

This homework looks forbidding, but it's much easier than it looks. Start early anyways, to make sure you know what's going on. You have a few scripts to write, but they all use the same few ideas.

The point of this assignment is to write a set of scripts to implement NH, a Notes Handler system. The idea is simply to implement a system that keeps a database of "notes", and to provide for a way to classify them and access them. Think computerized Post-It notes.

The notes will be kept in a special directory, that you get to choose. Each note will be stored in its own file, with a special format: the first line of the file will be the title of the note, the second line will contain the date and time the note was created, and the remaining lines will hold the actual content of the note. Notes are stored in files whose names are numbers.

To start off, create a directory in which the notes will be kept. Define an environment variable *NH_DATABASE* that points to that directory. Your script will use that environment variable to find the notes. Do not set that variable in your scripts! (For example, on my system, I will test your scripts with my own notes database, and I will set *NH_DATABASE* accordingly.)

For example, I created my notes directory on *babbage* at */home/cs214/HW2/nhdir/*, so I set *NH_DATABASE* to */home/cs214/HW2/nhdir*. You can have a look there to see how your notes should end up looking.

Remember, be creative! You have a lot of tools available: *cat*, *head*, *tail*, *ls*, *grep*, *sort*, and so on.

Part I

In this part, we'll create the basic scripts that add a note to the database (*nnew*), list the database (*nscan*), and display a given note in the database (*nshow*). **These scripts should work no matter where they are run from. The notes are kept in a fixed directory, stored in *NH_DATABASE*. Your scripts should access the notes in that directory. They cannot assume that the notes are kept in the same directory as the scripts!**

nnew

Write a script *nnew* that creates a new note in the database. If you pass an argument to *nnew*, that argument should be taken as the title of the note. If no argument is provided, supply a suitable default title, such as "Untitled". The date should be of the form "mm/dd/yy hh:mm:ss", obtained by using the unix command *date*.¹ The content of the note should be read from *stdin*.

¹Look at the man page to see how to obtain the above format. It's not hard. Experiment on the command line.

The interesting bit here is figuring out how to name the file that the note will be put in. Recall that all note files are created in the *NH_DATABASE* directory, and they are named using numbers. The number you should pick for the new file is one more than the largest number already used in the notes directory. For example, if the directory contains files 1, 3, 5, 8, your new file should be named 9. Figuring out how to do this is probably the hardest bit of the homework. (Hint: you need to look at all the files whose names are numbers, and probably sort them to get at the current highest number; all these operations can be performed by invoking appropriate Unix commands.)

Here's a sample interaction:

```
$ ls nhdir/
1 2 3 4
$ nnew 'This is a new note'      reads from stdin
Some content
Some more content
$ ls nhdir
1 2 3 4 5
$ cat nhdir/5
This is a new note
02/27/02 21:44:17
Some content
Some more content
$
```

nscan

Write a script *nscan* that lists the notes in the database in some nice way. For each note in the database, you want to output to *stdout* the note number (i.e., its filename), as well as the date and the title. There should be one line per note. The lines should be output in *increasing* order of note number. Watch out to make sure that you output 9 before 10 (I've messed that up myself in my first pass at this). Make sure that you do not output a line for files in the notes directory whose name is not a number! (we'll be adding files in that directory, we don't want those to be displayed as notes.)

Here's a sample output:

```
$ ls nhdir/
1 2 3 4 5
$ nscan
1 02/27/02  ORIE seminar
2 02/27/02  AI/NLP seminar
3 02/27/02  cs214 hw2
4 02/27/02  Some url
```

```
5 02/27/02 This is a new note
$
```

Notice that I've only displayed the "mm/dd/yy" part of the date in my output. Try to get that as well. If not, no big deal. What you've learned implementing *nnew* will certainly apply here, when it comes to getting those filenames that are numbers.

nshow

Allright. So now we have a way to add a note to the database, and to get a list of the notes in the database. Now, we would like to display a note simply by specifying its number. Write a script *nshow* that takes a single argument which must be a number, and displays the content of the corresponding note. (The user should not have to specify a path to the appropriate note—the path information is contained in *NH_DATABASE*—but only the note number; see the sample interaction below.) Make sure that the argument is indeed a number, and that there is a note in the notes directory with that number! (Report an error if not.) For the sake of readability, don't simply dump the file to *stdout*. Highlight the title by prefixing it with *Title:* , and prefix the date with *Date:* . Add a separation of some sort between the date and the content of the note. The following interaction may give you a clue what I mean:

```
$ cat nhdir/5
This is a new note
02/27/02 21:44:17
Some content
Some more content
$ nshow 5
Title: This is a new note
Date: 02/27/02 21:44:17
-----
Some content
Some more content
$ nshow 6
nshow: note 6 does not exit
$
```

Part II

In Part I, we have developed the basic infrastructure of the notes database: adding notes, listing notes, displaying notes. We can easily imagine scripts to remove notes, pack notes (for instance, if you have notes 1,3,5,7,10 in the database, you may want to pack them as 1,2,3,4,5), and so on.

Here, we will focus on a different extension. We will add a capability for *categorization*. Essentially, we will define a way to classify notes into categories, based on some characteristics of the notes themselves (for example, words appearing in the note, or something like that). We will then write a little script like *nscan*, except that it display all the notes in a certain category.

We will take a very flexible approach to categories. A category is just a string. For the sake of argument, the category *url* will be the category of notes containing a URL. Similarly, the category *talk* will be the category of notes announcing talks, etc. Note that a note can belong to more than one category (i.e., if a talk announcement includes a URL!). The way we will implement the classification is as follows. Write a little script called *categorize* in the notes directory, that takes a filename as argument (the filename of a note) and returns all the categories that note belongs to, one category per line. (This script takes a full path to the note file, differently than, say *nshow*. The reason why is that the users of NH will never invoke *categorize* themselves. So we don't need to be nice.) The choice of categories you want to implement is up to you, and reflects what you care about. When you dream up new categories, or new ways of classifying your notes, you can simply change the *categorize* script to reflect this. For the sake of testing, we'll recognize at least the two categories I've talked about earlier. Your *categorize* script should recognize that a note is in category *url* if it contains a URL, that is, if the note has a *http:* or *HTTP:* anywhere in it. Similarly, a note will belong to the *talk* category if it contains the string **TALK** anywhere in it. Remember, a note can belong to more than one category.

Here's how my implementation of *categorize* works on the sample notes on *babbage*:

```
$ nhdir/categorize nhdir/1
url
talk
$ nhdir/categorize nhdir/4
url
$ nhdir/categorize nhdir/3
$
```

Hence, note 1 is categorized as being in category *url* and *talk*, while note 4 is in category *url*. Note 3, in my case, is not recognized as being in any category.

Remember, the *categorize* script goes in the notes directory. It will be invoked by the next script.

nscanc

Write a script *nscanc* that lists the notes in the database in the same way as *nscan* does, but only those notes that belong to all the categories specified as arguments to *nscanc*. For example, *nscanc url* should list those notes that are in the *url* category, while *nscanc url talk* should list those notes that are in both the *url* and *talk* categories. The output should be in the same format as *nscanc* (you can reuse most of the code). Your script should call *categorize* in the notes directory to classify the notes.

Here's a sample output, given my implementation of *categorize* tested above:

```
$ nscanc url
1 02/27/02 ORIE seminar
2 02/27/02 AI/NLP seminar
4 02/27/02 Some url
$ nscanc talk
1 02/27/02 ORIE seminar
2 02/27/02 AI/NLP seminar
$ nscanc talk url
1 02/27/02 ORIE seminar
2 02/27/02 AI/NLP seminar
$ nscanc talk url foo
$ nscanc
nscanc: need at least one category
$
```

Notice that there are no notes in category *foo*, so the corresponding *nscanc* does not return anything. Moreover, an error is reported if *nscanc* is not supplied with any category. (One can imagine simply adding categories functionality to *nscan*. That's not been done for pedagogical reasons.)

That's it. Have fun. Submission instructions will be posted to the web site.