Basic - Make

EECS 201 Fall 2022

Submission Instructions

This assignment will be submitted as a repository on the EECS GitLab server. Create a private, blank, **README-less (uncheck that box!)** Project on it with the name/path/URL `eecs201-basic-make` and add `brng` as a Reporter. The submission branch will be `make`. If this branch is not already the default initial branch, you initialize the local repo with an additional argument: `git init --initial-branch=make` if your version of Git is recent enough. Otherwise you can create a branch with this name after your first commit.

Preface

In this assignment you'll be provided yet another zipped archive containing some starter files.

`https://www.eecs.umich.edu/courses/eecs201/fa2022/files/assignments/basic-make.tar.gz`

```
/
|-- report.txt
|-- 1
  |-- Makefile
|-- 2
  |-- Makefile
|-- 3
  |-- Makefile
|-- 4
  |-- Makefile
```

Note that this assignment is to be submitted on a remote branch called `make`. Initialize a Git repository **inside of the extracted** `basic-make` **directory**; as noted above `git init -b <branch name>` or `git init --initial-branch <branch name>` can initialize the repo with a different branch name (e.g. `make` as per the submission instructions). If your version of Git is too old for these options, you could create the `make` branch afterwards after your first commit, or you can set the local branch's tracking information manually. Create a file called `report.txt` in this directory. Add all of the present files and commit them.

Create a **private** project named `eecs201-basic-make` on the UMich GitLab (`gitlab.umich.edu`) and add the instructor `brng` as a **Reporter**. Set this UMich GitLab project as your remote: you'll be pushing to it in order to submit.

In this assignment you will be incrementally building up more complex Makefiles. First, we'll start at the beginning.

1 A fresh start

1. `cd` into the `1` directory.
2. Create a file named `Makefile`.
3. Create a rule with a target called `all` that has this for a recipe:
   `gcc -o nocat nocat.c`
4. Create a rule with target called `clean` that has this for a recipe:
   `rm -f nocat`
5. Move the `all` rule to the top so that it'll run by default when `make` is run without a target specified. (There's actually another way to do this without moving the position: I'll leave this as a personal exercise ;)

6. Make sure that your Makefile works correctly.
7. Add and commit `Makefile`.

2 Phonies

1. `cd` into the `2` directory.
2. Take a look at the Makefile and note the existing rules.
3. Try running `make all`, `make clean`, and `make test`.
4. Note that their recipes do not run.
5. Fix the Makefile so that each of the rules can run their recipes.

3 Dependencies

1. `cd` into the `3` directory.
2. Take a look at the Makefile. Note what each target in the Makefile requires which file.
3. Edit the Makefile so that each target has the proper prerequisites. One way to test this is to run `$ make clean` to delete any (one or all) intermediate build files, and then run `$ make <some target>` (e.g. `$ make nocat`) to build one of the targets. The build should succeed as Make will proceed to build any missing intermediate files, and the only files that should be built are the ones that are missing.
4. The code does compile. Ignore any **warnings** that the compiler prints out: these are just warnings and are not errors.

4 Not repeating yourself

1. `cd` into the `4` directory.
2. Create a file named `Makefile`.
3. Edit the `Makefile` so that:
   - It has a `CC` variable that is set to `gcc`
     This variable represents which C compiler to use.
   - It has a `BIN` variable that is set to `sum30`
     This variable represents what the output executable binary is named.
   - It has a `SRCS` variable that contains all the `.c` files under the `src` directory. This list should not be hardcoded (for this assignment): if a new `.c` file is added, the Makefile should not have to be edited to include it.
   - It has a rule to build the output executable binary. Don't worry about object code. Note that the `-o` flag for `gcc` sets the output name. This rule should have `SRCS` as a prerequisite. In addition, the recipe should make use of existing variables and **automatic variables** that refer to the target and prerequisites to avoid repeating yourself (e.g. you shouldn't be referring to `SRCS` or `BIN` in the compilation command).
   - There is a phony `all` target that builds the output executable binary.
   - There is a phony `clean` target that removes the output executable binary.
   - The `all` target should run when `$ make` is run (without a target specified).

5 Conclusion

1. Add and commit any changes you intend to submit.
2. Fill out the `report.txt` file in the following steps:
3. On the first line provide an integer time in minutes of how long it took for you to complete this assignment. It should just be an integer: no letters or words.
4. On the second line and beyond, write down what you learned while doing this assignment. If you already knew how to do all of this, put down "N/A".
5. Commit your `report.txt` file and push your commits to a branch called `make` on your remote. **Ignore GitLab's talk about merge requests if you happen to have a `main` remote branch**.
6. Remember that as a GitLab assignment, the autograder is available!