Git

Charles J. Geyer

School of Statistics
University of Minnesota

January, 24, 2020

`http://users.stat.umn.edu/~geyer/git/`

Version Control

revision control
version control
version control system (VCS)
source code control
source code management (SCM)
content tracking

all the same thing

Version Control (cont.)

It isn't just for source code. It's for any "content".

I use it for

- classes (slides, handouts, homework assignments, tests, solutions),
- papers (versions of the paper, tech reports, data, R scripts for data analysis),
- R packages (the traditional source code control, although there is a lot besides source code in an R package),
- notes (long before they turn into papers, I put them under version control).

Very Old Fashioned Version Control

```
asterLA.7-3-09.tex
asterLA.7-16.tex
asterLA.7-16c.tex
asterLA.7-19.tex
asterLA.7-19c.tex
asterLA.7-19z.tex
asterLA8-19.tex
asterLA8-19c.tex
asterLA.9-1.tex
asterLA.9-1c.tex
asterLA.9-11.tex
asterLA.10-3.tex
asterLA.10-3c.tex
asterLA10-4.tex
asterLA10-4c.tex
asterLA10-4d.tex
asterLA01-12.tex
asterLA01-12c.tex
asterLA01-20.tex
asterLA01-20c.tex
asterLA02-22.tex
```

A real example. The versions of Shaw and Geyer (2010, *Evolution*)

A Plethora of Version Control Systems

Ripped off from Wikipedia (under "Revision control")

Local Only	Client-Server	Distributed
SCCS (1972)	CVS (1986)	BitKeeper (1998)
RCS (1982)	ClearCase (1992)	GNU arch (2001)
	Perforce (1995)	Darcs (2002)
	Subversion (2000)	Monotone (2003)
		Bazaar (2005)
		Git (2005)
		Mercurial (2005)

There were more, but I've only kept the ones I'd heard of.

Don't worry. We're only going to talk about one (git).

Mindshare

Starting from nowhere in 2005, git has gotten dominant mindshare.

In Google Trends, the only searches that are trending up are for git and github. Searches for competing version control systems are trending down.

Many well known open-source projects are on git. The Linux kernel (of course since Linus Torvalds wrote git to be the VCS for the kernel), android (as in phones), Ruby on Rails, Gnome, Qt, KDE, X, Perl, Go, Python, Vim, and GNU Emacs.

Some aren't. R is still on Subversion. Firefox is still on Mercurial.

Why?

Section 28.1.1.1 of the GNU Emacs manual.

Version control systems provide you with three important capabilities:

- Reversibility: the ability to back up to a previous state if you discover that some modification you did was a mistake or a bad idea.
- Concurrency: the ability to have many people modifying the same collection of files knowing that conflicting modifications can be detected and resolved.
- History: the ability to attach historical data to your data, such as explanatory comments about the intention behind each change to it. Even for a programmer working solo, change histories are an important aid to memory; for a multi-person project, they are a vitally important form of communication among developers.

Getting Git

Type “git” into Google and follow the first link it gives you

`http://git-scm.com/`

Under “downloads” it tells you how to get git for Windows and for Mac OS X.

If you have Linux, it just comes with (use the installer for your distribution).

E. g., on Ubuntu

```
sudo apt-get update
sudo apt-get install git
```

What?

(again from Wikipedia under "Git (software)")

Torvalds has quipped about the name *git*, which is British English slang meaning "unpleasant person". Torvalds said: "I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'git'." The man page describes Git as "the stupid content tracker".

(and from memory)

Linus has also said that it was a short name that hadn't already been used for a UNIX command.

Before Anything Else

Tell `git` who you are.

```
git config --global user.name "Charles J. Geyer"
git config --global user.email charlie@stat.umn.edu
```

Of course, replace my name and e-mail address with yours.

Cloning an Existing Project

```
git clone git://github.com/cjgeyer/foo.git
```

Or you can copy an actual repository if it is on the same computer

```
git clone /home/geyer/GitRepos/Git
```

Or you can do the same from another computer

```
git clone geyer@ssh.stat.umn.edu:/home/geyer/GitRepos/Git
```

via `ssh` (requires password or passphrase). The syntax is the same as for remote files when using `scp`.

Starting a New Project

```
mkdir foo
cd foo
git init
```

As yet there are no files in the project, because you haven't put any there. But

```
ls -A
```

shows a directory named `.git` (the `-A` flag to the `ls` command is needed to show a file or directory beginning with a dot) where git will store all version control information.

Warning about Starting a New Project

If you are making an R package, put the package directory **in** your git repo. Do not make it your git repo. I. e., if the package is `foo`, then the directory `foo` which contains the package (has files `DESCRIPTION`, `NAMESPACE`, etc. and directories `R`, `data`, `man`, etc.) should not be the top level directory of the repo (the one the directory `.git` is in).

Commits

Git remembers what you **commit**. Nothing else.

Git offers very precise control of what a commit remembers. It remembers exactly the files you tell it to. Nothing else.

What it remembers in a commit is what you have put in the **index**, also called the **staging area**.

Commits (cont.)

Another way to look at what `git commit` does is that it makes a "snapshot" of the **working tree** (the contents of the directory in which you ran `git init` or which was produced by `git clone`).

This is true in one sense and false in another.

If you clone a git repository, you get the entire history, all the versions of files remembered in all commits, and the working tree will contain the most recent version of each file. But it does not contain versions never committed.

Git add

The command `git add` adds files to the index (staging area).

Shell file globbing is useful in conjunction with this.

```
git add [A-Z]*
git add R/*.R
git add src/*.{c,h,f,cc}
git add man/*.Rd
git add data/*.{R,tab,txt,csv,rda}
git add tests/*.{R,Rout.save}
git add vignettes/*.Rnw
```

adds most of what you might want to track in an R package.

Commits (cont.)

The command `git commit` does commits.

```
git commit -m "first commit"
```

does a commit. The argument of the `-m` flag is the commit message. If the `-m` flag is omitted, then git drops you into your default editor to compose the commit message.

Commits (cont.)

```
git commit -a
```

adds all files that are being tracked to the index (staging area) and then does the commit. This is useful when you have made changes to files that were already being tracked (were added in previous commits).

Git status

The command `git status` (from the man page)

> *Displays paths that have differences between the index file and the current HEAD commit, paths that have differences between the working tree and the index file, and paths in the working tree that are not tracked by git (and are not ignored by gitignore(5)). The first are what you would commit by running git commit; the second and third are what you could commit by running git add before running git commit.*

Gitignore

The file `.gitignore` in the top-level directory of a repository tells git what to ignore. This file for R package foo is

```
*.Rcheck
*.so
*.o
*.tar.gz
symbols.rds
```

Note that patterns recognized by the UNIX shell can be used.

You want git to track this file so it is part of every repository.

Git diff

The command

```
git diff
```

shows the differences between the working tree and the index (staging area), the command

```
git diff --cached
```

between the index (staging area) and the previous commit. And there are lots more possibilities.

Help

```
git status --help
man git-status
```

do the same thing (show the man page for `git status`) but

```
git --help
man git
```

are different (the first is terser).

Git log

The command `git log` describes all commits.

Halftime Summary

```
git init
git clone
git add
git commit
git diff
git status
git log
```

Are all you need to know to work on some projects.

But there is lots more (see `git --help`).

Branching and Merging

```
git branch dumbo
git checkout dumbo
```

switches to a new branch (named “dumbo”). Make changes and commits on this branch.

```
git checkout master
```

switches to the original branch (named “master” by convention). Make changes and commits on this branch.

Branching and Merging (cont.)

The two branches proceed independently from the point of the branch until

```
git checkout master
git merge dumbo
```

merges them.

Branching and Merging (cont.)

The merge will complete (making a "merge commit") or it may be unable to resolve conflicts (where overlapping changes have been made in the two branches). Then it stops, and you must resolve the conflicts manually, that is, edit files removing the conflict markers and leaving them in the form you want to commit. Then

```
git commit -a
```

does the merge commit (you do not need this step if there were no conflicts).

Branching and Merging (cont.)

From `get merge --help`

> *Warning: Running git merge with uncommitted changes is discouraged: while possible, it leaves you in a state that is hard to back out of in the case of a conflict.*

Never merge unless both branches have all changes committed (`git status` says "nothing added to commit").

Working with Others

Suppose you had a Github account, you had uploaded a public key following the instructions at Github, and I had set the `foo` repository at Github to allow you write permission.

Then

```
git clone git@github.com:cjgeyer/foo.git
```

is a different URL from which to clone the repository. If you had already cloned from the read-only URL, then

```
git remote origin git@github.com:cjgeyer/foo.git
```

would change the "origin" to the read-write URL.

Working with Others (cont.)

Suppose you have made some changes and committed them to your clone.

First get up-to-date with Github.

```
git pull origin master
```

This will do a merge and may require resolving merge conflicts (and doing another commit when they are). Then upload your commits to Github

```
git push origin master
```

Working with Others (cont.)

Never push changes to a project when you are not up-to-date. You can do this using options to `git push`, but don't. Your collaborators will hate you if you do.

Working with Others (cont.)

Alternatively, suppose I don't want you writing to my Github repository without my knowledge and hence don't give you write permission.

You can clone my repository on Github (making your own Github repository) upload your changes there, and either

- you have "forked" my project (o. k., if it has a free software license),
- you can ask me to pull from you and merge the changes, in which case the project is unified again.