6.170 Tutorial 1 - Git

Prerequisites

1. Have Git installed and configured. (See: https://help.GitHub.com/articles/set-up-git)
2. Sign-up for a GitHub account. (http://www.github.com)

Goals of this Tutorial

1. Get familiar with source code version control using Git
2. Create your first Git repository
3. Learn useful Git commands to be used throughout the course
4. Create a GitHub repository and learn to push/pull code from this repository

Today's Tutorial

1. What is Git?

1. A version control system for project code - allows "snapshot" like functionality so that you can view and compare previous versions of your files.
2. Allows easy collaboration for source code development between team members.
3. "The purpose of Git is to manage a project, or a set of files, as they change over time. Git stores this information in a data structure called a repository." [1]

2. How Git works

2.1 Core concept: Repository

A Repository consists of :

1. A set of commit objects identified by a SHA1 name. Think of a commit as a snapshot in time of your project code. You can create as many commits as you want.
2. A set of references to commit objects. These references are called "heads". (Not to be confused with HEAD (capitalized) which is a reference to the current head)

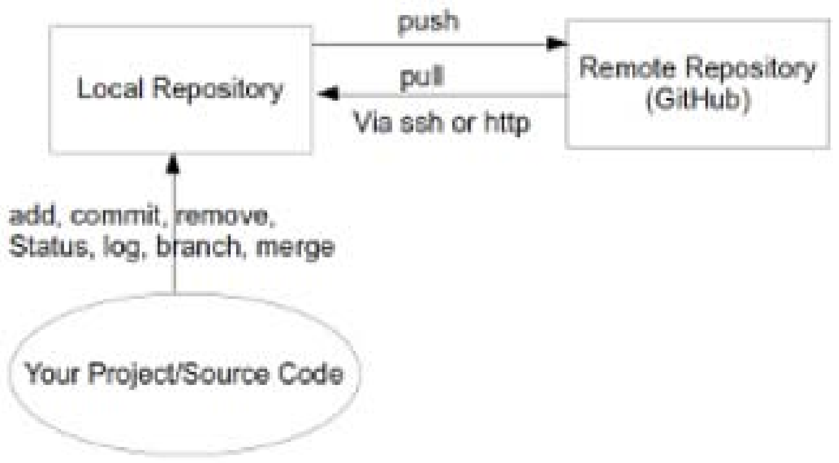
A commit object consists of:

1. A set of *files*, reflecting the state of a project at a given point in time.
2. References to *parent commit objects.*
3. An *SHA1 name*, a 40-character string that uniquely identifies the commit object. The name is composed of a hash of relevant aspects of the commit, so identical commits will always have the same name.
4. Example of a commit (obtained using "git log"):

```
$ commit 034072dc2c02da0ab2872c0315a3f237027eb5e7
$ Author: Your Name <your.name@mit.edu>
$ Date:   Sat Nov 10 17:08:57 2012 -0500
$   1. Better Logging. 2. Schema changes
```

2.2 Simple Operations on Repositories

1. To create a repository, use "`git init`"
   a. `$ mkdir <project_dir>`
   b. `$ cd <project_dir>`
   c. `$ git init`
   d. This will create a ".git" directory in <project_dir>
   e. **Note:** This is your local repository and not necessarily shared with others. See image below for operations on local and remote repositories. Remote repositories are explained later in this tutorial.


push
Local Repository
pull
Via ssh or http
Remote Repository (GitHub)
add, commit, remove, Status, log, branch, merge
Your Project/Source Code

2. To add files to the repository, use "`git add <file_or_dir>`" followed by "`git commit`"
   a. `$ cd <project_dir>`
   b. `$ echo 'Hello World' > hello1.txt`
   c. `$ echo 'Hello MIT' > hello2.txt`
   d. `$ echo 'Hello 6.170' > hello3.txt`
   e. `$ git add hello1.txt hello2.txt`
   f. This tells Git to include both files "hello1.txt" and "hello2.txt" in the next commit to the repository.
   g. "git add ." works recursively and includes all files and subdirectories. **Note:** Use this recursive command cautiously.
   h. To include individual files, use "git add /path/to/file/<filename>"
   i. Wildcard could be used. "git add '*.txt'" will add all files with suffix txt. Note: need to use single quotes.
   j. `$ git commit -a -m "Adding test files to the repository "`
   k. This will create a commit object consisting the two files and add them to the

repository along with the message specified by the “-m” flag.

   I. **Note**: “`git add <file_or_directory>`” must be repeated before each “`git commit`” operation.

3. To check history of commits in your repository, use “`git log`”
4. To delete files from the repository, use “`git rm`”
   a. `$ git rm hello1.txt`
   b. This will remove hello1.txt from your repository as well as your local copy of hello1.txt
5. Lets do a few more operations locally:
   a. `$ cd <project_dir>`
   b. `$ echo 'Hello World! git rocks!' >> hello2.txt`
   c. `$ rm hello3.txt`
6. To check the current status of your project, use “`git status`”
   a. `$ cd <project_dir>`
   b. `$ git status`
   c. This will show all modified files, new files, and deleted files.
7. Comparing two versions of a file
   a. `$ git diff hello2.txt`
   b. This will show you the differences between your local version and the repository version of the file.
   c. To see differences between two previous commits, use “`git diff <commit1> <commit2> [<path>]`”
8. Unstaging a file (Removing a file from a commit)
   Before you push, if you decide that you do not want a file to be included in a commit, you could unstage the file, using “`git reset HEAD [<path>]`”.
9. Checkout a particular version of the file
   To discard local changes, you could use “`git checkout [<path>]`”, which gives you a latest version of the file from the repository. If you would like a particular version of the file. You could do “`git checkout abcde [<path>]`” where ‘abcde’ is the hash of the commit that you want.
10. Revert commit
   To revert a particular commit, you could use “git revert”. One example usage is “`git revert HEAD~3`”, which revert the changes specified by the fourth last commit in HEAD and create a new commit with the reverted changes.
11. Asking for help
   a. `$ git help` or `$ git help <command>`
12. Configure Git using “`git config <option>`” or via “`.gitconfig`”, “`.gitignore`” files
   Sample usage:

```
$ git config --global user.name "Your Name"
$ git config --global alias.co checkout
```

a. My ~/.gitconfig file is shown below:

```
[user]
     name = First Last
     email = first.last@mit.edu
[alias]
     co = checkout
[core]
     excludesfile = /home/user1/.gitignore
```

b. My ~/.gitignore file is shown below:

```
# Ignore swp and tmp files.
*.swp
*~
*#
.#*
*.log.*
```

c. See “git help config” for more details

d. **Note:** Strongly recommend that you have a “.gitignore” file to prevent log files and other temporary files from being added to your repository. GitHub can also create a “.gitignore” file for you when you are creating a new repository.

2.3 Branching and Merging

1. Branches allow parallel development from a commit point, without affecting the parent branch. Branches will be useful when working on your class projects.
2. **Note:** “The terms “branch” and “head” are nearly synonymous in Git. Every branch is represented by one head, and every head represents one branch. Sometimes, “branch” will be used to refer to a head and the entire history of ancestor commits preceding that head, whereas “head” will be used to refer exclusively to a single commit object, the most recent commit in the branch.” [1]
3. To list current branches in your repository, use “`git branch`”
   a. `$ git branch`
   b. This will list all branches in your repository with the current active branch shown with a star against its name
   c. When a new repository is created, a default branch **master** is also created.
4. To create a new branch from a specific commit point, use “`git branch <head> <commit>`”
   a. `$ git branch assignment2 bb7d8622387cd70a31c198f6f84133fc2303ec85`
   b. This will create a branch named “assignment2” from the commit specified in the the hash value.
   c. `$ git branch assignment3 HEAD^`

    d. This will create a branch named “assignment3” from the parent of HEAD. (HEAD^ is a reference to the parent node of the node to which HEAD refers)
5. To switch to a new branch, use `“git checkout <new_branch>”`
    a. `$ git checkout assignment1`
    b. This will make HEAD point to “assignment1”
    c. **Warning:**: If there are any uncommitted changes when you run “git checkout”, Git will behave very strangely. The strangeness is predictable and sometimes useful, but it is best to avoid it. All you need to do, of course, is commit all the new changes before checking out the new head”.[1]
6. Merge a branch using `“git merge <head>”` or `“git pull <head>”`
    a. **Note:** `“git pull <head>”` is preferred.
    b. `$ git checkout master`
    c. Now HEAD points to **master**
    d. `$ git pull assignment1`
    e. This will merge changes from branch “assignment1” into “master”
    f. **Warning:** Git can get very confused if there are uncommitted changes in the files when you ask it to perform a merge. So make sure to commit whatever changes you have made so far before you merge. [1]
7. Delete a branch using `“git branch -d <head>”`

2.3 Team collaboration using GitHub

1. So far, we have been working on a local repository only.
2. Unlike other version control systems, Git does not have a client-server model. It is a peer-to-peer model and every developer has their own independent repository.
3. Git allows communication between repositories via “ssh”, “http”, or fs paths (if the other repository is on the same file system).
4. Using this communication feature, teams can have a “central” repository to share work within the team members. Each team member works on their own local repository. When satisfied with the work, s/he “pushes” code to the central repository. Other developers can then “pull” the code from this repository and get the latest changes. We will use GitHub as this central repository.
5. Login to GitHub and creating a new repository named “hello6170”. Be sure to choose the option of “Create a Readme file”. This will allow you to clone the repository to your local repository immediately.
6. Clone the remote repository using `“git clone <path_to_repository>”`
    a. `$ mkdir <new_repo>`
    b. `$ cd <new_repo>`
    c. `$ git clone https://github.com/<your_username>/hello6170.git`
    d. This will create a local version of the files in your remote repository into the current directory, along with a “.git” sub-directory - the local repository.
    e. The “.git/config” file has the following sections:
        i. `[remote "origin"]`

ii. `    fetch = +refs/heads/*:refs/remotes/origin/*`
iii. `    url = https://github.com/<username>/hello6170.git`
iv. `[branch "master"]`
v. `    remote = origin`
vi. `    merge = refs/heads/master`
vii. These two sections help Git to “push” and “pull” code between the “master” branch on your local repository and the remote repository - “origin” - as shown below.

f. Create a new file and add it to the local repository
    i. `$ cd <new_repo>`
    ii. `$ echo ‘Hello World’ > helloworld.txt`
    iii. `$ git add helloworld.txt`
    iv. `$ git commit -a -m “First commit”`
    v. You just created a new file and added it to your local repository.
    vi. `$ git status`
    vii. `$ # On branch master`
    viii. `$ # Your branch is ahead of 'origin/master' by 1 commit.`
    ix. Git knows that you are one commit ahead of the remote repository!

a. To push your new files from the local repository to the remote repository
    i. `$ git push origin master`
    ii. `$ Username for 'https://github.com': <github_username>`
    iii. `$ Password for 'https://<uname>@github.com': <pwd>`
    iv. `$ git status`
    v. `$ # On branch master`
    vi. `$ nothing to commit (working directory clean)`
    vii. Now your remote repository and local repository are identical.
    viii. **Note:** Only those files which are commited will be pushed to the remote repository. If you have files which are added to the repository but not commited, they will not be pushed. It is strongly recommended that you commit all your changes before pushing code/files to the remote repository.

b. To pull the latest code and files from the remote repo into your local repo
    i. `$ git pull origin`
    ii. `$ Already up-to-date.`
    iii. If there were files which were new or updated by others, they will be copied to your local repository.
    iv. **Repeat Warning:** Git can get very confused if there are uncommitted changes in the files when you ask it to perform a merge. So make sure to commit whatever changes you have made so far before you merge. [1]

Next Tutorial

Ruby on Rails Basics

References

1. http://www.sbf5.com/~cduan/technical/git/git-1.shtml
2. https://help.github.com/articles/set-up-git