Initializing a Git repository (existing directory/project)

If you have an existing directory or project and want to create a Git repository from it, do something like this:

```
$ git init
$ git add *

OR

$ git add .
```

and then:

```
$ git commit -m 'initial commit'
```

Note that your Git project directory will now have a .git subdirectory. (Explore that directory for more information.)

Git - Ignoring files with .gitignore

Before adding everything to your repository, it may be helpful to know .gitignore:

```
$ cat .gitignore

# ignore java class files
*.class
```

```
# ignore binary/compiled files
*.[oa]

# ignore temporary files
*~
```

Some .gitignore pattern/naming rules:

- Blank lines are ignored
- Lines beginning with '#' are ignored
- Standard glob patterns work (**.a,** .o)
- Can specify root directory files like '/TODO'
- End patterns with a slash to specify a directory (bin/)
- Negate a pattern by beginning it with an '!'

Another Git init example (Github)

Here's a Git init example from Github, showing how to create and push a Git repository to Github:

```
$ mkdir example
$ cd example
$ git init
$ touch README
$ git add README
$ git commit -m 'first init'
```

```
$ git remote add origin git@github.com:alvinj/example.git
$ git push origin master
```

How to push an existing Git repository to Github:

```
$ cd example
$ git remote add origin git@github.com:alvinj/example.git
$ git push origin master
```

Git checkout - Cloning a Git repository

There isn't really a "git checkout" command. Instead, you "clone" a Git repository:

```
# clone a git repo from github
$ git clone git@github.com:username/reponame.git

# one of my projects
$ git clone git@github.com:alvinj/Cato.git
```

Note: If you clone a repository, this command automatically adds that remote repository under the name "origin".

Renaming a repository/directory while cloning:

```
$ git clone git@github.com:alvinj/Cato.git cato
```

Git clone command using SSH:

```
$ git clone ssh://username@hostname:ssh-port/path/to/foobar.git
```

the SSH port isn't required, just leave it off:

```
$ git clone ssh://username@hostname:/path/to/foobar.git
```

If you don't specify a protocol, Git assumes SSH:

```
$ git clone username@hostname:projectname.git
```

Adding files to a Git repository

Ways to add a file to an existing Git repository:

```
# create new file
$ touch README

# add file to staging area
$ git add README
```

```
# commit the file
$ git commit -m 'yada yada'
```

A faster/easier way is to skip the Git staging area:

```
$ touch README
$ git commit -a -m 'yada yada'
```

Git status and diff commands

```
$ git status
```

```
$ git diff
```

```
$ git diff --cached
```

Git - Removing files

How to remove files from Git.

```
# TODO is this step necessary?
$ rm README

$ git rm README
$ git commit -m 'yada yada'
```

There is also a Git remove cached option:

```
$ git rm --cached README
```

See Pro Git Chapter 2 for more "Git remove" information.

Git - Moving files

Git is smart about files that have been moved/renamed without using "git mv", but you can do this for clarity/obviousness:

```
$ git mv README readme.txt
```

TODO - more here

Git push (sharing changes)

To share your commits with others, you need to push your changes back to the remote repository. The basic command is:

```
$ git push [remote-name] [branch-name]
```

As you saw earlier, a git push command to push your master branch to an origin server looks like this:

```
$ git push origin master
```

Note from Pro Git:

> This command works only if you cloned from a server to which you have write access and if nobody has pushed in the meantime. If you and someone else clone at the same time and they push upstream and then you push upstream, your push will rightly be rejected. You'll have to pull down their work first and incorporate it into yours before you'll be allowed to push.

Git remote commands

From Pro Git:

> Remote repositories are versions of your project that are hosted on the Internet or network somewhere. You can have several of them, each of which generally is either read-only or read/write for you.

To see which remote servers you have configured, run this command from within a Git directory:

```
$ git remote
origin
another-server
yet-another-server
```

Or for more detailed git remote information:

```
$ git remote -v
[remote-name]  [remote-url]
```

You can use a command like this to inspect a Git remote:

```
$ git remote show origin
```

where `origin` is more generally the name of your remote:

```
$ git remote show [remote-name]
```

"git update" - Getting data from remote repositories (git fetch, git pull)

To pull the latest changes from a git repository into your local already-existing repository, use the "git pull" or "git fetch" commands. I *think* you use git pull to get the latest changes from the repository and have them automagically merged (or possibly to overwrite) your local files (TODO - research this):

```
$ git pull
```

or

```
$ git pull [remote-name]
```

To pull down all the data from a remote project that you don't have, so you can then do a manual merge, I *believe* git fetch is the correct command:

```
$ git fetch
```

or

```
$ git fetch [remote-name]
```

Note from Pro Git:

> It's important to note that the fetch command pulls the data to your local repository — it doesn't automatically merge it with any of your work or modify what you're currently working on. You have to merge it manually into your work when you're ready.

I think the "git fetch" workflow here looks something like this, but I still have a lot to learn here:

```
$ git fetch origin
$ git log -p master origin/master
# manually check and adjust the differences
$ git merge origin/master
```

Please see the online Git book for more information here, I have only done basic "git fetch" exercises at this point, and typically use "git pull" (which is like a "cvs update" or "svn update", because I'm the only one working on my current projects).

You can also rename remote Git repositories. See the Pro Git book.

Git status (like 'cvs status' or 'svn status')

If you need to see the status of your local Git repository compared to the repository on a remote server named origin, you can use this command:

```
$ git remote show origin
```

If everything is up to date, this will show output like this:

```
Fetch URL: ssh://user@host:port/path/to/git/myapp.git
Push  URL: ssh://user@host:port/path/to/git/myapp.git
HEAD branch: master
Remote branch:
  master tracked
Local branch configured for 'git pull':
```

```
  master merges with remote master
Local ref configured for 'git push':
  master pushes to master (up to date)
```

This seems similar to the "cvs status" and "svn status" commands, where you're comparing your local sandbox/repository to the server's repository.

Git tagging

See the Pro Git tagging chapter for information on Git tagging.

Git branching

See the Pro Git Branching chapter for Git branching information.

Git help

Here's what the Git help output looks like on my current Mac system:

```
Config file location
    --global              use global config file
    --system              use system config file
    -f, --file <FILE>     use given config file

Action
    --get                 get value: name [value-regex]
    --get-all             get all values: key [value-regex]
    --get-regexp          get values for regexp: name-regex [value-regex]
```

```
    --replace-all         replace all matching variables: name value [value_regex]
    --add                 adds a new variable: name value
    --unset               removes a variable: name [value-regex]
    --unset-all           removes all matches: name [value-regex]
    --rename-section      rename section: old-name new-name
    --remove-section      remove a section: name
    -l, --list            list all
    -e, --edit            opens an editor
    --get-color <slot>    find the color configured: [default]
    --get-colorbool <slot>
                          find the color setting: [stdout-is-tty]

Type
    --bool                value is "true" or "false"
    --int                 value is decimal number
    --bool-or-int         value is --bool or --int
    --path                value is a path (file or directory name)

Other
    -z, --null            terminate values with NUL byte
```

Just type "git config" or something similar to see this Git usage statement.

You can also type commands like these:

```
$ git help
$ git help [command]
$ git [command] --help
$ man git-[command]
```