Configuring git:
For all users on the system:
```
    git --config system
```

For user level:
```
    git --config global
```

Set username and email address:
```
  git config --global user.name "Robert Holland"
  git config --global user.email rob@robholland.com
```

Set the colors for git:
```
  git config --global color.ui auto
  git config --global color.ui true
```

Set the pager for git diff:
```
  git config --global core.pager 'less -R'
```

After updating my username and email address I had to reset the author by typing:
```
  git commit --amend --reset-author
  (can use this to reset the author for the latest commit)
```

If you want to see what is already configured:
```
  git config --list
```

If you want to see the username or email address:
```
  git config user.name
  git config user.email
```

To configure the preferred editor:
```
  git config --global core.editor "EditorNameHere"
  git config --global core.editor "mate -wl1" (wait for textmate to
  finish and put the cursor on line one).
  git config --system color.ui true
```

Exploring Git Auto-completion:
Download git-completion.bash and rename it to .git-completion.bash.

Download it from here:
```
  curl -OL
  https://github.com/git/git/raw/master/contrib/completion/git-comp
  tion.bash
```

Rename the file to .git-completion.bash
```
    mv git-completion.bash .git-completion.bash
```

Exploring Git Auto Completion:
Enter this in the .bash_profile or equivalent:
```
    #Git configuration file:
    if [ -f ~/.git-completion.bash ]; then
        source ~/.git-completion.bash
    fi
```

Create a new repository on the command line make a new directory then type:
```
touch README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/username/reponame.git
git push -u origin master
```

Push an existing repository from the command line

```
git remote add origin https://github.com/username/reponame.git
git push -u origin master
```

Stuff to try:
I had a difficult time pushing data to the git repo until I typed:
`git pull https://github.com/rlholland/gitnotes.git`
This merged the remote with my local. I was able to push afterward.

I have also deleted the remote repository after cloning it locally then
creating a remote empty repository and pushing back to it by typing
`git push --set-upstream origin master`

Git checkout
Undo changes to the git repository:
I changed a file and saved it but I have not staged (git add) it yet. Git status show what was changed and I want to undo my changes. To replace the file in the local directory with a copy of what you have in the working area type:

```
git checkout -- filename
```

This will replace the file that you changed with an original version before the change. If you only use git checkout filename (without the dashes), you may accidentally checkout a branch that has the same name as the file you are trying to restore (unlikely because branches don't have extensions). The double dashes -- tells git to stay in the current branch and look for the file you want to restore.

If you want to make another version of your files just make a branch.
To create a new branch type:

```
  git branch newbranchname
```

Git will automatically copy the master repo into your new branch.
Change into the new branch by typing:

```
  git checkout newbranchname

  git checkout -b newbranchname : Will create the new branch and
  change into it.
```

You can make changes without disturbing the master
files. If all goes well you can merge your changes into the master branch.

To merge your new branch into master, first go to the master branch by typing: git checkout master

then type:

```
  git merge branchname
```

```
The master branch will now have the same changes as branchname

To back out of a merge conflict type:
  git merge --abort
  This will leave your changes alone and not do the merge.

To update a branch with the latest changes from master:
You have two options:
  The first is a merge, but this creates an extra commit for the
  merge.
  Checkout each branch:
    git checkout branch1
  Then merge:
    git merge origin/master
  Then push:
    git push origin branch1

Alternatively, you can do a rebase:
  git fetch
  git rebase origin/master

If you want to change or revert the master branch to the previous
commit:
  Checkout previous commit on master
    git checkout abc123...
  While in the abc123... detached branch, create branch for new
  master
    git checkout -b new_master
  Delete old master
    git branch -D master
  Make new_master master
    git branch -mv new_master master
  Alternatively you can reset current branch to one commit ago on
  master
    git reset --hard abc123...

If you want to delete the branch (make sure you are on the master
branch) type:
  git branch -D branchname

If you want to see all of the branches:
  git branch -a
  git branch show-all <-- Doesn't seem to show all branches unless
  you have checked them out at some point.

If you want to see only the remote branches:
  git branch -r

If you want to see only local branches:
  git branch

If you want to see the branches along with the latest commit
message:
  git branch -v

How to find out if one branch has all of the commits of another
branch.
```

```
  git branch --merged
  (This will show a list of all of the branches that contain what
  is in your current branch. This will allow you to delete the othe
  matching branches without causing any harm).
```

Git diff
To see all of the changes made to the files type:

```
  git diff
```

You can see a line by line difference between what is in the repository or staging area compared to what is in the working area.
If you just want to see what is in a single file type:

```
  git diff singlefilename
```

If you have already staged a file and want to see the differences between that staged file and what you have in the repository then type:

```
  git diff --staged
```

Git blame:
To see all the changes line by line in a file and who made the change over time type:

```
  git blame -w filename
```

The -w does not show differences in whitespace.

To see the commits for only one file:

```
  git log --follow filename
```

Undo changes in the staging area:
I have made a change to a file and it shows up as changed in the working area. I used git add filename to add the file to the staging area. Now I want to take the file out of the staging area and put it back in the working area.

```
  git reset HEAD filename
```

You will see an "M" next to the file that indicates that the staging area was modified.

When you are in the diff view you can use the (minus sign + shift + S) to word wrap the long lines. Repeat the same key combination to undo the word wrap. Also you can see the changes side by side by typing:

```
  git diff --color-words filename
```

The -a option sends the commit directly to the repository skipping the staging area:

```
  git commit -am "Message"
```

You cannot use the "-am" option when committing modified files individually. This is used in situations where you have multiple files that have been modified but you only want to commit some of them. The "-a" will have to be left out and only use "-m".
Example:

```
  git commit -m "Commit message" "File1.txt" "File2.txt"
```

Amending Commits:
You can only amend the last commit because it doesn't have any more

commits after it. Once you have added another commit you can't amend a previous commit. If you need to change previous commit just make a new commit with the changes that you need.
I have already committed a file and then I make another change to the file and add it to the staging area. I want to commit the additional change into the previous commit:

```
  git commit --amend -m "Same message or different one."
```

Add a "Sign-off" to the commit type:

```
  git commit -s -am "Message"
```

To use GPG/PGP to sign a commit you must first configure GPG/PGP so that Git knows about it.

```
  git config user.signingkey <HEX KEY ID>
```

  or for global setting using the same key for every repository.

```
  git config --global user.signingkey <HEX KEY ID>
```

To GPG/PGP sign a commit type:

```
  git commit -S -am "Message" <--(Uppercase S for GPG signed
  commits).
```

Can use the same command to change the commit message.
You can amend a commit that is previous to the most recent commit. You will have to use git to checkout the commit using the SHA hash and then re-commit it with the new changes.

```
  git checkout 2d2323d23423d -- filename
```

This will put the file in the staging area and not the working area. When you checkout a file from the staging area it will go to the working area. (git diff --staged)

Git Revert:
Undo the changes made by a commit. It will take all of the changes and add everything that was deleted and delete everything that was added.

```
  git revert SHA
```

Git reset:
Moves the HEAD pointer. Similar to making a recording on a tape then rewinding 10 seconds.

Has 3 options.

```
  git reset --soft sha
```

Does not change the staging index or working directory. Only the repository is set back to an earlier version.

```
  git reset --soft sha

  git reset --mixed sha (default)
```

Changes the staging index to match the repository.
Does not change the working directory.
--mixed is almost as safe as --soft.

```
  git reset --hard sha
```

  changes staging index and working directory to match the repository. (Rewinding 10 seconds and pressing record.) It remove every change that was made before the commit that you select.

Git clean:
Remove unwanted files from your git directory.

```
    git clean -n (does a dry run)
    git clean -f (permanently deletes the untracked files from your
    directory)
    (see below for git rm --cached filename) to remove untracked
    files from being seen.
```

Git Ignore
Create a file in the root directory called .gitignore and place files and extensions in .gitignore that you want git to ignore.

```
    Can use regular expressions: * ? [aeiou] [0-9]
    Negate expressions with !
        Example: !index.php tells git to not ignore index.php.
    Ignore all files in a directory with a trailing slash.
        DirectoryName/DirectoryName/
    Comment with #
```

Global Ignore
Put all of your git ignore commands in one file and point git to it using git config --global. The file can be named anything you want. You just have to tell git where it is.

```
    git config --global core.excludesfile path/to/filename (Ex:
    .ignore_global)
```

Ignore files that have been previously tracked but you don't want to track them anymore. You can use the git rm command to delete the file from your directory and git will automatically remove it from being tracked by the repository. If you want to keep the file but remove it from being tracked you can use:

```
    git rm --cached filename
```

This removes the file from being tracked by git. If you do a git status it will show as "deleted" but it really isn't. It is just no longer being tracked.

Git does not track empty directories. If you want to track a directory you will have to put a file in it. Most commonly people put a file named .gitkeep inside the directory so that git will watch the directory.

Comparing Branches

```
  git diff master..newbranch
    (The order doesn't matter. The diff above matches the one
    below)
  git diff newbranch..master
```

A different view for diff.

```
  git diff --color-words newbranch..master
```

If you want to compare two branches but not the latest commit of one branch (the previous commit).

```
  git --color-words master..newbranch^
```

Rename branches:

```
git -m or git --move
git -m oldbranchname newbranchname
```

Using fast-forward vs true merge

Example: If you make a new branch from master and make changes to the new branch without making any changes to master, then merge t new changes into master, you are doing a fast forward merge. No n commit will need to be made because you are just adding more or t "updating" master.

If you make a new branch and make changes to the new branch and also make changes to master and do a merge then you are doing a true merge and you will have a merge conflict and will need to specify the differences that you want to keep or discard.

If you want to specify a merge tool to use when merging files you can do so by typing:

```
git mergetool --tool=NameOfMergeTool
```

Process Tracking (rebase)

Process tracking is when you merge the master branch into your working branch to update your working branch with new stuff from the master branch. This helps you reduce the number of merge conflicts that occur when you finally have to merge your changes back into master.

Stash/Stashing

A stash is not a commit and they do not have a SHA associated with them. You use "stash" when you are in one branch then try to checkout another branch without first saving the changes for the branch you are in. You will get a message telling you that you wi lose the changes that you have already made. You can save them by stashing them and then continue to checkout the other branch.

To stash type:

```
git stash save "Message for stash."
```

If you want to see a list of items in the stash type:

```
git stash list
```

You will see stash{0}, stash{1}, etc. It doesn't matter which branch you are on, you will still be able to see the stash and pu it out of the stash. This is helpful if you realize you are makin changes to the wrong branch. Just stash the changes and checkout the branch you need to apply the changes to and apply the stash.

If you want to see a particular stash, you will have to reference it by its number. For example: If I wanted to see what was in stash{0} I would type:

```
git stash show stash@{0}
```

To see the stash in a more detailed way (diff), you will have to use the "Patch" option. Type:

```
git stash show -p stash@{0}
```

To take the stash out of the stash repository you can use two commands,

```
git stash pop
git stash apply
```

The difference is that git stash pop will remove the stash from the stash repository and git stash apply will leave a copy in the stash repository.

Delete single items that are in the stash type:

```
  git stash drop stash{0}
```

Delete all items in the stash by typing:

```
  git stash clear
```

If you want to commit a file and add a message:

```
  commit --message="This is my message." "filename.txt"
```

If you want to see a particular commit type:

```
  git show [SHA Key]
```

If you want to see all of the branches on the remote type:

```
  git remote
  git remote -v (more verbose information)
```

If you want to add a remote repository type:

```
  git remote add <alias> <url>
  Example: git remote add origin
  https://github.com/rlholland/reponame.git
```

You don't have to use the name "origin" you can change it if you want.

If you want to change/update the URL in .git/config to point to a different repository:

```
  git remote set-url <alias> <url>
  Example:
  git remote set-url origin
  B:/Millennium/mPage/gitrepositories/azb_custom_components.git
```

If you want to remove a remote repository type:

```
  git remote rm remoterepositoryname
  Example: git remote rm origin
```

If you want to look at the remote SHA you can type:

```
  cat .git/refs/remotes/origin/master
```

This will show you the latest SHA on the remote

If you want to see what branches git is tracking type:

```
  cat .git/config
  git push -u origin master (The -u tells git to track this branch)
```

If you have a branch that is not tracking you can add it to be tracked by typing:

```
    git config branch.branchname.remote origin
    or
    git config branch.branchname.merge refs/heads/master
    or
    git branch --set-upstream branchname origin/branchname (This
    works with version 1.7 and later).
```

If you want to clone a remote repository into a folder that you name type:

```
  git clone https://github.com/rlholland/reponame.git
  newlocalfoldername
  You can also clone a specific branch by using the -b option.
  Example:
```

If you want to clone using SSH:

```
  git clone ssh://username@servername/absolute/path/to/git/repo.git
```

If you want to push all changes:

```
  git push --all "https://github.com/rlholland/reponame.git"
```

Git log
If you want to see the commits that were made to the git repo, type:

```
  git log
```

This will show you all of the commits and the messages that were entered when the commit was made. If you want to limit the display of message to a certain number then type:

```
  git log -n 10
```

This will limit the number of messages displayed on the screen to 10.

If you want to see all of the commits from the beginning up to a certain date type:

```
  git log --until=2014-05-25
```

If you want to see all of the commits since a certain date to the present type:

```
  git log --since=2014-05-25
```

You can use both commands together to see commits between two dates.

```
  git log --since="two weeks ago" --until="3 days ago"
  git log --since="two.weeks" --until="3.days"
```

You can see commits made by a specific person (author) by typing:

```
  git log --author="Robert" (can use quotes if search term has a
  space).
```

You can search the commit messages for an expression by using grep (Global Regular Expression). Type:

```
  git log --grep="Text you are looking for" (this is case
  sensitive).
```

You can ignore case with:

```
  git log --grep="Text you are looking for" -i  (the dash "i" will
  ignore case).
```

You can show the log file by SHA:

```
    git log SHA1..SHA2
    Example: git log 23fadf323..938533arad --oneline
    The --oneline shows part of the SHA and the  commit on one
    line.
    git log --format=oneline shows the complete SHA and message on
    one line.
```

You can show the log on just one file from a certain point and view the changes.

```
    git log -p SHA.. index.html
    git log -p --since="2014-04-01" index.html (The -p shows the
    differences)
```

To see the status or summary of the commits you can use:

```
    git log --stat
    git log --summary
    git log --stat --summary
```

Git can search diffs with the -S option (it's called pickaxe in the docs)
  http://stackoverflow.com/questions/4468361/search-all-of-git-hist
  ory-for-a-string

```
git log -Spassword
```

This will find any commit that added or removed the string password. Here are a few options:

  -p: will show the diffs. If you provide a file (-p file), it will generate a patch for you.
  -G: looks for differences whose added or removed line matches the given regexp, as opposed to -S, which "looks for differences that introduce or remove an instance of string".
  --all: searches over all branches and tags; alternatively, use --branches[=<pattern>] or --tags[=<pattern>]

To see a GPG signature:

```
  git log --show-signature
```

A good Git log command to show a lot of detail is:

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

If you want to see the log for a specific branch type:

```
  git log specificbranch --oneline -3 (the -3 shows the latest 3
  logs)
```

If you want to see the log differences in patch mode type:

```
  git log -p branchname..origin/branchname
```

If you only want to see the log entries for particurlar files type:

```
  git log -- foo.txt bar.txt
```

It's also possible to search for commits that introduce or remove a particular line of source code. This is called a pickaxe, and it takes the form of -S"<string>". For example, if you want to know when the string Hello, World! was added to any file in the project, you would use the following command:

```
  git log -S"Hello, World!"
```

If you want to search using a regular expression instead of a string, you can use the -G"<regex>" flag instead. This is a very powerful debugging tool, as it lets you locate all of the commits that affect a particular line of code. It can even show you when a line was copied or moved to another file.

Git Fetch

This synchronizes any remote branches we don't have locally so when you type git branch it will show you all of the branches.

If you want to fetch type:

```
git fetch origin
```

If you are tracking you don't need to type the "origin" just type:

```
git fetch
```

Three basic guidelines:

Always fetch before you work.
Fetch before you push
Fetch often
You can also use "git pull"

Git Pull

git pull does the same thing as git fetch except it automatically does the merge.

```
git pull = git fetch + git merge
```

Checkout Remote Branches
These commands will checkout the branch and track them.

```
git branch newbranchname HEAD
git branch newbranchname anycommitSHA
git branch newbranchname origin/branchname
```

To delete a remote branch use a colon.

```
git push origin :branchname
```

A little information history on the git push command. It used to be done like this: git push origin branchname:branchname. The col
between them means that you are telling git to push to origin the local branchname to the remote branchname. If you don't specify t
colon and you only have one branchname, git assumes they are the same.

So the git push origin :branchname means to push to origin nothing locally to the remote branchname. The remote branchname i
now getting nothing pushed to it and is deleted.

The new way to delete a branch is:

```
git push origin --delete branchname
```

If you want to modify someones elses code that you find on GitHub, you will have to "Fork" it then make your changes. Once you are done with your changes, you can create a "Pull Request" so that the original owner can look at your code and decide to incorporate it into theirs.

Configure the prompt to show the git branch when in a git repo:

```
export PS1='\W$(__git_ps1 "(%s)") > '
```

The \W will show the pwd information before the branch is displayed. When you are not in a git repo the pwd information wil
still show.

If you want to see the current prompt string "PS1" settings type

```
  "echo $PS1"
```

Configuring Git Aliases
You can create aliases for Git in the .gitconfig file two ways. You can edit the file directly or you can use the git config --global command.
For example:
I didn't want to type the entire log line below so I created a shortcut for it in the .gitconfig file by using the git config --global command.

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

  I created an alias for it named "logg" by typing:

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

  Now I only have to enter "git logg" to get the same output.

Re-install Git icons on Windows

```
  github --reinstall-shortcuts
```

Here are some more common Git aliases:

```
  git config --global alias.co checkout
  git config --global alias.cm commit
  git config --global alias.br branch
  git config --global alias.dfs "diff --staged"
  git config --global alias.logg "log --graph --oneline --decorate
  --all"
```

Merging Git Repositories:
  http://blog.caplin.com/2013/09/18/merging-two-git-repositories/

List of Gui Interfaces for Git can be found on the Git Wiki:
  https://git.wiki.kernel.org/index.php/InterfacesFrontendsAndTools

Git Hosting
There are two ways to host. Use a hosting company or host yourself.

  Popular Git hosting companies are:
    http://github.com
    http://bitbucket.org
    http://gitorious.org

  Self-Hosting:
    Gitosis - http://github.com/tv42/gitosis (development for this stopped a few years ago)
    Gitolite - http://github.com/sitaramc/gitolite

  Michael's Git Tutorial - Setting Up a Git Server
  https://www.youtube.com/watch?v=SyMkLQLC3Kg
  How to Setup a Git SSH Server and Client on Ubuntu
  https://www.youtube.com/watch?v=lXSZUuDW4nY
  Creating a Git Server on a Windows OS
  https://www.youtube.com/watch?v=w3eRlEhzAZk&ebc=ANyPxKpNRpbhQZ_nl
  IxaYeM_5rOLOI2RPJi2kv1jJuw6DUEEe14nsBvDRHvBjQKmO7DIvaOUzprWUCtFUX
  O9X-cLOpYrHg

Ubuntu Server Configuration:
To find the Gateway: route -n
On the server:

```
  sudo vi /etc/network/interfaces (Set the eth0 interface to a
  static IP address)
  auto lo
  iface lo inet loopback
  auto eth0
  iface eth0 inet static
  address 192.168.1.2
  netmask 255.255.255.0
  gateway 192.168.1.1

From the client:
  sudo vi /etc/hosts (enter the Git Server's information)
sudo vi /etc/ssh/sshd_config (Configure SSH to accept access by SHA
key instead of tunnelled clear text passwords)
  #Find the line that says: PasswordAuthentication yes
  #and change it to read: PasswordAuthentication no
  #uncomment the line.

sudo restart ssh

sudo adduser git
su git
mkdir .ssh
chmod 700 .ssh
touch .ssh/authorized_keys
chmod 600 .ssh/authorized_keys (Make sure only the root user has
access rights)

Create an RSA key on the client and copy it to the server.
  ssh-keygen -t rsa
  ssh-keygen -t rsa -b 4096
  (Copy id_rsa.pub to the server and import it into the Git
  authorized_keys file)

Another method using openssl:
  openssl genrsa -out mykey.pem 4096
  openssl rsa -in mykey.pem -pubout > mykey.pub

Install Git on the server:
  sudo apt-get -y install git

Change the git login to git operations only:
  cat /etc/shells

Find out which shell git is using:
  which git-shell

Copy the git-shell to the list of valid shells in the /etc/shells
file.

On Fedora 24 and 25, the chsh command may not exist. Install it:
dnf install util-linux-user

Change the git user login shell to git-shell:
  sudo chsh git
  Enter: /usr/bin/git-shell

Create a git repository:
```

```
mkdir -p /opt/git
Give the git user access to the git folder where the repositories
will live.
Use with caution! This will overwrite group shared repositories if
they already exist within the git repository.
  sudo chown -R git:git /opt/git

Create a git repository and a project:
  mkdir -p /opt/git/project-name.git
  cd /opt/git/project-name.git
  git init --bare
  git init --bare --shared (if the repository will be shared by a
  group)

If your repository is only going to be used by one user then you
can just use the git user for simplicity.
Change the owner of theprojectname.git to the git user and group.
  sudo chown -R git:git /opt/git/theprojectname.git
  sudo chmod o-rwx /opt/git/theprojectname.git (remove access to
  anyone else).

If your repository will be used by multiple people, create a group
for the Git repository.
Make sure the group has access to the repository:
      sudo chgrp -R thegroupname thereponame.git (-R set the group
      ownership recursively).
      sudo chmod g+rws thereponame.git (set the sticky bit so
      changes are owned by the group).
      sudo chmod o-rwx thereponame.git (remove access for "other"
      so only group members can clone).

On the client:
  Configure the git username and email.
  git config --global user.name "Joe Bob"
  git config --global user.email "joebob@joebob.com"

Create a local git repo called project-name
Initialize the git repo
  git init

Add some files and commit them:
  git add .
  git commit -m "initial commit"

Link your local repository to the server.
If you are using the git user (before pushing commits see
generating SSH keys below):
  git remote add origin git@gitserver/opt/git/project-name.git

If you are using your own username that is a member of the group
that has access to theprojectname.git:
  git remote add origin
  ssh://username@servername/opt/git/theprojectname.git

Push your files:
  git push origin master

After the initial push you only need to type:
  git push
```

Push all branches:

```
  git push origin '*:*'
  git push origin --all
  git push REMOTE '*:*'
  git push REMOTE --all
```

To automatically push all branches to their matching branch:

```
  git config --global push.default matching
```

To only push the current branch to its matching branch:

```
  git config --global push.default simple
```

If you are using the git user you will need to generate a public
and private SSH key for authentication (unless you know the
password for the git user).
To generate the SSH keys:
If there is no .ssh directory in your home directory you will have
to create one.

```
  mkdir .ssh
```

Change the permissions to 700.

```
  chmod -R 700 .ssh

  cd ~/.ssh
  ssh-keygen -t rsa
    -Accept the default name.
    -You can enter a password if you like. If you do not, you can
    take advantage of automatic authentication using your public SS
    key (id_rsa.pub) amd private SSH key (id_rsa) pair.
```

Copy the contents of your public key to the git users authorized
key file.

If you want to change your repository to point to a different
server and still keep the history.
  Pull everything from your soon-to-be-old-server into your local
  repository.
  Create the new repository on the new server.
  In your local repository update the URL path to the new server
  (git remote set-url origin ...). See above for example.
  "git push origin master" to the new location.
  The Git log should still show the history and all of your data
  will be on the new server.

I installed GitExtensions on my laptop and it would not push to a
UNC path until I pointed it to GitHub Desktop.
GitHub Desktop (Git Shell):

```
C:\Users\rlholland\AppData\Local\GitHub\PortableGit_d7effa1a4a32247
8cd29c826b52a0c118ad3db11\usr\bin
C:\Users\rlholland\AppData\Local\GitHub\GitHub.appref-ms
--open-shell
```

MINGW32 (Git Bash):

```
C:\Program Files\Git\usr\bin
"C:\Program Files\Git\git-bash.exe" --cd-to-home
```

Making Git auto-commit:
I'd like to use git to record all the changes to a file.

Is there a way I can turn git 'commit' on to automatically happen every time a file is updated - so there is a new commit for every change to a file?

Ideally I'd like my users to not even know that git is running behind the scenes. A user could then potentially "undo" changes to a file - and this could be achieved by pulling a previous version out of git.

On Linux you could use inotifywait to automatically execute a command every time a file's content is changed.
Edit: the following command commits file.txt as soon as it is saved:

```
inotifywait -q -m -e CLOSE_WRITE --format="git commit -m
'autocommit on change' %w" file.txt | sh
```

The earlier inotifywait answer is great, but it isn't quite a complete solution. As written, it is a one shot commit for a one time change in a file. It does not work for the common case where editing a file creates a new inode with the original name. inotifywait -m apparently follows files by inode, not by name. Also, after the file has changed, it is not staged for git commit without git add or git commit -a. Making some adjustments, here is what I am using on Debian to track all changes to my calendar file:

/etc/rc.local:

```
su -c /home/<username>/bin/gitwait -l <username>
```

/home/<username>/bin/gitwait:

```
#!/bin/bash
#
# gitwait - watch file and git commit all changes as they happen
#

while true; do

  inotifywait -qq -e CLOSE_WRITE ~/.calendar/calendar

  cd ~/.calendar; git commit -a -m 'autocommit on change'

done
```

This could be generalized to wait on a list of files and/or directories, and the corresponding inotifywait processes, and restart each inotifywait as a file is changed.

Use the -r flag to inotifywait, but note that the kernel has a limit on the number of inotifywait watches it can set up. man inotifywait will tell you more.

This Github repository was recommended:
https://github.com/nevik/gitwatch.git. I have cloned it in my personal repository.
Source:

http://stackoverflow.com/questions/420143/making-git-auto-commit.

The index.lock issue:

I was getting an error when made a commit. I can't remember what the error was but I had a choice to answer "Y" or "N" to retry. I selected "N" and then was taken back to the command prompt. I looked on StackOverflow.com and saw a post that someone had submitted about creating the .git/index.lock file and trying the commit again and if it didn't work. Delete the index.lock file which seemed to work for the person that made the post. I tried i and it worked for me.

Index.lock issue cannot write to index:
While the prompt is still showing "Do you want to retry Y/N", open another command prompt and go into the .git directory and copy the index.lock file to index.lock.bak.

```
  type index.lock > index.lock.bak
  cat index.lock > index.lock.bak
```

Go back to the original command prompt and type "N" to not retry.
Go back to the other command prompt and copy index.lock.bak into index.

```
  type index.lock.bak > index
  cat index.lock.bak > index

  del index.lock
  rm index.lock
```

Type git status/log to see the changes.