Guide to Git

Git is a widely used version control system that allows us to create repositories and collaborate on projects. This guide will serve as a quick reference to the commonly used tools and commands of Git. For more in-depth documentation, please visit here.

This guide will cover how to:

- Create and use a repository
- Start and manage a new branch
- Commit your changes
- Work remotely
- Open and merge a pull request

At the end is a useful table of essential Git commands.

Create and Use a Repository

A **repository** is usually used to organize a single project. Repositories can contain folders and files, images, videos, spreadsheets, and data sets – anything your project needs.

To create a new repository:

1. In the upper right corner, next to your avatar or identicon, click and then select **New repository**.
2. Name your repository `hello-world`.
3. Write a short description.
4. Select **Initialize this repository with a README**.

Start and Manage a New Branch

Branching is the way to work on different versions of a repository at one time.

By default your repository has one branch named `master` which is considered to be the definitive branch. We use branches to experiment and make edits before committing them to `master`.

When you create a branch off the `master` branch, you're making a copy, or snapshot, of `master` as it was at that point in time. If someone else made changes to the `master` branch while you were working on your branch, you could pull in those updates.

To create a new branch:

1. Go to your repository.
2. Click the drop down at the top of the file list that says **branch: master**.
3. Type a branch name into the new branch text box.
4. Select the blue **Create branch** box.

Commit Your Changes

On GitHub, saved changes are called *commits*. Each commit has an associated *commit message*, which is a description explaining why a particular change was made. Commit messages capture the history of your changes, so other contributors can understand what you've done and why.

Make and commit changes:

1. Navigate to a file in your repo you wish to edit.
2. Click the pencil icon in the upper right corner of the file view to edit.
3. Make the desired changes.
4. Write a commit message that describes your changes.
5. Click **Commit changes** button.

These changes will be made on your current branch. If you are on a branch other than master, this branch now contains content that's different from master.

Work Remotely

Since you will all be working in teams, the best way to collaborate on a project is to **pull**, or **clone**, the project repository from Git, make changes remotely, commit those changes, and **push** the new changes to the project repository.

To clone a repository:
1. Copy the web URL of your repository by going to your repository, clicking the "Clone or Download" button, and copying the URL that pops up.
2. Open your terminal and run, "**`git clone <web URL of repo>`**".

To commit changes you've made remotely:
1. Open your terminal.
2. Run "**`git add -A`**". This adds all of your files to your commit. If you want to choose individual files, you can use the syntax, "**`git add <name of file>`**".
3. Run "**`git commit -m <message about commit>`**".

Finally, to push these changes to your repository on Git, run "**`git push`**".

Open and Merge a Pull Request

Pull Requests are the heart of collaboration on GitHub. When you open a *pull request*, you're proposing your changes and requesting that someone review and pull in your contribution and merge them into their branch. Pull requests show *diffs*, or differences, of the content from both branches. The changes, additions, and subtractions are shown in green and red.

As soon as you make a commit, you can open a pull request and start a discussion, even before the code is finished.

To open a pull request:

1. Click the Pull Request tab, then from the Pull Request page, click the green New Pull Request button.
2. In the **Example Comparisons** box, select the branch you wish to merge, to compare with master (the original).
3. Look over your changes in the diffs on the Compare page, make sure they're what you want to submit.
4. When you're satisfied that these are the changes you want to submit, click the big green **Create Pull Request** button.

Essential Git Commands

`git clone URL-or-path-to-repo`	Clone a repository. This makes a new folder in the current directory containing the files in the repository in which you can work on the code.
`git add`	Add the given file to the repository so that it will be tracked by git. Use this when you author a new code file and want to include it in a commit. Alternatively, include -a to add all files from your local repo
`git commit`	Finalize the current changes to your code as a commit to your current branch and repo on your local machine. Include -a to automatically add changed files that git is already tracking and -m <message> to include a message about the commit (otherwise you will be kicked to an editor in which to type out your message).
`git pull`	Pull any changes that have happened on the remote server you initially cloned and bring them into your local repository. You will

	need to have a clean repo so you will probably want to commit first if you have changes (or stash them). (NOTE: a git pull is a combination of a git fetch and a git merge which will lead to extra commits of the form Merge branch 'master' of .... For a cleaner history, use git pull --rebase)
`git push`	Push whatever commits you have made locally to the remote repository that you cloned previously. You may need to pull first to ensure that you are in sync with the repo.
`git log`	View history of commits that you have made. They will be displayed with their unique identifier (a big hex mess like 766f98f32fa...) and their commit message.
`git checkout -b <branch>`	The git checkout command lets you navigate between the branches created by git branch. Checking out a branch updates the files in the working directory to match the version stored in that branch, and it tells Git to record all new commits on that branch.

`git fetch`
 	
`git merge`
	When you `git fetch`, Git gathers any commits from the target branch that do not exist in your current branch and **stores them in your local repository**. However, **it does not merge them with your current branch**. This is particularly useful if you need to keep your repository up to date, but are working on something that might break if you update your files.
	
	To integrate the commits into your master branch, you use `git merge`.

`git diff`	View the difference between the current code and the latest commit. You can also enter two commit IDs to see the difference between the code at each of those commits.
`git reset <ID>`	Reset the state of your repo to that of a specific commit from the past.
`git stash`	Stash away the latest uncommitted changes. Later, you can run `git stash apply` to restore these changes.

Sources:
http://cs.brown.edu/courses/cs0320/docs/git-reference.html
https://guides.github.com/activities/hello-world/