Once upon a time there were no web browsers, file browsers, start menus, or search bars. When somebody booted up a computer all they got a was a shell prompt, and all of the work they did started from that prompt. Back then people still loved to share software, but there was always the problem of how software should be installed. The make program is the best attempt at solving this problem, and make's elegance has carried it so far that it is still in wide use today. The guiding design goal of make is that in order to install some new piece of software one would:

1. Download all of the files required for installation into a directory.
2. cd into that directory.
3. Run make.

This is accomplished by specifying a file called makefile, which describes the relationships between different files and programs. In addition to installing programs, make is also useful for creating documents automatically. Let's build up a makefile that creates a readme.txtfile which is automatically populated with some information about our current directory.

Let's start by creating a very basic makefile with nano:

```
cd ~/Documents/Journal
nano makefile

draft_journal_entry.txt:
  touch draft_journal_entry.txt
```

The simple makefile above shows illustrates a **rule** which has the following general format:

```
[target]: [dependencies...]
  [commands...]
```

In the simple example we created draft_journal_entry.txt is the target, a file which is created as the result of the command(s). It's very important to note that any commands under a target must be indented with a Tab. If we don't use Tabs to indent the commands then make will fail. Let's save and close the makefile, then we can run the following in the console:

```
ls

## makefile
```

Let's use the make command with the target we want to be "made" as the only argument:

```
make draft_journal_entry.txt

## touch draft_journal_entry.txt

ls

## draft_journal_entry.txt
## makefile
```

The commands that are indented under our definition of the rule for the draft_journal_entry.txt target were executed, so now draft_journal_entry.txt exists! Let's try running the same make command again:

```
make draft_journal_entry.txt

## make: 'draft_journal_entry.txt' is up to date.
```

Since the target file already exists no action is taken, and instead we're informed that the rule for draft_journal_entry.txt is "up to date" (there's nothing to be done).

If we look at the general rule format we previously sketched out, we can see that we didn't specify any dependencies for this rule. A dependency is a file that the target depends on in order to be built. If a dependency has been updated since the last time make was run for a target then the target is not "up to date." This means that the commands for that target will be run the next time make is run for that target. This way, the changes to the dependency are incorperated into the target. The commands are only run when the dependencies or change, or when the target doesn't exist at all, in order to avoid running commands unnecessarily.

Let's update our makefile to include a readme.txt that is built automatically. First, let's add a table of contents for our journal:

```
echo "1. 2017-06-15-In-Boston" > toc.txt
```

Now let's update our makefile with nano to automatically generate a readme.txt:

```
nano makefile
draft_journal_entry.txt:
  touch draft_journal_entry.txt

readme.txt: toc.txt
  echo "This journal contains the following number of entries:" > readme.txt
  wc -l toc.txt | egrep -o "[0-9]+" >> readme.txt
```

Take note that the -o flag provided to egrep above extracts the regular expression match from the matching line, so that only the number of lines is appended to readme.txt. Now let's run make with readme.txt as the target:

```
make readme.txt

## echo "This journal contains the following number of entries:" > readme.txt
## wc -l toc.txt | egrep -o "[0-9]+" >> readme.txt
```

Now let's take a look at readme.txt:

```
cat readme.txt

## This journal contains the following number of entries:
## 1
```

Looks like it worked! What do you think will happen if we run make readme.txt again?

```
make readme.txt

## make: 'readme.txt' is up to date.
```

You guessed it: nothing happened! Since the readme.txt file still exists and no changes were made to any of the dependencies for readme.txt (toc.txt is the only dependency) make doesn't run the commands for the readme.txt rule. Now let's modify toc.txt then we'll try running make again.

```
echo "2. 2017-06-16-IQSS-Talk" >> toc.txt
make readme.txt

## echo "This journal contains the following number of entries:" > readme.txt
## wc -l toc.txt | egrep -o "[0-9]+" >> readme.txt
```

Looks like it ran! Let's check readme.txt to make sure.

```
cat readme.txt

## This journal contains the following number of entries:
## 2
```

It looks like make successfully updated readme.txt! With every change to toc.txt, running make readme.txt will *programmatically* update readme.txt.

In order to simplify the make experience, we can create a rule at the top of our makefilecalled all where we can list all of the files that are built by the makefile. By adding the all target we can simply run make without any arguments in order to build all of the targets in the makefile. Let's open up nano and add this rule:

```
nano makefile

all: draft_journal_entry.txt readme.txt

draft_journal_entry.txt:
  touch draft_journal_entry.txt

readme.txt: toc.txt
  echo "This journal contains the following number of entries:" > readme.txt
  wc -l toc.txt | egrep -o "[0-9]+" >> readme.txt
```

While we have nano open let's add another special rule at the end of our makefile called clean which destroys the files created by our makefile:

```
all: draft_journal_entry.txt readme.txt

draft_journal_entry.txt:
  touch draft_journal_entry.txt

readme.txt: toc.txt
  echo "This journal contains the following number of entries:" > readme.txt
  wc -l toc.txt | egrep -o "[0-9]+" >> readme.txt

clean:
  rm draft_journal_entry.txt
  rm readme.txt
```

Let's save and close our makefile then let's test it out first let's clean up our repository:

```
make clean
ls

## rm draft_journal_entry.txt
## rm readme.txt
## makefile
## toc.txt

make
ls

## touch draft_journal_entry.txt
## echo "This journal contains the following number of entries:" > readme.txt
## wc -l toc.txt | egrep -o "[0-9]+" >> readme.txt
## draft_journal_entry.txt
## readme.txt
## makefile
## toc.txt
```

Looks like our makefile works! The make command is extremely powerful, and this section is meant to just be an introduction. For more in-depth reading about make I recommend Karl Broman's tutorial or Chase Lambert's makefiletutorial.com.

Summary

- make is a tool for creating relationships between files and programs, so that files that depend on other files can be automatically rebuilt.
- makefiles are text files that contain a list of rules.
- Rules are made up of targets (files to be built), commands (a list of bash commands that build the target), and dependencies (files that the target depends on to be built).

Mark as completed