4 Lecture 4, Jan 14

Last time

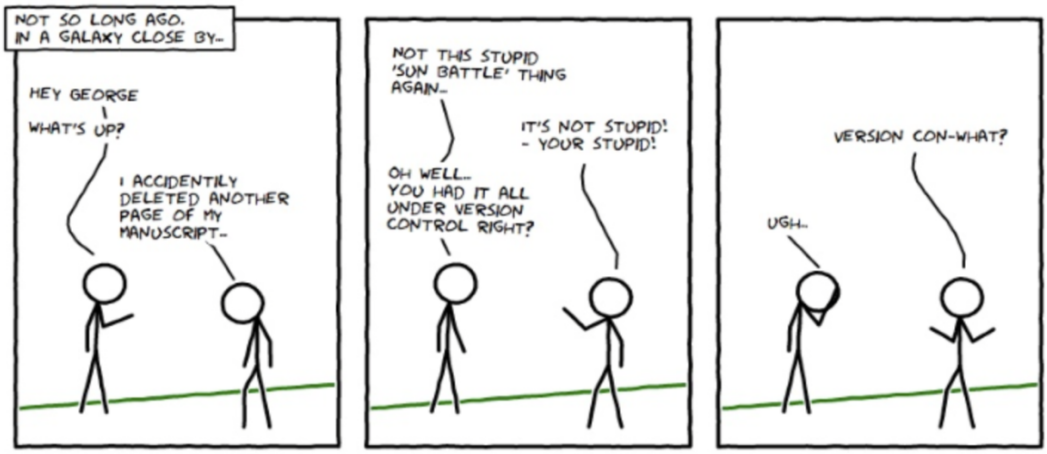
- Computer languages.
- Comparison of R, MATLAB, and JULIA.
- R, RStudio, RMarkdown.

Today

- Version control.

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`.
  - Advise yourself thru `github.com`.

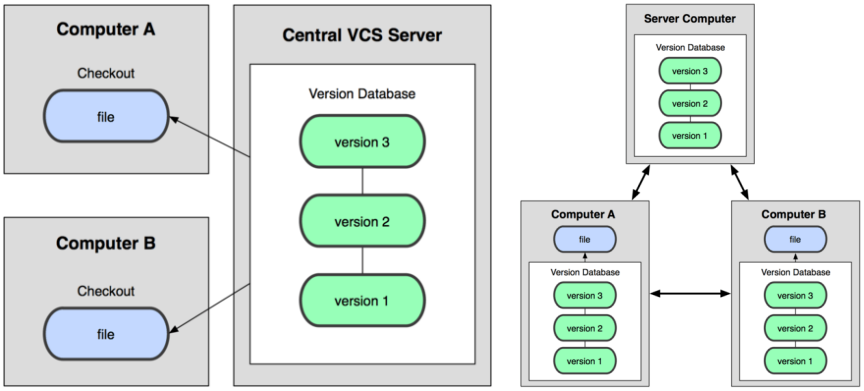
- Available version control tools.
  - Open source: cvs, subversion (aka svn), Git, ...
  - Proprietary: Visual SourceSafe (VSS), ...
  - Dropbox? Mostly for file back and sharing, limited version control (1 month?), ...

  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).

    We use `bitbucket.org` in this course.

  – 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 started 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`. Read the (free) book *R Packages* (`http://r-pkgs.had.co.nz`) by Hadley Wickham for development of R packages and distribution via `github.com`.

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

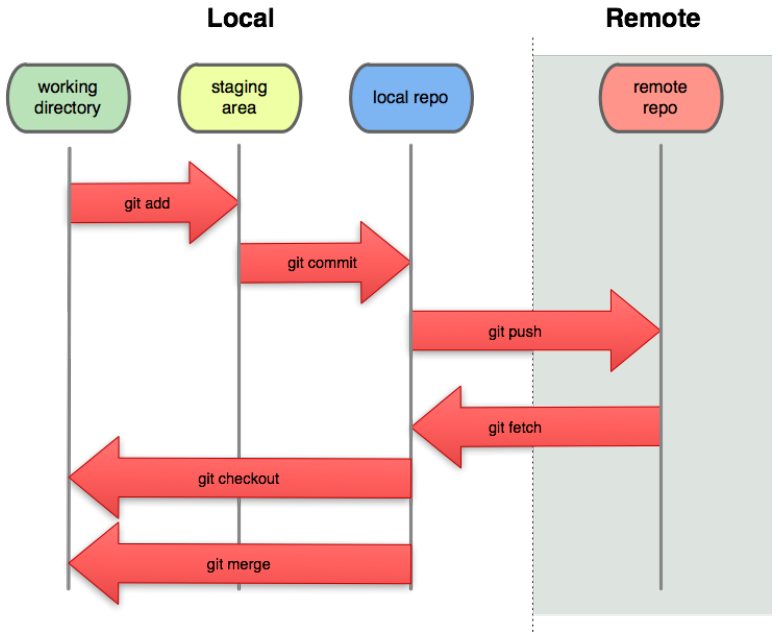
  – Challenges: keep all version history. It's doable.

  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-winter
  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.

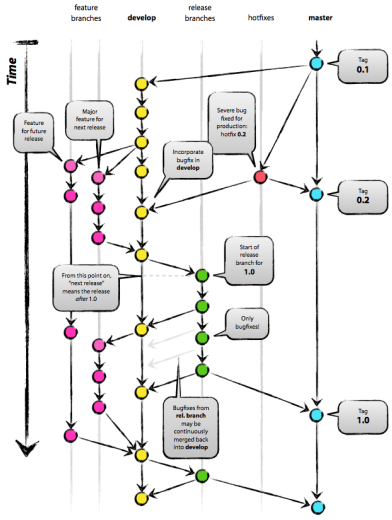
`git checkout filename` go back to the last commit.

`git rm` different from `rm`.

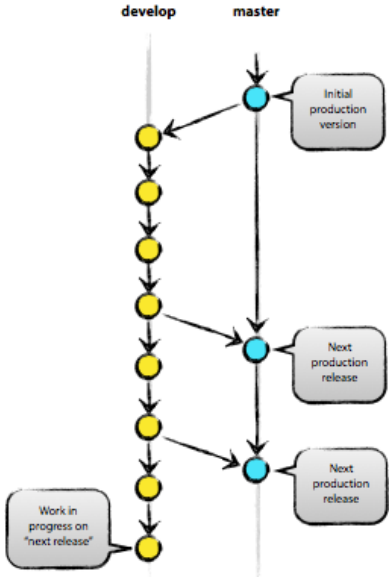
Although `git rm` deletes files from working directory. They are still in Git history and can be retrieved whenever needed. So always be cautious to put large data files or binary files into version control.

```
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.

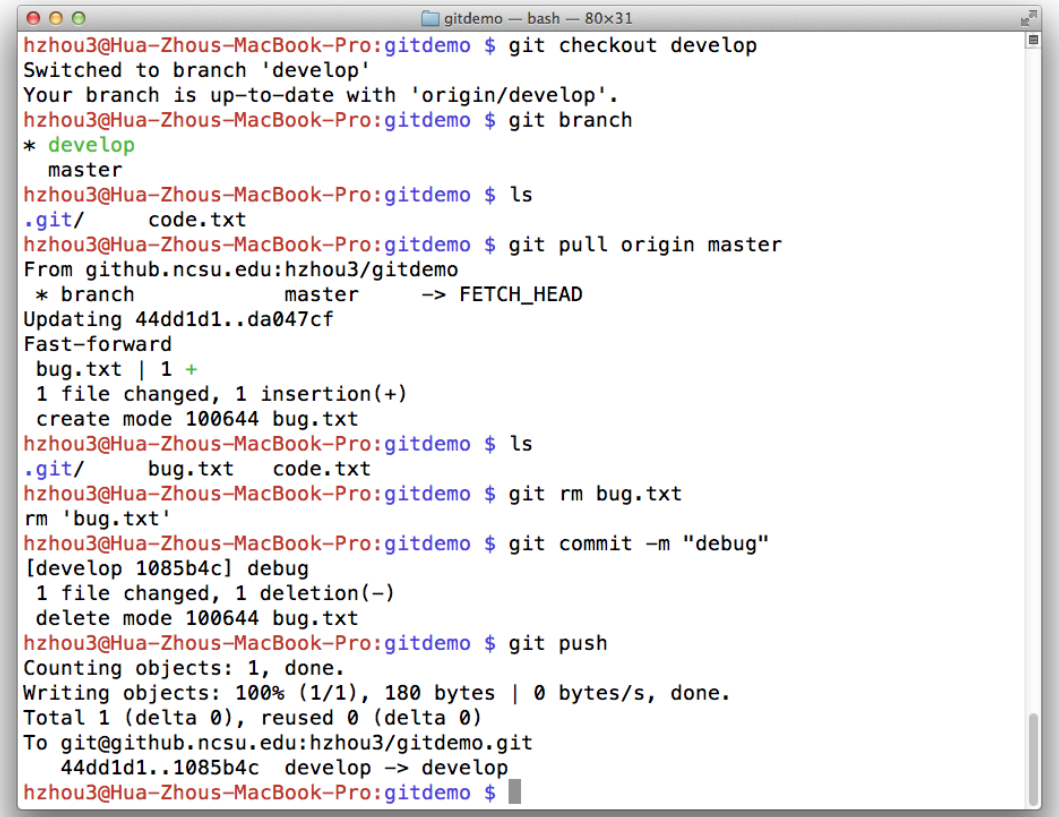
`git checkout branchname` switches to a branch.

`git tag` shows tags (major landmarks).

`git tag tagname` creates a tag.

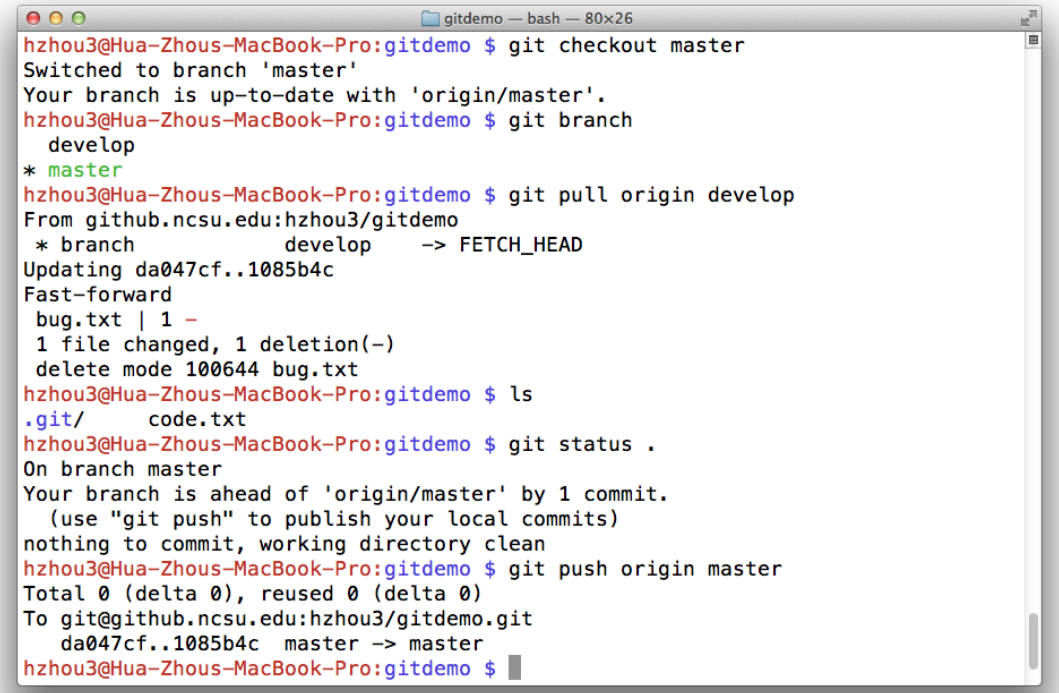
- Let's look at a typical branching and merging workflow.
  - Now there is a bug in v0.0.3 ...

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/`

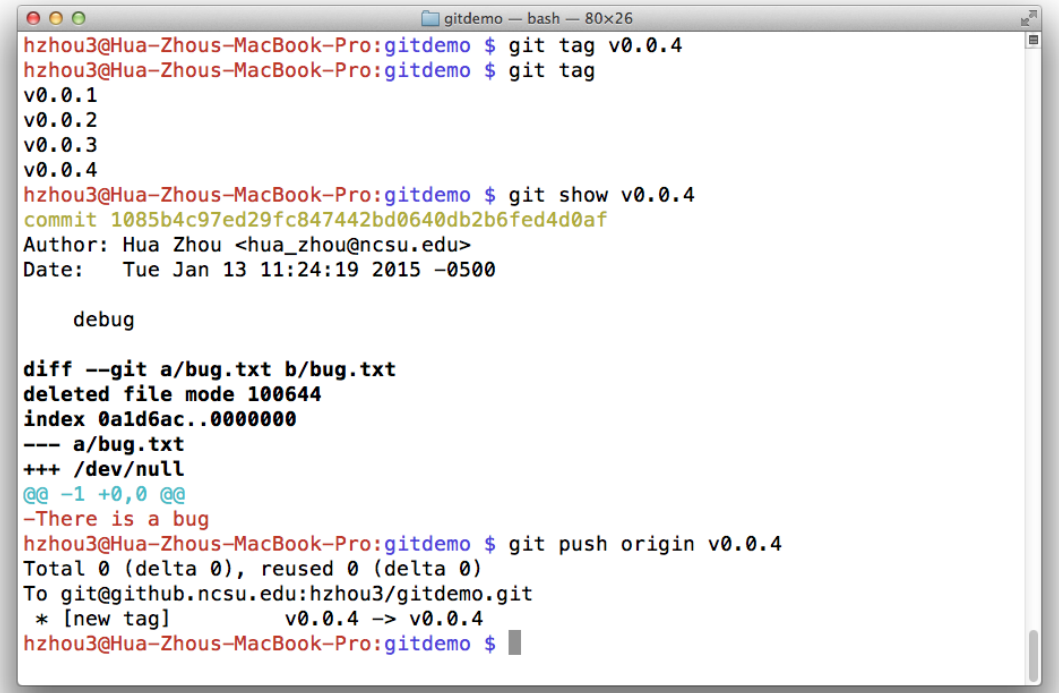


Now 'debug' in develop branch is ahead of master branch.

* 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."

Algorithms

- *Algorithm* is loosely defined as a set of instructions for doing something. Input $\rightarrow$ Output.
- Knuth (2005): (1) finiteness, (2) definiteness, (3) input, (4) output, (5) effectiveness
- Basic unit for measuring efficiency is flop. A *flop* (floating point operation) consists of a floating point multiply (or divide) and the usually accompanying addition, fetch and store. Some books such as Lange (2010) and Golub and Van Loan (2013) consider addition as a separate flop.
- How to measure efficiency of an algorithm? Big O notation. If n is the size of a problem, an algorithm has order $O(f(n))$, where the leading term in the number of flops is $c \cdot f(n)$.
- E.g., matrix-vector multiplication `A %*% b`, where $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^n$, takes $O(mn)$ flops. Matrix-matrix multiplication `A %*% B`, where $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{B} \in \mathbb{R}^{n \times p}$, takes $O(mnp)$ flops.
- *Exponential* order $O(b^n)$ (NP-hard="horrible"), *polynomial* order $O(n^q)$ (doable), $O(n \log n)$ (fast), linear order $O(n)$ (fast), log order $O(\log n)$ (super fast).
- One goal of this course is to get familiar with the flop counts for some common numerical tasks in statistics.

  > The form of a mathematical expression and the way the expression should be evaluated in actual practice may be quite different.

- Compare flops of the following two R expressions:

```
G %*% Xt %*% y
G %*% (Xt %*% y)
```

where `G` $\in \mathbb{R}^{p \times p}$, `Xt` $\in \mathbb{R}^{p \times n}$, and `y` $\in \mathbb{R}^n$. "Matrix multiplication is expensive."

- Hardware advancement, e.g., CPU clock rate, only affects constant c. Unfortunately, data size n is increasing too and often at a faster rate.
- Classification of data sets by Huber (1994, 1996).

Data Size	Bytes	Storage Mode
Tiny	10^2	Piece of paper
Small	10^4	A few pieces of paper
Medium	10^6 (megabyte)	A floppy disk
Large	10^8	Hard disk
Huge	10^9 (gigabytes)	Hard disk(s)
Massive	10^{12} (terabytes)	RAID storage

- Difference of $O(n^2)$ and $O(n \log n)$ on massive data. Suppose we have a teraflop supercomputer capable of doing 10^{12} flops per second. For a problem of size $n = 10^{12}$, $O(n \log n)$ algorithm takes about $10^{12} \log(10^{12})/10^{12} \approx 27$ seconds. $O(n^2)$ algorithm takes about 10^{12} seconds, which is approximately 31710 years!
- QuickSort and FFT are celebrated algorithms that turn $O(n^2)$ operations into $O(n \log n)$. *Divide-and-conquer* is a powerful technique. Another example is the Strassen's method, which turns $O(n^3)$ matrix multiplication into $O(n^{\log_2 7})$.