Chapter 1

Makefile Examples

This chapter includes examples of Makefiles included in the practical data. We assume that you have unzipped the practical data into a directory somewhere in your local environment. Once you have done that, you should set an environment variable called `MAKEPIPELINES` to refer to this directory. In the command below we assume you have placed it in your home directory:

```
$ export MAKEPIPELINES=~/makepipelines
```

The examples included below will reference this environment variable as necessary to find the correct files.

Running FreeSurfer

This is an example of how to use a makefile to execute FreeSurfer's longitudinal pipeline. Note that FreeSurfer itself is a large pipeline built using `make`. However, we do not need to know that if we treat the program `recon-all` as a single executable and show how to use `make` to call it. Here, the Makefile functions more as a way to permit parallel execution of `recon-all` rather than a way to track dependencies.

The code for this example is in `oasis-longitudinal-example-small/freesurfer/Makefile`.

```
PROJHOME=$$MAKEPIPELINES/oasis-longitudinal-sample-small
❶ SUBJECTS_DIR=$(PROJHOME)/freesurfer

QA_TOOLS=/usr/local/freesurfer/QAtools_v1.1
FREESURFER_SETUP = /usr/local/freesurfer/stable5_3/SetUpFreeSurfer.sh
RECON_ALL = /usr/local/freesurfer/stable5_3/bin/recon-all $(RECON_FLAGS)
❷ RECON_FLAGS = -use-mritotal -nuintensitycor-3T -qcache -all  -notal-check

❸ SHELL=/bin/bash
```

❶ FreeSurfer normally likes to work with all subjects in a single directory. We set the `make` variable `PROJHOME` for convenience, and the `SUBJECTS_DIR`.

❷ Because we have multiple versions of FreeSurfer installed, and because it is possible to run FreeSurfer with different flags, we set several variables that describe what version of FreeSurfer and what options we are using in the Makefile. Note that the definition for `RECON_ALL` refers to `RECON_FLAGS` seemingly before it is set. Recall that `make` dereferences variables when it uses them, so the order that these variables are set does not matter. This is not like `bash`!

❸ By default, `make` uses `/bin/sh` to interpret recipes. Sometimes this can cause confusion, because `sh` has only a subset of the functionality of `bash`. We can avoid such confusion by setting the `make` variable `SHELL` explicitly.

```
❹ SUBJECTS=$(notdir $(wildcard $(PROJHOME)/subjects/*))
```

❹ We need to obtain a list of subject identifiers to process. Here, we form this list by using a wildcard to obtain all the subject directories in `PROJHOME` and then stripping away all the directory prefixes using the `notdir` call.

```
SESSION=1
❺ inputdirs=$(SUBJECTS:%=%.t$(SESSION))

❻ .PHONY: qa setup freesurfer

❼ setup: $(inputdirs)

❽ %.t$(SESSION):  $(PROJHOME)/subjects/%/visit$(SESSION)/mpr-1.nifti.nii.gz
        mkdir -p $@/mri/orig; \
        cp $^ $@/mri/orig; \
        cd $@/mri/orig; \
        mri_convert mpr-1.nifti.nii.gz 001.mgz
```

❺ This Makefile is intended to handle a longitudinal acquisition. Normally, one indicates the timepoint (here, the `SESSION` variable indicates the timepoint) by appending some suffix to the subject identifier. Here, we append the suffix `.t1` to each subject identifier to indicate that we are processing the first session. To run the makefile on the second timepoint, one could either edit it, or set this variable when calling `make` as follows:

```
$ make SESSION=2
```

❻ We define three targets that do not correspond to files, so these are denoted as phony targets.

❼ The phony target `setup` depends upon the input directories we defined in ❻ .

❽ This recipe creates the input directories by transforming the first MPRAGE image from the subject directory into mgz format. More complicated recipes may include conditionally choosing one of multiple MPRAGE images, using two images if available, and so forth.

❾
```
%.t$(SESSION)/mri/aparc+aseg.mgz: %.t$(SESSION)
        rm -rf `dirname $@`/IsRunning.*
        source $(FREESURFER_SETUP) ;\
        export SUBJECTS_DIR=$(SUBJECTS_DIR) ;\
        $(RECON_ALL) -subjid $*.t$(SESSION) -all
```

❾ FreeSurfer creates many output files when it runs. Here, we select one of the critical files that should exist upon successful completion, `mri/aparc+aseg.mgz` to be the target of this rule. It depends upon the directory having been created so that we can call `recon-all` by specifying the subject directory. You might think that it would be wise to specify multiple FreeSurfer output files as targets. In this case, if the multiple targets are specified using a pattern rule, the recipe would be executed only once to create the targets. However, if we did not use a pattern rule, the recipe could be executed once per target. This is clearly not the intended behavior. To avoid confusion, we usually pick a single late-stage output file to be the target.

❿
```
qa: $(inputdirs:%=QA/%)
```

⓫
```
QA/%: %
        source $(FREESURFER_SETUP) ;\
        $(QA_TOOLS)/recon_checker -s $*
```

❿ We can create a number of quality assurance (QA) images from the FreeSurfer directories using the `recon_checker` program. The `qa` target depends upon directories within the `QA` subdirectory. These are created by `recon_checker` in ⓬ .

⓬
```
Makefile.longitudinal:
        $(PROJHOME)/bin/genctlongitudinalmakefile > $@
```

⓫ After all the cross-sectional runs have been completed, we can run the longitudinal pipeline. The first step in this pipeline is to create an unbiased template from all timepoints for each subject. The second step is to longitudinally process each timepoint with respect to the template.

Here, we have a bit of a problem specifying these commands to `make` because each subject may have a different number of timepoints and subjects may be missing a timepoint that is not the first or last. The syntax of the `recon-all` command to create an unbiased template does not lend itself well to using wildcards to resolve these issues:

```
$ recon-all -base <templateid> -tp <tp1id> -tp <tp2id> ...  -all
```

We solve these problems by writing a shell script that generates a correct Makefile (`Makefile.longitudinal`). This is an example of taking a "brute force" approach rather than trying to use pattern rules or something more sophisticated. It gets the job done.

The new makefile defines a target `longitudinal`, and can be called as follows, adding additional flags for parallelism.

```
$ make -f Makefile.longitudinal longitudinal
```

Running XNAT access

This is an example of how to use a Makefile to create and populate a project directory with images from an open dataset stored in an XNAT (eXtensible Neuroimaging Archive Toolkit) database, in this case from the NITRC (Neuroimaging Informatics Tools and Resources Clearinghouse) 1000 Functional Connectomes project image repository at http://www.nitrc.org/ir.

In order to run this pipeline you will need to have first created an individual user account on NITRC, at http://www.nitrc.org/account/register.php.

To create the SUBJECTS file: From the NITRC Image Repository home page "Search," choose Projects → 1000 Functional Connectomes. At the 1000 Functional Connectomes project home page, select Options → Spreadsheet to download a CSV file of the 1288 subjects AnnArbor_sub00306 ... Taipei_sub91183. Select a subset, or all, of the subjects from the first column of the downloaded spreadsheet, and save to a text file. This will be your `Subjects` input file that the Makefile will iterate through.

In our example, the 23 Baltimore subjects were chosen, and the text file is named `Subjects_Baltimore`.

The directory hierarchy that this Makefile creates under your project home, from which make is run, is: subject `$SUBJECTS` visit1 T1.nii.gz T1_brain.nii.gz rest.nii.gz visit1 `$SUBJECTS`

Note that here, the Makefile functions to enable parallel execution of `mkdir`, `curl`, and `ln` as well as tracking dependencies.

The code for this example is in `$MAKEPIPELINES/fcon_1000/Makefile`. Sections are described as follows.

```
# Site hosting XNAT
❶ NITRC=http://www.nitrc.org/ir3

❷ SHELL=/bin/bash

# Obtain the list of subjects to retrieve from NITRC
❸ SUBJECTS = $(shell cat Subjects_Baltimore)

❹ .PHONY: clean all allT1 allT1_brain allrest allsymlinks
```

This portion of the Makefile defines key variables and targets. ❶ We set the "base name" of the XNAT web site in a variable `NITRC`. This can be changed when using another XNAT repository, and the variable can be named accordingly.

❷ By default, `make` uses `/bin/sh` to interpret recipes. Sometimes this can cause confusion, because `sh` has only a subset of the functionality of `bash`. We set the `make` variable `SHELL` explictly.

❸ The `SUBJECTS` variable will contain a list of the subject data we wish to download. The individual subject names will be used to create directory names.

❹ We define six targets that do not correspond to files, so these are denoted as phony targets.

```
❺ all: sessionid allT1 allT1_brain allrest allsymlinks
❻ allT1: $(SUBJECTS:%=subjects/%/visit1/T1.nii.gz)
❼ allT1_brain: $(SUBJECTS:%=subjects/%/visit1/T1_brain.nii.gz)
❽ allrest: $(SUBJECTS:%=subjects/%/visit1/rest.nii.gz)
❾ allsymlinks: $(SUBJECTS:%=visit1/%)
```

❺ "all" is the default target, and simply defines the five dependencies ❻ This is the formula for the first image file dependency, a T1 for each of the SUBJECTS. The pattern matching names each the subjects/ subdirectories with the individual subject identifier ❼ T1 skullstripped is the second image file dependency ❽ A resting state scan is the third image file dependency ❾ The last thing the Makefile does is create a visit1/ directory after the subjects/ directory has been populated. Pattern matching here names each of the visit1 subdirectories with one of the individual subject identifiers. Each visit1/`SUBJECTS` will be a symbolic link to the actual subjects/`SUBJECTS`/visit1 directory.

```
single session ID, store in "sessionid" file
```

```
sessionid:
        @echo -n "Username:␣" ;\
        read username ;\
        curl --user $$username $(NITRC)/REST/JSESSION > $@
```

10 Here we are using the client URL Request Library (cURL) to create a session with the XNAT server. The first line prompts for the user's name on the XNAT server, the second line reads and stores that in the variable `username`. With one single REST transaction, the cURL call on the following line, we authenticate with the XNAT server, entering a password only once, and saving the return value `SESSIONID` in a file named `sessionid`. This single session will persist for an hour.

```
ad "1000␣Functional␣Connectomes" subject data in NII.GZ format from NITRC
*subjects/%/visit1/T1.nii.gz:  | sessionid
# Create this subject's directory
*       mkdir -p `dirname $@`; \
*       curl --cookie JSESSIONID=`cat sessionid` $(NITRC)/data/projects/fcon_1000/
    subjects/$*/experiments/$*/scans/ALL/resources/NIfTI/files/scan_mprage_anonymized.
    nii.gz > $@
```

11 This and the two file download recipes that follow will have a dependency on the `sessionid` obtained from the preceding recipe. 12 Pattern substitution with `SUBJECTS` will create the subjects/ subdirectories that do not already exist with each of the subject identifiers in turn. The `dirname` shell command will result in this command making visit1/ subdirectories which will hold the target file T1.nii.gz defined in the rule. 13 This cURL command uses the JSESSIONID which was stored in the `sessionid` file. The URL defined here is specific to the location where scan data of interest is stored on the NITRC instance of XNAT. Note the pattern substitution with `SUBJECTS` in two places. The XNAT-stored file scan_mprage_anonymized.nii.gz is downloaded and saved under the local name T1.nii.gz

```
*$subjects/%/visit1/T1_brain.nii.gz:  | sessionid
mkdir -p `dirname $@`; \
curl --cookie JSESSIONID=`cat sessionid` (NITRC)/data/projects/fcon_1000/subjects/$*/
    experiments/$*/scans/ALL/resources/NIfTI/files/scan_mprage_skullstripped.nii.gz >
    $@
```

14 This recipe also creates the visit1/ subdirectory for each of the `SUBJECTS` in the list. The cURL command is identical to the one above except for the name of the scan data file of interest, in this case scan_mprage_skullstripped.nii.gz, which is saved as T1_brain.nii.gz

```
*subjects/%/visit1/rest.nii.gz:  | sessionid
mkdir -p `dirname $@`; \
curl --cookie JSESSIONID=`cat sessionid` $(NITRC)/data/projects/fcon_1000/subjects/$*/
    experiments/$*/scans/ALL/resources/NIfTI/files/scan_rest.nii.gz > $@
```

15 This recipe will create the visit1/ subdirectory if necessary for each of the `SUBJECTS` in the list. The cURL command is identical to the ones above except for the name of the scan data file of interest. Here scan_rest.nii.gz is saved as rest.nii.gz

```
ic links from visit1 to the individual subject visit1 directories
*visit1/%:
ln -s ../subjects/$*/visit1 $@
```

16 This recipe populates the project top-level visit1/ directory with symbolic links, pointers to the actual locations of the subjects' visit1 data downloaded above. This enables an alternate way to access the subject data.

```
over

# Remove all - subject directories, visit1 symbolic links, and JSESSIONID file
```

```
*rm -rf subjects/*; \
rm -rf visit1/*; \
rm -f sessionid
```

17 This clean recipe will delete all subjects/ subdirectories, links in the visit1/ directory, and the prior sessionid.