GWU CSCI 3411 - Fall 2019 - Lab - `git` Primer

James Taylor - Fall 2017

1 Introduction

Source Configuration Management (SCM) is critical to managing the complexity of developing and maintaining software.

In this course we will be using `git` for SCM and we will be using `github.com` as our cloud based host for both backup and submission. The terminology and examples presented in this document will be defined in terms of `git`.

2 Terminology

2.1 Repository

A repository encapsulates a project in a version control framework. A repository may be stored locally on one or more user machines and/or may be stored remotely on a web server. A number of applications and/or users can access the same `git` repository.

If a repository is maintained in a number of locations whether by one user or several users, the repository must be synchronized among all of the different changes. There are a number of possible mistakes that can occur if the repository is not properly synchronized and these mistakes can create a nasty mess to correct. Until you are well versed in using `git`, you should be very deliberate when issuing commands.

A repository can be created locally for existing code or cloned from a web server.

2.2 Remote

A remote is a repository stored on a remote web server. `git` allows for any number of `git` servers to host the same repository; however, it is important to remember that repositories are not guaranteed to have the same state or content. Updating a local repository does not automatically update remote repositories, and it is up to the repository users to synchronize each of the `git` servers by issuing commands.

2.3 Clone

To clone a repository is to copy the repository from a remote web server onto your local machine. When a repository is cloned, the remote `origin` of the repository is set as the URL of the web server. The `origin` references the default remote that the repository will communicate with when `push` or `pull` commands are issued.

Note: The `origin` is not automatically updated by changes to your local repository. Updates between local and remote repositories must be explicitly transmitted through specific commands.

2.4 Branch

A `git` project is not just a snapshot of the current state of a project's development; instead, it is a historical graph of all updates published to the repository. The graph is not strictly a tree as branches of the tree can merge back into other

branches.

A project may consist of a number of branches. A branch indicates a point or node within the hierarchy of updates set by the user as a reference point where the development diverges into an individual history to be maintained. The `master` branch is the default central branch of a project, but a `git` user can establish critical points within the set of updates where the development diverges into a unique history. Updates can be attached to a specific branch.

Branches allow new developments to be added to a project without affecting the main project. Practically, a branch allows further development of a project while also maintaining the working distribution without changes to one affecting the other. This means that multiple sub-projects based on the main project can proceed with new development without disturbing the main project. Each of these sub-projects can then be tested and then merged back into the main project or into individual sub-projects on their own schedules.

When multiple users are working in the same project on different tasks, it will become critical to properly use branches; however, even a single user can benefit from working with different branches. It is recommended that each student practice working with branches and merging branches on individual projects so that they develop the skills necessary to manage a more complex multi-user project later in the term.

2.5 Add

To add a file to a `git` repository is to stage an update of that file in its current state so that the file can be copied into the local repository. A file must be staged before it can be updated into the repository. Files may be staged one at a time or in bulk; however, a `git` user should take great care in staging files in bulk, as once a file is inserted into the repository, the file is always a part of the repository database.

2.6 Commit

To commit is to copy all files that have been staged in the user's working directory into the local repository database. Once a file has been committed, a copy of that file is inserted into the respository database.

2.7 Checkout

To checkout is to request the local `git` server overwrite the current version of files in the user's working directory with the specified files from the local repository. A checkout can request one or more files or a branch.

2.8 Merge

To merge is to request the local `git` server combine two branches into a single codebase. The git server will attempt to reconcile the differences between the files; however, a merge can often result in differences that are not automatically reconcilable. If differences cannot be automatically reconciled, a user will have to manually reconcile the differences and then commit the corrected codebase.

2.9 Push

To push is to request the local `git` server transmit the state of the local repository to a remote `git` server's repository.

2.10 Pull

To pull is to request the local `git` server to request the current state of the repository from a remote `git` server and to have the local `git` server update its local repository from the remote repository.

3 Commands

All of the following assumes that you will be working with a command line interface into `git`. While there are applications that can automate these functions for you, it is in your best interest to familiarize yourself with the console interface.

If a command contains a label in angled brackets, *i.e.* `<a_label>`, the label is not to be used literally; instead, an appropriate label should be substituted. Labels may be user defined or may be explicitly defined by the `git` system.

3.1 Getting help with `git`

The following command will list all of the top level `git` commands for reference:

```
git --help
```

To examine man page reference of each specific `git` command, follow the command with the `--help` parameter:

```
git <command> --help
```

3.2 Repository

To create a new repository in the current directory:

```
git init
```

To clone a remote repository to your local machine:

```
git clone <url>
```

3.3 Remote

To list both the remote repositories and their associated URL's:

```
git remote -v
```

3.4 Branch

To list the active branch in the current working directory:

```
git branch
```

To create a new branch with the name `<branchname>` that is a copy of the active branch:

```
git branch <branchname>
```

3.5 Add

To stage a single file with the name <filename> for commit:

```
git add <filename>
```

To stage one or more files at the path <pathspec> for commit:

```
git add <pathspec>
```

Note: File types can be automatically filtering out using `.gitignore`. It is debatable whether you should use the `*.*` wildcard to add all files in your working directory to a commit as this tends to temporary files that may not be vetted yet. You should not add source files to your source tree as these bloat the size of the repository. Remember that all the repository is the history of all changes, so adding large files that are later deleted from the working directory does not reduce the size of the repository. You should only ever add source files that are explicitly required to be maintained to the project.

3.6 Commit

To commit files to your local repository that have been added using `git add` use the `git commit` command. Each commit requires a commit message that will annotate and accompany the commit as a reference.
To commit the current set of staged files:

```
git commit
```

Note: The system will open the associated text editor for the user to add a commit message.

To commit the current set of staged files with the associated commit message <commitmessage>:

```
git commit -m "<commitmessage>"
```

3.7 Checkout

To overwrite a single file in the working directory with the name <filename> in the local repository:

```
git checkout <filename>
```

To overwrite all files from a path in the working directory with all files at the path <pathspec> in the local repository:

```
git checkout <pathspec>
```

To overwrite all files and folders in the working directory with the state of the branch named <branch> in the local repository:

```
git checkout <branch>
```

To copy the state of the current branch into a new branch named <new_branch> in the local repository:

```
git checkout -b <new_branch>
```

3.8 Push

To request the local `git` server transmit the state of the current branch to the remote `origin` server:

```
git push
```

To request the local `git` server transmit the state of the branch named `branch` to the remote server named `remote`:

```
git push <remote> <branch>
```

3.9 Pull

To request the local `git` server receive the state of the current branch from the remote `origin` server:

```
git pull
```

To request the local `git` server recieve the state of the branch named `branch` from the remote server named `remote`:

```
git pull <remote> <branch>
```

4 Example Workflows

4.1 Cloning a new repository

```
git clone <url>
```

4.2 Update a file and commit to the current branch of the local repository

```
git add <filename>
git commit -m "<commitmessage>"
```

4.3 Update a file and commit to the current branch of both local and `origin` repositories

```
git add <filename>
git commit -m "<commitmessage>"
git push
```

4.4 Pull (and merge) remote changes with your local changes from the `origin`'s `master` branch

```
git pull origin master
```

5 Resources

`https://git-scm.com/doc`