Git

Git Configuration

```
[user]
    email = alex.brown7711@gmail.com    // Set user email
    name = Alex Brown           // Set user name

[merge]
    tool = vimdiff

[mergetool]
    prompt = true

[mergetool "vimdiff"]
    cmd = nvim -d $LOCAL $REMOTE

[core]
    editor = nvim
    autocrlf = input

[help]
    autocorrect = 1             // Enable fuzzy search on misspelled words

[color]
    ui = auto               // Enables colors when commands are ran in the console
```

Working Locally With Git

Viewing History and Diffs

`git log` show the commits made to the repository in reverse chronological order. If you want to see the differences between the commits, you can run

`git diff #######...#######`

However, if you don't want to work with the hashes, you can count back from head using `Head~#`, where `#` is a number that represents the commit that many numbers back. For example,

`git diff Head~5...Head`

compares the 5th commit back to the head. You can also shorten the syntax by using

`git diff Head~5...`

Git will assume you meant to compare with `HEAD`.

Undoing Changes In The Working Directory

If you want to undo changes made to a single file, you can run

`git checkout [file]`

and by default, it will grab the version from `HEAD` and replace your copy.

Undoing/Redoing Changes In The Working Directory

`git reset --hard`

Resets back to `HEAD`, but doesn't keeps all the changes that you made staged.

`git reset --soft`

Resets back to `HEAD`, but keeps all the changes that you made staged.

Cleaning The Working Copy

If you would like to clear out certain files that you don't need, such as log files or build artifacts.

```
git clean -n    // Displays what you would do (much like make -n)
git clean -f    // Cleans all shown in git clean -n
```

Ignoring files With .gitignore

If you don't want to include files, you can use a `.gitignore` to not commit/push those file types or directories.

```
// .gitignore

[dir]/      // ignores everything in this directory
*.txt       // Ignores all of this filetype
[file].txt  // Ignores this file
```

Working Remotely with Git

Basic Repository Statistics

In addition to `gitlog`, you can use `git log --oneline` to show all the commits on a single line. You can then find the number of commits using `git log --oneline | wc -l`.

If you want to look at the git log in graph form you can use the command `git log --oneline --graph`.

`git shortlog` gives that name of each all the authors, their commit messages, and the number of commits they have committed. There are a variety of flags that you can put such as `s` which is a summary (not commit messages), `n` ordered numerically by number of commits, and `e` shows their email address.

Viewing Commits

If you want to look at the last commit you use the command `git remote HEAD`, or something like `git remote HEAD~3` which looks 3 commits back from head. Or you can just use the hash values.

Pushing Tags to a Remote

To push tags to the remote you need to specify that you want to do that by saying `git push --tags`.

Branching, Merging, and Rebasing with Git

Visualizing Branches

A way to visualize the branch, you can did as we did before and run

`git log --graph --oneline`

If we want to visualize branches, you can add

`git log --graph --oneline --all --decorate`

If you would like to alias name, you can use

`git config --global alias.lga "log --graph --oneline --all --decorate`

So now if you type `git lga` it will run the command above.

Renaming and Deleting Branches

You can rename by moving the commit to the same spot like you do in the command line `git branch -m fix1 fix2`. To delete a branch you do `git branch -d [branch]`.

Recovering Deleted Commits

Although you may have deleted a commit, there is a way of getting it back. If you run `git reflog`, you can see all the points where head was pointing to. Now if you want to get back, you can run `git branch [name of branch you want to create] [hash value]`, and it will create a new branch with the name you specified.

Stashing Changes

If you are working on something and are not ready to commit them yet, but you need to change branches or just work on something else you can save that work by using `git stash`. By running a `git stash list` afterwards, you should see the items you stashed, and you can then checkout to other branches and do other things. To get the stash back, you can run `git stash apply`. Even if you run a `git reset --hard`, it will not get rid of the stash. However, if you run `git stash pop`, it will apply the item on the top of the stack and delete it.

If you don't to keep a stash, you can run `git stash drop`.

If you want to make a stash a branch you can run `git stash branch [name]`, it will take the item on the top of the stash, create a branch, check it out, and apply the stash.

Deleting a Remote Branch

If you want to create a remote branch, all you have to is push the branch by saying `git push origin [branch]`. You can also specify a new name for the remote branch by stating `git push origin [branch]:[remote branch name]` and the remote branch name will now have the name `[remote branch name]` with the content from `[branch]`.

If you want to delete a remote branch, you can do this by stating nothing in the local branch name. For example:

```
git push origin :[remote branch name]
```