CS506/CS606 (Research Programming)
Distributed version control with Git

Kyle Gorman
Center For Spoken Language Understanding
Oregon Health & Science University

1 Background

Version control (or *revision control*) systems allow users to track and control changes to files (such as documents or software) by preserving (either implicitly or explicitly) earlier versions of the files. *Distributed* version control (DVC) systems—essentially, those that support multiple users and run on multiple machines—allow for *non-linear development*, with several developers working simultaneously on multiple *branches* of a single project. Tech blogger Joel Spolsky, writing in 2010, called DVC "…the biggest advance in software development technology in the [past] ten years."[1].

Git is a popular DVC system created by Linus Torvalds (also the creator of Linux). While Git was designed as a command-line tool, Git servers are often enhanced with a web interface which provides additional support for reporting bugs, tracking progress, and distributing open-source software. Examples of web-enabled Git servers include include GitHub (`github.com`) and our internal GitLab server (`gitlab.cslu.ohsu.edu`).

While Git is primarily used for software development, it is also an excellent tool for collaborative writing and "open research" (i.e., science conducted in the spirit of the open-source movement, with freely available data and code).

2 Git's metaphors

Git is organized around several "metaphors" for version control. These are sufficiently novel that they deserve some explication.

A *repository* is the highest-level data structure for a Git project. The repository files are stored in a subdirectory of the project directory which is called `.git/`. This contains *commits* (to be defined below) and associated metadata. Since Git is a distributed version control system, there may be multiple copies of the repository, including the *remote* on a Git server as well as local copies on each developer's machine. The remote branch is traditionally called `origin`.

When you *clone* a repository, you make a local copy of it that links back to the original. You may make changes to the local repository, but you may not be able to make changes to the remote repository unless you have been granted permission to do so by the project leader(s). When you *fork*

[1] `http://joelonsoftware.com/items/2010/03/17.html`

a repository, you once again make a local copy, but you also make a copy on the remote repository. This is usually done with the intent to propose changes to the original ("upstream") repository.

Each repository contains one or more *branches*; the default branch is traditionally called `master`. A branch represents an "independent line of development" with a separate working directory and project history. Within a single branch, changes are "staged" by marking files in the repository as having been added, modified or removed. Each *commit* consists of this index of changes, a unique identifier, and text describing the changes that have been made. Once a commit is ready, the user can *push* it to a remote repository so that others can use it.

Here is a true story about clones, forks, and branches.

> Kyle implements a system for sentence boundary detection using supervised classification and uploads it to GitLab. While the results are good, he suspects that there are more features than strictly necessary. So, he clones the remote repository and creates a new branch called `ablation`, which he uses to test models with fewer features. Once he has eliminated several features that perform poorly, he stages and commits the changes and pushes the new branch to the remote, and then merges the `ablation` branch into the `master` branch.

3 Basics

We will now walk through a simple example of managing a project with Git. Each Git command is of the form `git VERB ...`, where `VERB` indicates the type of action to be taken.

- Git uses your name and email address to identify your repositories. You can set this up globally once, and then forget about it.

```
$ git config --global user.name "Kyle Gorman"
$ git config --global user.email gormanky@ohsu.edu
```

- To start a project, create a new directory and navigate to it.

```
$ mkdir myproj
$ cd myproj
```

- To make a local directory into a git repository, use the `git init` command.

```
$ git init
Initialized empty Git repository in /Users/gormanky/Documents
   /Code/myproj/.git/
$ ls -a
.    ..   .git
```

- Use the `git add` command to "stage" new files.

```
$ vi program.c
...
$ git add program.c
```

- The `git status` command provides information about changes currently staged.

```
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   program.c
```

- Once files have been added, the `git commit` command is used to generate a *commit* encapsulating the additions or changes made thus far. Each commit has a "message" associated with it describing the changes made. Other developers (as well as your future self!) will use these messages to understand why each change was made, so it is important to write an informative commit message. By default, `git commit` command drops you into a text editor where you write a commit message.

```
$ git commit
...
[master (root-commit) b5c3f11] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 program.c
[master 50eb5cc] initial commit
 1 file changed, 1 insertion(+)
 create mode 100644 program.c
```

  If you would rather write a very short commit message, use the `-m` flag.

```
$ git commit -m "First commit ever"
```

  You can also edit an already-existing commit message using `git commit --amend`.

- Now is the time to make a remote copy of the repository. First, use the web interface on GitHub or GitLab to create an empty remote repository (see `https://help.github.com/articles/create-a-repo/`, steps 1-4 and 6). Then, use `git remote add` to link the local and remote repositories.

```
$ git remote add origin https://github.com/kylebgorman/myproj
   .git
```

  This tells the local repository that it has a corresponding remote repository called `origin` located at `https://github.com/kylebgorman/myproj.git`.

- The command `git push` sends the commit from the current branch (`master`) to the remote repository.

```
$ git push origin master
```

  While it's common to push after every commit, you can have multiple commits per push.

4 Playing well with others

- To make a local copy of a remote repository, use `git clone`.

```
$ git clone https://github.com/kylebgorman/myproj.git
$ cd myproj
```

- The `git fetch` command downloads metadata about any changes to the remote repository that have happened since the local clone was created.

```
$ git fetch origin master
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From https://github.com/kylebgorman/myproj
* branch            master     -> FETCH_HEAD
095111a..ef6764b  master     -> origin/master
```

- However, it does *not* change any of the files in the project directory. To bring them up to date with the remote repository, use `git merge`.

```
$ git merge origin/master
Updating 095111a..ef6764b
Fast-forward
README.md | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)
```

- The command `git pull` is a synonym for `git fetch; git merge`.

- When you want to modify the code, the polite thing to do is to create a new branch. The command `git branch` is used for managing branches; the `git checkout` command is used for switching between branches. To switch to a new branch, use `git checkout -b`.

```
$ git checkout -b evenbetter
Switched to a new branch 'evenbetter'
```

- Then, make changes, stage them, write a commit, and push to the new branch.

```
$ vi program.c
...
$ git status
On branch evenbetter
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in
     working directory)
```

```
        modified:   program.c

no changes added to commit (use "git add" and/or "git commit
   -a")
$ git add program.c
$ git commit -m "made things even better"
[evenbetter 477fd6d] made things even better
1 file changed, 1 insertion(+), 1 deletion(-)
$ git push origin evenbetter
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 309 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To https://github.com/kylebgorman/myproj.git
* [new branch]      evenbetter -> evenbetter
```

- When ready to merge the two branches, checkout `master` and use the `git merge` command.

```
$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
$ git merge evenbetter
Updating ef6764b..ad1b2fe
Fast-forward
README.md | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)
```

5 Contributing via the "fork and pull request" model

Many (if not most) open-source software projects are now developed with Git and hosted at GitHub. Each project has a small team of core maintainers who have "commit privileges", as well as many external contributors, who contribute new features and bug-fixes but do not have commit privileges. Instead, these contributors use the "fork and pull request" model.

1. The contributor forks the upstream repository on GitHub.
2. The contributor clones the forked repository, creating a local copy.
3. The contributor creates a new branch in the forked repository.
4. The contributor modifies the source code and adds the changes to the staging area.
5. The contributor commits the changes and pushes the new branch to the remote fork.
6. The contributor files a pull request with the upstream repository.

7. The maintainers of the upstream repository review the pull request, accept it, and merge the forked branch into the upstream master.

Here is a true story about the "fork and pull request" model.

> Kyle uses an open-source software tool downloaded from GitHub. One day, he stumbles upon a bug that causes demons to come out of his nose (after a manner of speaking). He reads the source code and figures out how to fix the bug. He forks the remote repository on GitHub, clones the fork, and creates a branch called `bugfix`. He modifies the source code to fix the bug, commits the changes, pushes the new branch to the remote fork, and then files a pull request with the upstream repository. The upstream maintainers review the pull request and then merge the `bugfix` branch of Kyle's fork into the upstream master.