Key Concepts

repository
: A collection of files and their combined version history. Typically one repository per project. Created with `git init`, or checked out with `git clone`.

commit
: A collection of related changes to a repository, as made by `git commit`.

staging area
: Also referred to as the *index*, a set of uncommitted changes a repository. Managed with `git add/restore`.

remote
: A copy of your git repository that has been *pushed* to another server. e.g. GitHub. Managed with `git remote/push/pull`

Getting Started

`git init` creates a new git repository from a directory of untracked files. (You will still need to use `git add` to add the files you wish to track.)

`git clone` clones a remote repository to your local machine.

```
git clone git@github.com:capp-camp/quickrefs.git
```

Making Local Commits

`git add {files}` adds one or more files to the *staging area*, to be included in next commit.

`git status` prints the status of your current repository, including what files are modified, and what is already staged using `git add` for the next commit.

`git restore {files}` restores changed files to their prior version. **This is a destructive command and you can lose uncommitted work.**

`git commit -m {message}` creates a new commit with all files currently in staging area. A message should be passed to indicate the nature of the changes.

Example:

```
$ code part1.py     # make edits to part1.py as needed
$ git add part1.py  # adds part1.py to staging area
$ code part2.py     # make edits to part2.py (unwanted)
$ git restore part2.py   # revert part2.py changes
$ git status        # check on status of staging area
$ git commit -m "add solution for part 1"
```

Working With Remotes

`git push` sends local commits (but not staged/unstaged changes!) to remote server.

`git pull` gets commits from remote server and updates local copy.

`git remote -v` shows what remote(s) are configured for this repository.

Viewing Changes

`git diff` can be used to view changes between different revisions. There are **a lot** of different options, use `tldr git diff` or `git diff --help` to see more examples.

Common usage:
`git diff` with no arguments will show *unstaged* changes.

`git diff {filename}` will show *unstaged* changes to a given file.

`git diff --staged` will show *staged but uncommitted* changes.

`git diff --summary {commit}` will show a summary of changes since a given commit. Substitute a commit ID from `git log` for `{commit}`.

`git log` will give a list of all commits:

```
$ git log
commit 878b2c740f3481ddcfc8a5b35e53830cab69e5bb (HEAD -> main)
Author: James Turk
Date:   Fri Aug 23 15:01:41 2024 -0400

    generate more example images

commit 8707cb5c808c949846112c4ca300c6ccdb04786e
Author: James Turk
Date:   Tue Aug 20 15:29:38 2024 -0400

    update readme

commit b692490a41489f3e79b05b086ecf6cc01f5ced5a
Author: James Turk
Date:   Tue Aug 20 15:24:31 2024 -0400

    add casts
```

These long ids (`b692490a41489f3e79b05b086ecf6cc01f5ced5a`) can be used to reference specific commits from many git commands. You can typically use the first 4-5 characters. e.g. `git diff --summary b692`