An Overview of Using `git`

Westminster College Computer Science Department

Greg Gagne

May 17, 2014

How to Create a Repository

Make a directory where your repository will be stored.

```
mkdir git-repo
```

and now `cd` to that directory.

Next, initialize the `git` repository:

```
git init
```

To determine the status of your repository, enter:

```
git status
```

Now, you must populate the repository with some files. Create the following 3 files in the `git-repo` directory:

```
file1.txt
```

```
file2.txt
```

```
file3.txt
```

How to Commit to the Repository

`git` distinguishes between **staged** and **unstaged** files. Only staged files can be committed to the repository. To stage a file, you must add it to the repository:

```
git add file1.txt file2.txt file3.txt
```

This could have been simplified with the command

```
git add -A
```

which adds all files to the staging.

Next, you must commit the contents of the staging area to the repository:

```
git commit -m ”Initial Commit”
```

The string following the `-m` option allows you to enter a string message describing the changes you are committing. In this instance, since it is only the initial commit, we will provide the appropriate message.

We may not want all files sent to the staging area and committed to the repository. In this instance, we can use a `.gitignore` file which specifies which files we **do not** want committed to the repository.

Branching and Merging

A git repository has a single **master** branch. From this master branch, you may create separate branches to try different features. The basic approach is that your master branch contains your working code, any separate **developer** branches contain features you are experimenting with. When code you are experimenting with has passed tests, you can then integrate – or **merge** – the development branch.

To create a branch named `dev` , enter the command

```
git branch dev
```

which creates the branch named `dev` . Next you must switch to that using using the `checkout` command:

```
git checkout dev
```

This switches your branch to the `dev` branch. Now, any work you perform in the `dev` branch is independent from the master branch. The `checkout` command allows you to switch between any branches you wish.

You can also merge your `dev` branch with the master by entering

```
git checkout master
git merge dev
```

This first switches you back to the master branch, and then merges any changes in the `dev` branch with the master branch.

Restoring Previous Versions from Repository

Each `git` commit associates an MD5 checksum with the transaction. This checksum allows you to revert to previous instances from the repository. For example, you may have modified a version of a file, and then realized your modified version no longer works. The easiest thing to do is restore the earlier version from the repository.

The following command outputs the log (the checksum associated with each commit):

```
git log
```

If you can determine the checksum associated with the previous commit, you can use that checksum to restore a file. Let's assume you wish to restore the file `file1.txt` and the checksum is `fc73a5a9e0b8c46c15afd2a656794d48621c4647`. You would enter

```
git checkout fc73a5a9e0b8c46c15afd2a656794d48621c4647 file1.txt
```

This will replace your local copy of `file1.txt` with the version (specified with the checksum) from the repository.

Restoring Previous Commit

If you wish to restore the state to the most recent previous commit (let's assume you had made several changes to files and wish to revert to the previous commit), enter:

```
git reset –hard
```

This restores the branch to the previous commit.

If you wish to restore to earlier commits, you would enter

```
git reset –hard // 1 commit back
git reset - -hard HEAD // 2 commits back
git reset –hard HEAD // 3 commits back
```

[THIS HAS TO BE CLARIFIED - LOOK AT `man git-reset`]

Working with a Remote Repository

A `git` repository can be deployed as a remote repository (i.e. github). Assuming the repository has been constructed. This assumes you have a github account established. Create a github repository – let's assume the name of that repository is `git-repo` and is associated with the user `greggagne` . The url for this repository appears as

```
https://github.com/greggagne/Hello-World.git
```

Once you have committed changes to your repository, you can then move that repo to github:

```
git remote add origin https://github.com/username/Hello-World.git
```

This specifies the name of the repository as the remote origin.

Now, you must **push** your local repo to github:

```
git push origin master
```

This sends your commits in your local repository to github.

Another collaborator can then **clone** the repository by entering

```
git clone https://github.com/username/Hello-World.git
```

If one makes changes to the contents of their local repository, they must add the changed file to the staging area, followed by committing the change. Changes are then pushed to the remote repository.

Once someone has cloned a remote repository, they can then get the latest changes by **pulling** from the repository:

```
git pull origin master
```

After cloning a repository, a good practice is to always first pull from that remote repository before making changes. This ensures you are always working with the latest updates from the remote repository.