Exercises

Most makefiles provide a target called `clean`. This isn't intended to produce a file called `clean`, but instead to clean up any files that can be re-built by make. Think of it as a way to "undo" all of the build steps. Implement a `clean` target for the `paper.pdf` `Makefile` above. You will have to make the target phony. You may find the `git ls-files` subcommand useful. A number of other very common make targets are listed here.

Before implementing 'clean' recipe:

```
simple-pdf-builder$ ls
data.dat  paper.aux  paper.pdf  plot-data.png  README.md
Makefile  paper.log  paper.tex  plot.py
# list tracked files
simple-pdf-builder$ git ls-files
Makefile
README.md
data.dat
paper.tex
plot.py
# list untracked files (generated files that can be rebuilt)
$ git ls-files --others
paper.aux
paper.log
paper.pdf
plot-data.png
```

Implement 'clean' recipe to remove generated secondary files:

```
simple-pdf-builder$ vim Makefile
# specify target 'clean'
.PHONY: clean

paper.pdf: paper.tex plot-data.png
	pdflatex paper.tex
```

```
plot-%.png: %.dat plot.py
	./plot.py -i $*.dat -o $@

# recipe to remove generated secondary files
clean:
	@# deletes untracked files
	@git ls-files --other | xargs rm

$ make clean && ls
data.dat  Makefile  paper.tex  plot.py  README.md
```

Take a look at the various ways to specify version requirements for dependencies in Rust's build system. Most package repositories support similar syntax. For each one (caret, tilde, wildcard, comparison, and multiple), try to come up with a use-case in which that particular kind of requirement makes sense.

Specifying version requirements for dependencies:

Caret requirements

> allow updates if the new version does not modify the left-most non-zero digit in the major, minor, patch grouping
>
> Example:
>
> ^1.2.3 := >=1.2.3, <2.0.0
>
> Use case:
>
> Minor version updates and latest patches (e.g. bug fixes or a new minor feature) of dependencies not heavily used should be compatible. Major upgrades might cause issues or perhaps even break the project.

Tilde requirements

> specify a minimal version with some ability to update
>
> Example:
>
> ~1.2.3 := >=1.2.3, <1.3.0

Use case:

May be used when there is a strict constraint over major and minor updates to the dependency. Only bug fixes and performance or security improvements are allowed.

Wildcard requirements

allow for any version where the wildcard is positioned

Example:

* := >=0.0.0

Use case:

May be used if the dependency is fairly stable and not heavily used by the project.

Comparison requirements

manually specify a version range or an exact version to depend on

Example:

= 1.2.3

Use case:

For critical and unstable dependencies, you may want to lock a particular version to make sure it functions as expected.

Multiple requirements

Example:

>= 1.2, < 1.5

Use case:

If a dependency works fine until a given version or if you are confident that it is compatible in a given version range, you may use multiple requirements.

Git can act as a simple CI system all by itself. In `.git/hooks` inside any git repository, you will find (currently inactive) files that are run as scripts when a particular action happens. Write a `pre-commit` hook that runs `make paper.pdf` and refuses the commit if the `make` command fails. This should prevent any commit from having an unbuildable version of the paper.

```
$ vim .git/hooks/pre-commit
#!/bin/sh
#
# A pre-commit hook to prevent commit if make command fails

# redirect output to stderr.
exec 1>&2

if ! make; then
        cat <<\EOF
Error: Failed to make pdf
EOF
        exit 1
fi
```

Test pre-commit hook

```
$ vim plot.py
# messed with the code
$ git add plot.py && git commit -m "Added colors and labels"
./plot.py -i data.dat -o plot-data.png
Traceback (most recent call last):
  File "./plot.py", line 13, in <module>
    plt.plot(data[:, 0], data[:, 1])
NameError: name 'plt' is not defined
Makefile:8: recipe for target 'plot-data.png' failed
make: *** [plot-data.png] Error 1
Error: Failed to make pdf
```