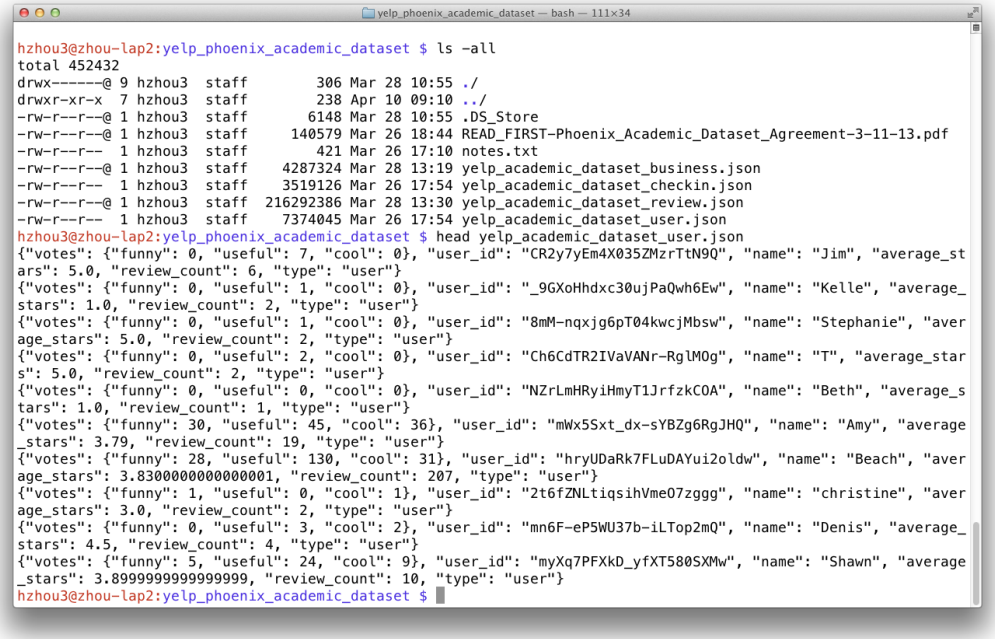


- Types of languages

  1. Compiled languages: C/C++, Fortran, ...
     – Directly compiled to machine code that is executed by CPU
     – Pros: fast, memory efficient
     – Cons: longer development time, hard to debug
  2. Interpreted language: R, Matlab, SAS IML, JavaScript, BASIC, ...
     – Interpreted by interpreter
     – Pros: fast prototyping
     – Cons: excruciatingly slow for loops
  3. Mixed (dynamic) languages: Julia, Python, JAVA, Matlab (JIT), R (JIT), ...

- Compiled to bytecode and then interpreted by virtual machine
- Pros: relatively short development time, cross-platform, good at data preprocessing and manipulation, rich libraries and modules
- Cons: not as fast as compiled language

4. Script languages: shell scripts, Perl, ...
   - Extremely useful for data preprocessing and manipulation
   - E.g., massage the Yelp (`http://www.yelp.com/dataset_challenge`) data before analysis



5. Database languages: SQL, Hadoop. Data analysis never happens if we do not know how to retrieve data from databases.

- Messages
  - To be versatile in the big data era, be proficient in at least one language in each category.
  - To improve efficiency of interpreted languages such as R or MATLAB, avoid loops as much as possible. Aka, vectorize code
    "The only loop you are allowed to have is that for an iterative algorithm."

For some tasks where looping is necessary, consider coding in C or Fortran. It is "convenient" to incorporate compiled code into R or Matlab. But do this **only after profiling!**
Success stories: `glmnet` and `lars` packages in R are based on Fortran.

- Modern languages such as Julia are trying to bridge the gap between interpreted and compiled languages. That is to achieve efficiency without vectorizing code.

- Language features of R, Matlab, and Julia:

Features	R	Matlab	julia
Open source	☺	☹	☺
IDE	RStudio ☺☺	☺☺☺	☹
Dynamic document	☺☺☺(RMarkdown)	☺☺☺	☺☺(IJulia)
Multi-threading	`parallel` pkg	☺	☺
JIT	`compiler` pkg	☺	☺
Call C/Fortran	wrapper, Rcpp	wrapper	no glue code
Call shared library	wrapper	wrapper	no glue code
Typing	☹	☺☺	☺☺☺
Pass by reference	☹	☹	☺☺☺
Linear algebra	☹	MKL, Arpack	OpenBLAS, Eigpack
Distributed computing	☹	☺	☺☺☺
Sparse linear algebra	☹ (`Matrix` package)	☺☺☺	☺☺☺
Documentation	☹	☺☺☺	☺☺
Profiler	☹	☺☺☺	☺☺☺

- Benchmark code `R-benchmark-25.R` from `http://r.research.att.com/benchmarks/R-benchmark-25.R` covers many commonly used numerical operations used in statistics. We ported (literally) to Matlab and Julia and report the run times (averaged over 5 runs) here.

  Matlab benchmark code:
  `http://hua-zhou.github.io/teaching/biostatm280-2016winter/benchmark_matlab.m`
  Julia benchmark code:

`http://hua-zhou.github.io/teaching/biostatm280-2016winter/benchmark_julia.jl`

Machine specs: Intel i7 @ 2.6GHz (4 physical cores, 8 threads), 16G RAM, Mac OS 10.11.2.

Test	R 3.2.2	MATLAB R2014a	JULIA 0.4.2
Matrix creation, trans, deformation (2500×2500)	0.77	0.17	**0.14**
Power of matrix (2500×2500, $A.^{1000}$)	0.26	**0.11**	0.21
Quick sort ($n = 7 \times 10^6$)	0.76	**0.24**	0.69
Cross product (2800×2800, $A^T A$)	10.72	0.35	**0.31**
LS solution ($n = p = 2000$)	1.28	**0.07**	0.09
FFT ($n = 2,400,000$)	0.40	**0.04**	0.64
Eigen-values (600×600)	0.82	**0.31**	0.57
Determinant (2500×2500)	3.76	0.18	**0.17**
Cholesky (3000×3000)	4.23	**0.15**	0.20
Matrix inverse (1600×1600)	3.37	**0.16**	0.21
Fibonacci (vector calc)	0.34	**0.17**	0.68
Hilbert (matrix calc)	0.21	0.07	**0.01**
GCD (recursion)	0.31	0.14	**0.12**
Toeplitz matrix (loops)	0.36	**0.0014**	0.02
Escoufiers (mixed)	0.45	0.40	**0.21**

- A slightly more complicated (or realistic) example taken from Doug Bates's slides `http://www.stat.wisc.edu/~bates/JuliaForRProgrammers.pdf`. The task is to use Gibbs sampler to sample from bivariate density

$$f(x, y) = kx^2 \exp(-xy^2 - y^2 + 2y - 4x), x > 0,$$

using the conditional distributions

$$\begin{aligned} X|Y &\sim \Gamma\left(3, \frac{1}{y^2+4}\right) \\ Y|X &\sim \mathcal{N}\left(\frac{1}{1+x}, \frac{1}{2(1+x)}\right). \end{aligned}$$

Let's sample 10,000 points from this distribution with a thinning of 500.

  - How long does R take? 42 seconds.
    `http://hua-zhou.github.io/teaching/biostatm280-2016winter/gibbs_r.html`

- How long does Julia take? 0.43 seconds. `http://hua-zhou.github.io/teaching/biostatm280-2016winter/gibbs_julia.html`
- With similar coding efforts, Julia offers $\sim$ 100 fold speed-up! JIT in R didn't kick in. Neither does Matlab, which took about 20 seconds.
- Julia offers the capability of strong typing of variables. This facilitates the optimization by compiler.
- With little extra efforts, we can do parallel and distributed computing using Julia.

Benchmark of the same example in other languages including Rcpp is available in the blogs by Darren Wilkinson (`http://bit.ly/IWhJ52`) and Dirk Eddelbuettel's (`http://dirk.eddelbuettel.com/blog/2011/07/14/`).

> *"As some of you may know, I have had a (rather late) mid-life crisis and run off with another language called Julia.* ***`http://julialang.org`*** *"*
>
> Doug Bates (on the `knitr` Google Group)

Reproducible research (in computational science)

> *An article about computational result is* **advertising**, **not** *scholarship. The actual scholarship is the full software environment, code and data, that produced the result.*
>
> Buckheit and Donoho (1995)
> also see Claerbout and Karrenbach (1992)

- 3 stories of *not* being reproducible.
  - Duke Potti Scandal.

Potti et al. (2006) Genomic signatures to guide the use of chemotherapeutics, *Nature Medicine*, 12(11):1294–1300.

Baggerly and Coombes (2009) Deriving chemosensitivity from cell lines: Forensic bioinformatics and reproducible research in high-throughput biology, *Ann. Appl. Stat.*, 3(4):1309–1334. `http://projecteuclid.org/euclid.aoas/1267453942`

More information is available at

`http://en.wikipedia.org/wiki/Anil_Potti`

`http://simplystatistics.org/2012/02/27/the-duke-saga-starter-set/`

– Nature Genetics (2013 Impact Factor: 29.648). 20 articles about microarray profiling published in *Nature Genetics* between Jan 2005 and Dec 2006.

ANALYSIS

nature genetics

Repeatability of published microarray gene expression analyses

John P A Ioannidis[1–3], David B Allison[4], Catherine A Ball[5], Issa Coulibaly[4], Xiangqin Cui[4], Aedín C Culhane[6,7], Mario Falchi[8,9], Cesare Furlanello[10], Laurence Game[11], Giuseppe Jurman[10], Jon Mangion[11], Tapan Mehta[4], Michael Nitzberg[5], Grier P Page[4,12], Enrico Petretto[11,13] & Vera van Noort[14]

Given the complexity of microarray-based gene expression studies, guidelines encourage transparent design and public data availability. Several journals require public data deposition and several public databases exist. However, not all data are publicly available, and even when available, it is unknown whether the published results are reproducible by independent scientists. Here we evaluated the replication of data analyses in 18 articles on microarray-based gene expression profiling published in *Nature Genetics* in 2005–2006. One table or figure from each article was independently evaluated by two teams of analysts. We reproduced two analyses in principle

research, the Uniform Guidelines of the International Committee of Medical Journal Editors state that authors should "identify the methods, apparatus and procedures in sufficient detail to allow other workers to reproduce the results"[12]. Making primary data publicly available has many challenges but also many benefits[13]. Public data availability allows other investigators to confirm the results of the original authors, exactly replicate these results in other studies and try alternative analyses to see whether results are robust and to learn new things. Journals such as *Nature Genetics* require public data deposition as a prerequisite for publication for microarray-based research. Yet, the extent to which data are indeed made fully and accurately publicly available and permit con-

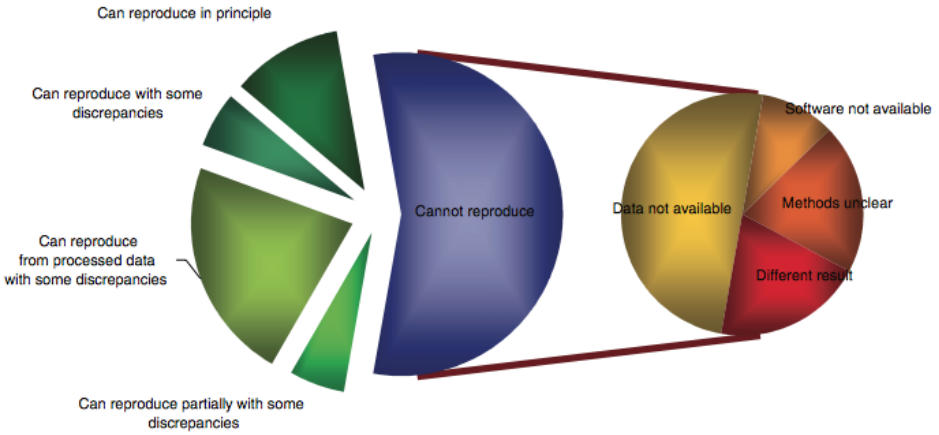
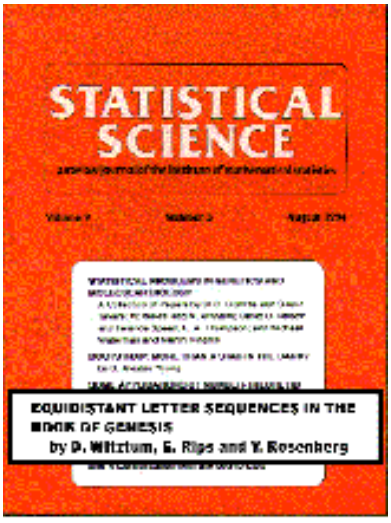
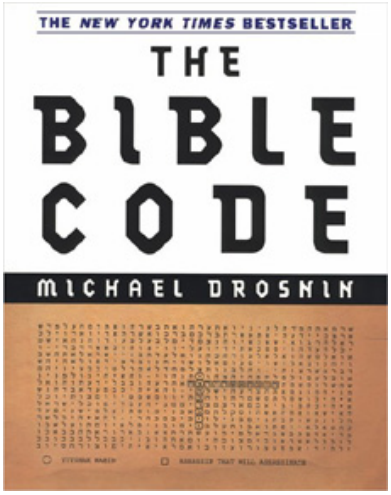


Figure 1 Summary of the efforts to replicate the published analyses.

– Bible code.





Witztum et al. (1994) Equidistant letter sequences in the book of genesis. *Statist. Sci.*, 9(3):429438. `http://projecteuclid.org/euclid.ss/1177010393`

McKay et al. (1999) Solving the Bible code puzzle, *Statist. Sci.*, 14(2):150–173.
`http://cs.anu.edu.au/~bdm/dilugim/StatSci/`

- Why reproducible research?
  - Replicability has been a foundation of science. It helps accumulate scientific knowledge.
  - Better work habit boosts quality of research.
  - Greater research impact.
  - Better teamwork. For you, it means better communication with your advisor (Buckheit and Donoho, 1995).
  - ...
- Readings.
  - Buckheit and Donoho (1995) Wavelab and reproducible research, in *Wavelets and Statistics*, volume 103 of *Lecture Notes in Statistics*, page 55–81.

Springer Newt York. http://statweb.stanford.edu/~donoho/Reports/1995/wavelab.pdf

Donoho (2010) An invitation to reproducible computational research, *Biostatistics*, 11(3):385-388.

- Peng (2009) Reproducible research and biostatistics, *Biostatistics*, 10(3):405–408.

  Peng (2011) Reproducible research in computational science, *Science*, 334(6060):1226–1227.

  Roger Peng's blogs *Treading a New Path for Reproducible Research*.
  http://simplystatistics.org/2013/08/21/treading-a-new-path-for-reproducible-
  http://simplystatistics.org/2013/08/28/evidence-based-data-analysis-treading
  http://simplystatistics.org/2013/09/05/implementing-evidence-based-data-anal

- *Reproducible research with R and RStudio* by Christopher Gandrud. It covers many useful tools: R, RStudio, LaTeX, Markdown, *knitr*, Github, Linux shell, ...

  This book is nicely reproducible. Git clone the source from https://github.com/christophergandrud/Rep-Res-Book and you should be able to compile into a pdf.

- *Reproducibility in Science* at
  http://ropensci.github.io/reproducibility-guide/

- How to be reproducible in statistics?

> *When we publish articles containing figures which were generated by computer, we also publish the complete software environment which generates the figures.*
>
> Buckheit and Donoho (1995)

  - For theoretical results, include all detailed proofs.
  - For data analysis or simulation study
    - Describe your computational results with painstaking details.

    - Put your code on your website or in an online supplement (required by many journals, e.g., *Biostatistics*, *JCGS*, ...) that allow replication of entire analysis or simulation study. A good example: `http://stanford.edu/~boyd/papers/admm_distr_stats.html`
    - Create a dynamic version of your simulation study/data analysis.

- What can we do now? At least make your homework reproducible!
  - Document everything!
  - Everything is a text file (.csv, .tex, .bib, .Rmd, .R, ...) They aid future proof and are subject to version control.

    Word/Excel are not text files.
  - All files should be human readable. Abundant comments and adopt a good style.
  - Tie your files together.
  - Use a dynamic document generation tool (weaving/knitting text, code, and output together) for documentation.
  - Use a version control system proactively.
  - Print `sessionInfo()` in R.

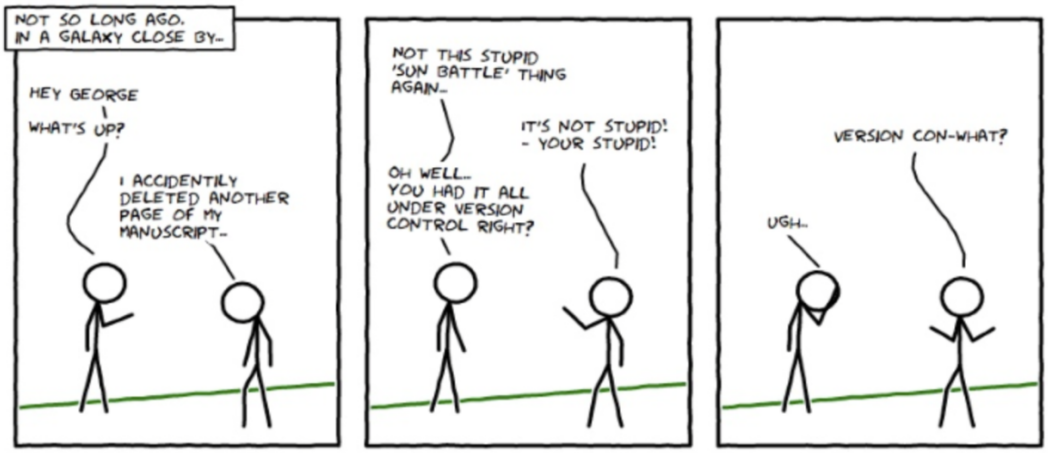
  For your homework, submit (put in the `master` branch) a final pdf or html report and all files and instructions necessary to reproduce all results.

- Tools for dynamic document/report generation.
  - R: RMarkdown, knitr, Sweave.
  - Matlab: automatic report generator.
  - Python: IPython, Pweave.
  - Julia: IJulia.

  For this course, please write homework report in RMarkdown (or IJulia if you use Julia).

Version control by Git

If it's not in source control, it doesn't exist.



- Collaborative research. Statisticians, as opposed to "closet mathematicians", rarely do things in vacuum.
  - We talk to scientists/clients about their data and questions.
  - We write code (a lot!) together with team members or coauthors.
  - We run code/program on different platforms.
  - We write manuscripts/reports with co-authors.
  - ...
- Why version control?
  - A centralized repository helps coordinate multi-person projects.
  - Synchronize files across multiple computers and platforms.
  - Time machine. Keep track of all the changes and revert back easily (reproducible).
  - Storage efficiency.
  - `github.com` is becoming a *de facto* central repository for open source development. E.g., all packages in Julia are distributed through `github.com`.

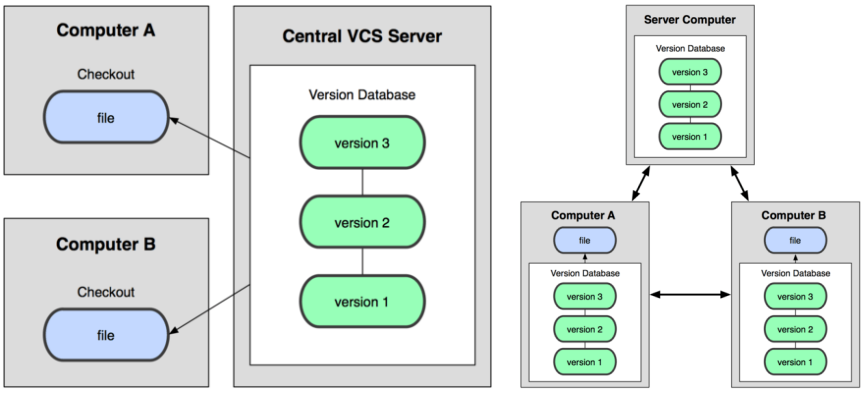
- Available version control tools.
  - Open source: cvs, subversion (aka svn), Git, ...
  - Proprietary: Visual SourceSafe (VSS), ...
  - Dropbox? Mostly for file back and sharing, limited version control (1 month?), ...

  We use Git in this course.

- Why Git?
  - The Eclipse Community Survey in 2014 shows Git is the most widely used source code management tool *now*. Git (33.3%) vs svn (30.7%).
  - History: Initially designed and developed by Linus Torvalds in 2005 for Linux kernel development. "`git`" is the British English slang for "unpleasant person".

    > I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'git'.
    >
    > Linus Torvalds

  - A fundamental difference between svn (centralized version control system, left plot) and Git (distributed version control system, right plot):



  - Advantages of Git.
    * Speed and simple (?) design.

    * Strong support for non-linear development (1000s of parallel branches).
    * Fully distributed. Fast, no internet required, disaster recovery,
    * Scalable to large projects like the Linux kernel project.
    * Free and open source.
  - Be aware that svn is still widely used in IT industry (Apache, GCC, SourceForge, Google Code, ...) and R development. E.g., type
    `svn log -v -l 5 https://svn.r-project.org/R`
    on command line to get a glimpse of what R development core team is doing. Good to master some basic svn commands.

- What do I need to use Git?
  - A Git server enabling multi-person collaboration through a centralized repository.
    * `github.com`: unlimited public repositories, private repositories costs $, academic user can get 5 private repositories for free.
    * `bitbucket.org`: unlimited private repositories for academic account (register for free using your UCLA email).

    For this course, use `bitbucket.org` please.
  - Git client.
    * Linux: installed on many servers. If not, install on CentOS by `yum install git`.
    * Mac: install by `port install git`.
    * Windows: GitHub for Windows (GUI), TortoiseGIT (is this good?)

    Don't rely on GUI. Learn to use Git on command line.

- Life cycle of a project.
  Stage 1:
  - A project (idea) is born on `bitbucket.org`, with directories say `codebase`, `datasets`, `manuscripts`, `talks`, ...
  - Advantage of `bitbucket.org`: privacy of research ideas (free private repositories).

– Downside of `bitbucket.org`: not as widely known as `github.com`.

Stage 2:

– Hopefully, research idea pans out and we want to distribute a standalone software development, e.g., an R package, repository at `github.com`.

– This usually inherits from the `codebase` folder and happens when we submit a paper.

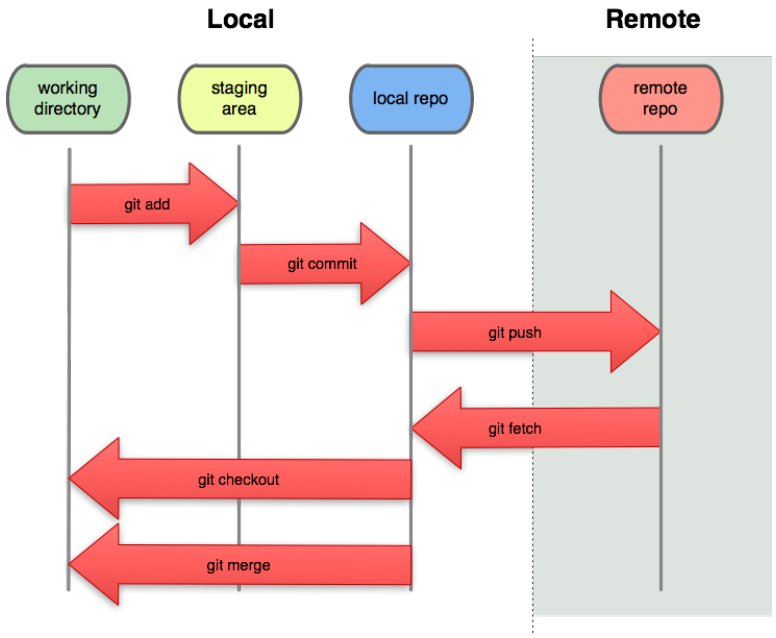
– Challenges: keep all version history. It's doable.

Stage 3:

– Active maintenance of the public software repository.

– At least three branches: `develop`, `master`, `gh-pages`.
`develop`: main development area.
`master`: software release.
`gh-pages`: software webpage.

- Basic workflow of Git.



– Synchronize local Git directory with remote repository (`git pull`).

– Modify files in local working directory.

  - Add snapshots of them to staging area (`git add`).
  - Commit: store snapshots permanently to (local) Git repository (`git commit`).
  - Push commits to remote repository (`git push`).

- Basic Git usage.

  - Register for an account on a Git server, e.g., `bitbucket.org`. Fill out your profile, upload your public key to the server, ...
  - Identify yourself at local machine:
    ```
    git config --global user.name "Hua Zhou"
    git config --global user.email "huazhou@ucla.edu"
    ```
    Name and email appear in each commit you make.
  - Initialize a project:
    * Create a repository, e.g., `biostat-m280-2016-winter`, on the server `bitbucket.org`. Then clone to local machine
      ```
      git clone git@bitbucket.org:username/biostat-m280-2016-winter.git
      ```
    * Alternatively use following commands to initialize a Git directory from a local folder and then push to the Git server
      ```
      git init
      git remote add origin git@bitbucket.org:username/biostat-m280-2016-winte
      git push -u origin master
      ```
  - Edit working directory.
    `git pull` update local Git repository with remote repository (fetch + merge).
    `git status` displays the current status of working directory.
    `git log filename` displays commit logs of a file.
    `git diff` shows differences (by default difference from the most recent commit).
    `git add ...` adds file(s) to the staging area.
    `git commit` commits changes in staging area to Git directory.
    `git push` publishes commits in local Git directory to remote repository.
    Following demo session is on my local Mac machine.

```
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ pwd
/Users/hzhou3/github.ncsu/mglm
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ ls
.DS_Store    .gitignore   datasets/    manuscripts/
.git/        codebase/    literature/  talks/
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git pull
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 5 (delta 3), reused 5 (delta 3)
Unpacking objects: 100% (5/5), done.
From github.ncsu.edu:hzhou3/vctest
   80be212..b22d29f  master     -> origin/master
Updating 80be212..b22d29f
Fast-forward
 manuscripts/letter-skat-famskat/Letter_to_the_editor.tex | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ echo "hello st790 class" > gitdemo.txt
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ ls
.DS_Store    .gitignore   datasets/    literature/  talks/
.git/        codebase/    gitdemo.txt  manuscripts/
```

```
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git add gitdemo.txt
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git status .
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

   new file:   gitdemo.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

   codebase/Example_RNAseq_top100/
   codebase/MGLM/R/.Rhistory
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git commit -m "git demo for st790 class"
[master ea636ff] git demo for st790 class
 1 file changed, 1 insertion(+)
 create mode 100644 gitdemo.txt
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git log gitdemo.txt
commit ea636ff5665bc26bf8a79751b75d0e9d67bdb7d1
Author: Hua Zhou <hua_zhou@ncsu.edu>
Date:   Sun Jan 11 17:06:23 2015 -0500

    git demo for st790 class
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git push
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 301 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To git@github.ncsu.edu:hzhou3/mglm.git
   77145d2..ea636ff  master -> master
```

`git reset --soft HEAD 1` undo the last commit.

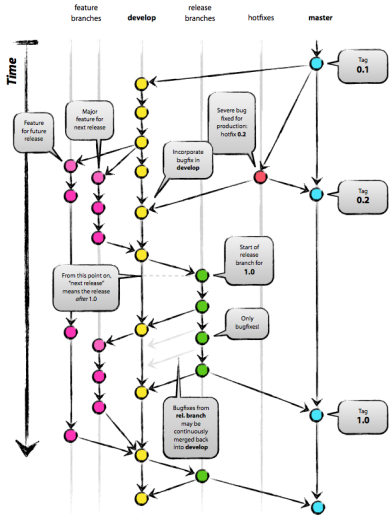
`git checkout filename` go back to the last commit.

`git rm` different from `rm`.

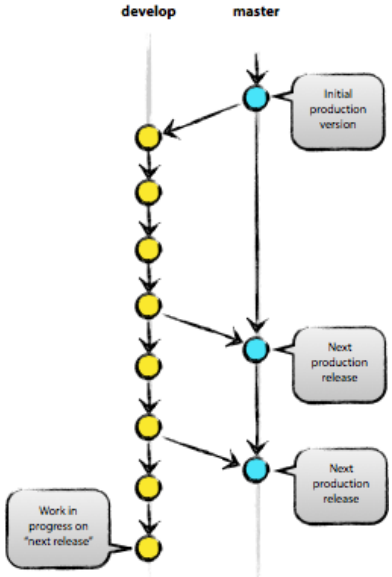
Although `git rm` deletes files from working directory. They are still in Git history and can be retrieved whenever needed. So always be cautious to put large data files or binary files into version control.

```
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ echo "bye st790 class" >> gitdemo.txt
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ cat gitdemo.txt
hello st790 class
bye st790 class
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git diff gitdemo.txt
diff --git a/gitdemo.txt b/gitdemo.txt
index ece6d4e..2bb77f8 100644
--- a/gitdemo.txt
+++ b/gitdemo.txt
@@ -1 +1,2 @@
 hello st790 class
+bye st790 class
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git checkout gitdemo.txt
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ cat gitdemo.txt
hello st790 class
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git rm gitdemo.txt
rm 'gitdemo.txt'
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git status .
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

   deleted:    gitdemo.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

   codebase/Example_RNAseq_top100/
   codebase/MGLM/R/.Rhistory
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git commit -m "delete the git demo file for st790"
[master 4b4f9c5] delete the git demo file for st790
 1 file changed, 1 deletion(-)
 delete mode 100644 gitdemo.txt
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git push
Counting objects: 2, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 238 bytes | 0 bytes/s, done.
Total 2 (delta 1), reused 0 (delta 0)
lTo git@github.ncsu.edu:hzhou3/mglm.git
   ea636ff..4b4f9c5  master -> master
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ ls
.DS_Store    .gitignore   datasets/    manuscripts/
.git/        codebase/    literature/  talks/
```

- Branching in Git.
  - Branches in a project:



– For this course, you need to have two branches: `develop` for your own development and `master` for releases (homework submission). Note `master` is the default branch when you initialize the project; create and switch to `develop` branch immediately after project initialization.



– Commonly used commands:

`git branch branchname` creates a branch.

`git branch` shows all project branches.

`git checkout branchname` switches to a branch.

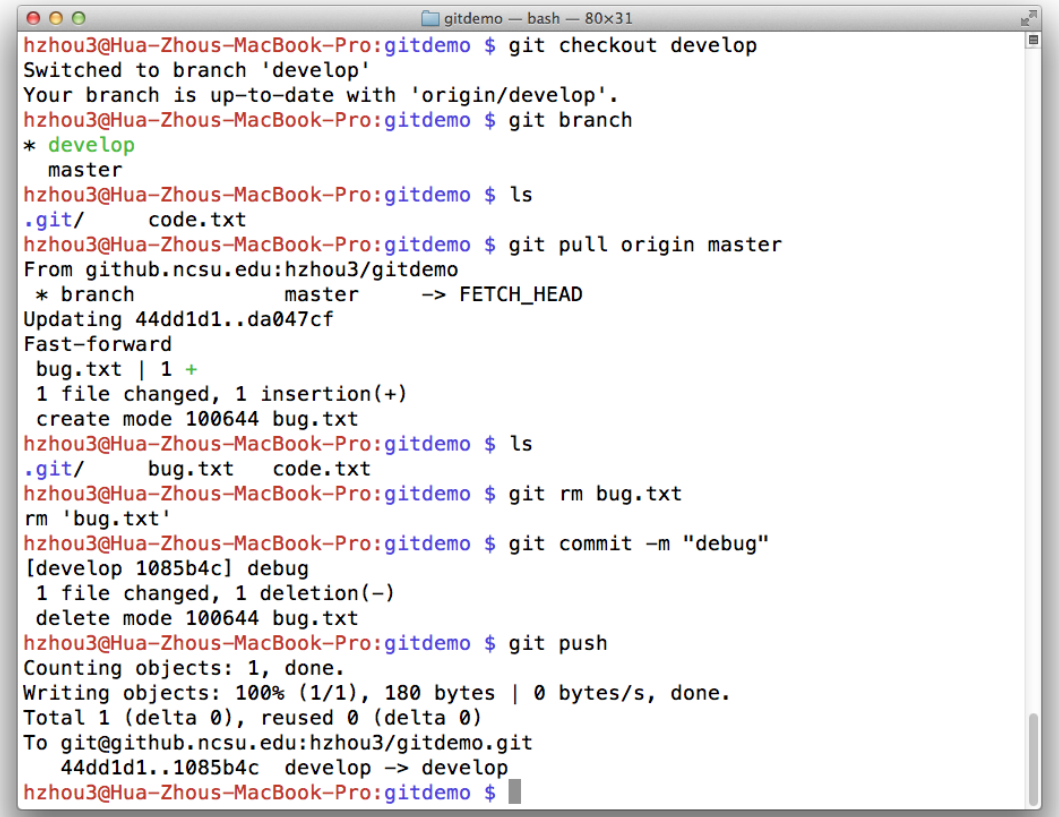
`git tag` shows tags (major landmarks).

`git tag tagname` creates a tag.

– Let's look at a typical branching and merging workflow.

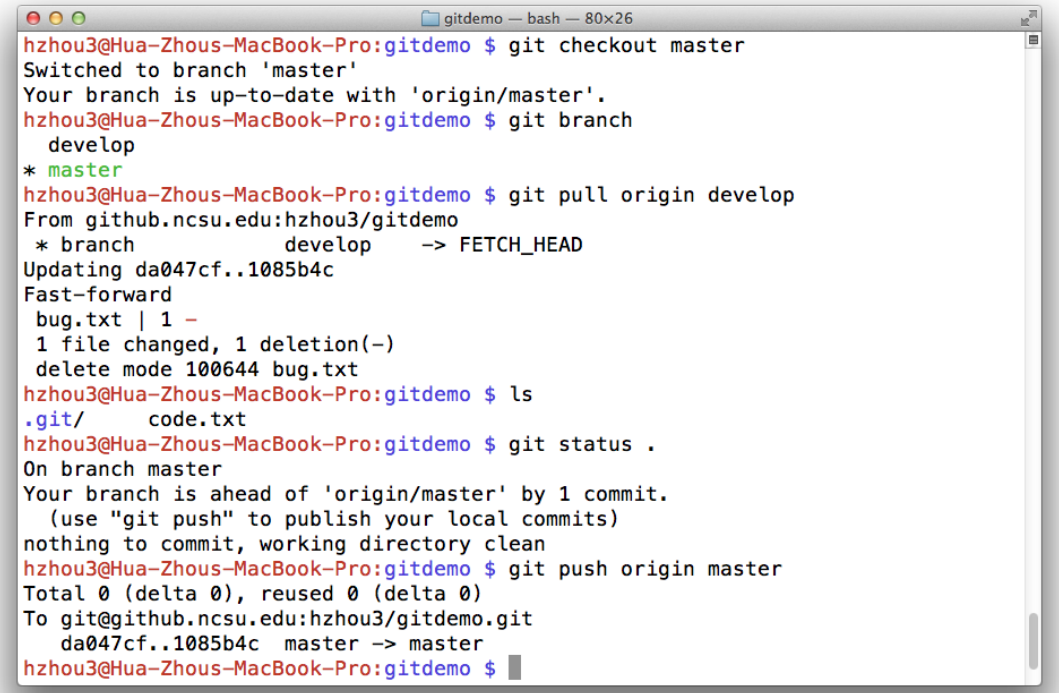
  * Now there is a bug in v0.0.3 ...

How to organize version number of your software? Read blog "R Package Versioning" by Yihui Xie
`http://yihui.name/en/2013/06/r-package-versioning/`

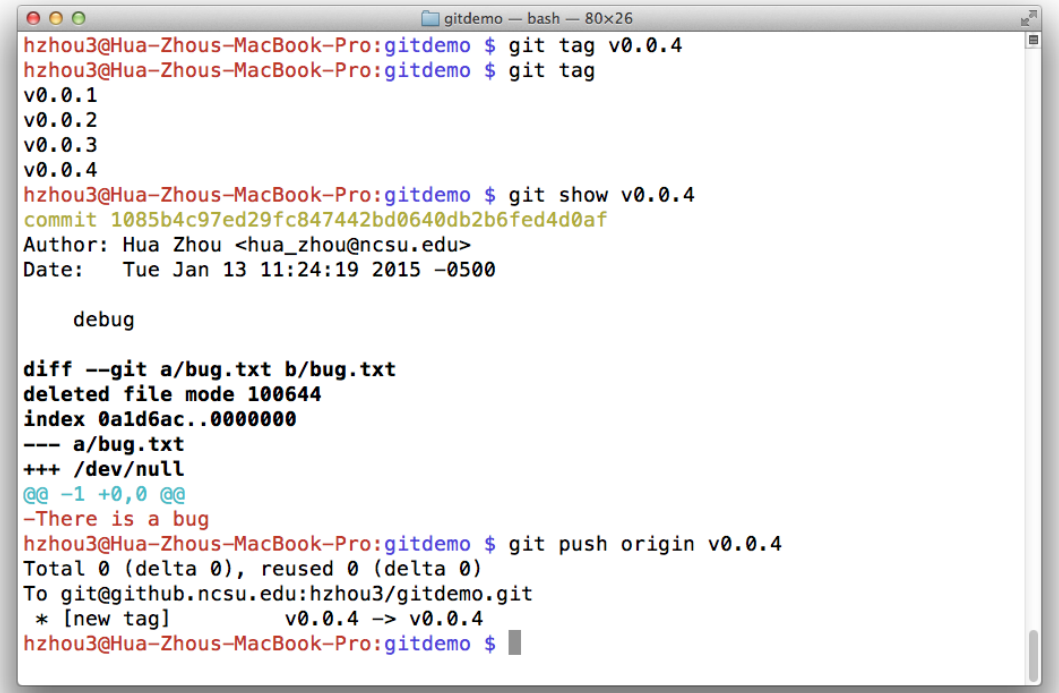


Now 'debug' in develop branch is ahead of master branch.

* Merge bug fix to the `master` branch.



* Tag a new release v0.0.4.



- Further resources:
  - Book *Pro Git*, `http://git-scm.com/book/en/v2`
  - Google
- Some etiquettes of using Git and version control systems in general.
  - Be judicious what to put in repository
    - Not too less: Make sure collaborators or yourself can reproduce everything on other machines.
    - Not too much: No need to put all intermediate files in repository.

    Strictly version control system is for source files only. E.g. only `xxx.tex`, `xxx.bib`, and figure files are necessary to produce a pdf file. Pdf file doesn't need to be version controlled or frequently committed.
  - "Commit early, commit often and don't spare the horses"

– Adding an informative message when you commit is *not* optional. Spending one minute now saves hours later for your collaborators and yourself. Read the following sentence to yourself 3 times:
"Write every commit message like the next person who reads it is an axe-wielding maniac who knows where you live."

References

Baggerly, K. A. and Coombes, K. R. (2009). Deriving chemosensitivity from cell lines: Forensic bioinformatics and reproducible research in high-throughput biology. *Ann. Appl. Stat.*, 3(4):1309–1334.

Buckheit, J. and Donoho, D. (1995). Wavelab and reproducible research. In Antoniadis, A. and Oppenheim, G., editors, *Wavelets and Statistics*, volume 103 of *Lecture Notes in Statistics*, pages 55–81. Springer New York.

Diaconis, P. (2009). The Markov chain Monte Carlo revolution. *Bull. Amer. Math. Soc. (N.S.)*, 46(2):179–205.

Donoho, D. L. (2010). An invitation to reproducible computational research. *Biostatistics*, 11(3):385–388.

McKay, B., Bar-Natan, D., Bar-Hillel, M., and Kalai, G. (1999). Solving the bible code puzzle. *Statist. Sci.*, 14(2):150–173.

Peng, R. D. (2009). Reproducible research and biostatistics. *Biostatistics*, 10(3):405–408.

Peng, R. D. (2011). Reproducible research in computational science. *Science*, 334(6060):1226–1227.

Potti, A., Dressman, H. K., Bild, A., and Riedel, R. F. (2006). Genomic signatures to guide the use of chemotherapeutics. *Nature medicine*, 12(11):1294–1300.

Stigler, S. M. (1986). *The History of Statistics*. The Belknap Press of Harvard University Press, Cambridge, MA. The measurement of uncertainty before 1900.

Teets, D. and Whitehead, K. (1999). The discovery of Ceres: how Gauss became famous. *Math. Mag.*, 72(2):83–93.

Witztum, D., Rips, E., and Rosenberg, Y. (1994). Equidistant letter sequences in the book of genesis. *Statist. Sci.*, 9(3):429–438.