Advanced Unix Tools	**Homework 2**
CS214 Spring 2004	**Due Date: Friday, March 12, 2004**

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 large script to write, but it uses relatively few ideas.

The point of this assignment is to write a script called `todo` to handle a to-do list. The idea is simply to implement a system that keeps a database of "things to do" (I'll call them entries from now on), and to provide for a way to classify them and access them. Think computerized Post-It notes.

The script `todo` should take a number of options, to add an entry to the todo list, or to list all the entries, or the entries that have to do with today, or to search the entries for keywords, etc. For example, invoking `todo -new` will ask you for a new entry to add to the database. Invoking `todo -list` should list all the entries, while invoking `todo -today` should list all the entries that pertain to today, and so on.

The entries will be kept in a special directory, that you get to choose. Each entry will be stored in its own file, with a special format: the first line of the file will be the name of the entry, the second line the month it is due, the third line the day it is due, the fourth line the year it is due, and the remaining lines hold the text associated with the entry. (For instance, giving some details as to what there actual is to do, etc.). Entries are stored in files whose names are numbers.

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

For example, I created my entries directory on `babbage` at `~cs214/HW2/tododb/`, so I should set `TODO_DATABASE` to `~cs214/HW2/tododb`. You can have a look there to see how your entries should end up looking.

Remember, be creative! You have a lot of tools available: `cat`, `head`, `tail`, `ls`, `grep`, `sort`, and so on. Isolating functionality in various functions in your script is also good software engineering (assuming one can use such grandiose terms for shell scripting...)

Part I: New Entries

Let's create the script and give it the basic functionality of adding a new entry to the database.

Write a script `todo` that if supplied an argument of the form `-new` creates a new entry in the database. The script should ask the user for a title for the entry, for a month, day, and year by which the entry should be completed, and for the actual text of the entry. This entry should then be put in its own file in the directory pointed to by the `TODO_DATABASE` environment variable.

To ask the user for input, you can use the `read` command. Here's the info from Lecture 4 (in case you haven't read them...):

> The command `read` *`var`* will read a line of input from the user, and put the line in variable *`var`*. If more than one variable is supplied, i.e., `read` *`var1 var2 var3`*, then the first word of the line input from the user is put in *`var1`*, the second in *`var2`*, and the rest of the line is put in *`var3`*. A prompt may be supplied by using a `-p` option, as in `read -p 'some test' var` (notice the quotes to give a prompt which may contain spaces...) The following lines will repeatedly query the user for a `yes` or `no` until he gets it right:

```
read -p "yes or no? " answer
while [ "$answer" != "yes" -a "$answer" != "no" ]
do
  echo "Please enter yes or no"
  read -p "yes or no? " answer
done
echo "the answer was $answer"
```

The interesting bit here is figuring out how to name the file that the entry will be put in. Recall that all entries are created in the `$TODO_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 entries 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 (you don't have to do things exactly this way... as long as you figure out a way to input the information and put it in a file in the right place, I'll be happy). User input is written in italics.

```
$ ls tododb/
1  2  3
$ todo -new
Title: get plane ticket for ETAPS
Month: 3
Day: 5
Year: 2004
Enter text of the entry (Press CTRL-D when done):
This needs to get done ASAP
Also, should look for hotel
$ ls tododb
1  2  3  4
```

```
$ cat tododb/4
get plane ticket for ETAPS
3
5
2004
This needs to get done ASAP
Also, should look for hotel
$
```

Part II: Listing Entries

Update your script so that if supplied with argument `-list`, it lists the entries in the database in some nice way. For each entry in the database, you want to output to `stdout` the entry number (i.e., its filename), as well as the date due and the title. There should be one line per entry. The lines should be output in *increasing* order of entry 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 entries directory whose name is not a number!

Here's a sample output:

```
$ ls tododb/
1  2  3  4
$ todo -list
 1  02/27/2002  ORIE seminar
 2  02/27/2002  AI/NLP seminar
 3  03/02/2004  finish cs214 hw2
 4  03/05/2004  get plane ticket for ETAPS
$
```

Again, you don't have to display things exactly as here, but put in all the information in some aesthetically pleasing way. Don't display the text of the entry, that's what's next. What you've learned implementing `-new` will certainly apply here, when it comes to getting those filenames that are numbers.

Part III: Showing Entries

Allright. So now we have a way to add an entry to the database, and to get a list of the entries in the database. Now, we would like to display the full text of an entry simply by specifying its number. Update your script so that if it supplied with the two arguments `-show` and a number, it displays the content of the corresponding entry. (The user should not have to specify a path to the appropriate entry—the path information is contained in `$TODO_DATABASE`—but only the entry number; see the sample interaction below.) Make sure that the number argument is indeed a number, and that there is a entry in the entries 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 entry. The following interaction may give you a clue what I mean:

```
$ cat tododb/4
get plane ticket for ETAPS
3
5
2004
This needs to get done ASAP
Also, should look for hotel
$ todo -show 4
Title: get plane ticket for ETAPS
Date: 3/5/2004
--------
This needs to get done ASAP
Also, should look for hotel
$ todo -show 5
todo: entry 5 does not exit
$
```

Part IV: Finding Entries by Date

Let's do a variation of listing all the entries in the database, by allowing you to list entries that are due on a given date. Update your script so that when it is supplied with four arguments, the first `-date`, the second, third and fourth numbers (interpreted as the month, day, and year), it lists all the entries due on that day. You'll be doing essentially what you've done in Part II, except that now you only display the entries whose date match.

Here's a sample output:

```
$ ls tododb/
1  2  3  4
$ todo -date 3 2 2004
 3  03/02/2004  finish cs214 hw2
$
```

You can use the same output format as that of Part II (with some thought, you'll actually be able to reuse code you've written for Part II...).

Once you've implemented the above, add a new option that displays those entries for today. Update your script so that when it is supplied with an argument `-today`, it lists all entries due today. That means you have to get your hands on today's date. The `date` command does that under Unix. In fact, the `date` command lets you query what the current month is, what the current day is, what the current year is, all information you need to match entries that are due on the current day. See the `man` pages for `date`. For example, `date +%m` sends the current month number to `stdout`.

Once you've got the current month, day, and year, you should be able to simply reuse the code that lists all the entries due on a given date to list all the entries due today. Here's a sample output:

```
$ ls tododb/
1  2  3  4
$ todo -today
 4  03/05/2004  get plane ticket for ETAPS
$
```

To think about: If you have time to spare, try to write an option that list all the entries that are due between now and a given date. That requires you to figure out date arithmetic. There are ways of doing that, none too clean. The simplest is probably to use the `date +%s` functionality.

Part V: Searching Entries

Last bit. Another variation on listing the entries in the database. Update your script so that if you supply it with two arguments `-search` and a string, it lists all the entries whose text contain an occurrence of the supplied string. This should be a simply variation of what you did in Part IV. In fact, if you've done things nicely, you should simply be able to reuse the code you wrote for Part IV.

Here's a sample output:

```
$ ls tododb/
1  2  3  4
$ todo -search hotel
 4  03/05/2004  get plane ticket for ETAPS
$ todo -show 4
Title: get plane ticket for ETAPS
Date: 3/5/2004
--------
This needs to get done ASAP
Also, should look for hotel
$
```

To think about: How would you change your code so that it would let you search for an arbitrary regular expression in the entries?