Week07 - cron, tar, SQL, and exam

March 16, 2020

In this lecture you'll learn cron, how to backup your linux machine, a bit about SQL (not essential, but should help you get more out of next week's lecture) and then we'll discuss the exam a bit.

1 Introduction

Plan for the evening:

- **7:00 - 7:20** *get setup*
- **7:20 - 7:30** *morale boosting review*
- **7:30 - 7:55** *backups with cron + tar*
- **7:55 - 8:00** *Break*
- **8:00 - 9:00** *sqlite3 murder mystery*
- **9:00 - 9:05** *Break*
- **9:05 - 9:45** *prepare for exam*

2 Setup

Create a NEW $5 Debian 10 server and add your ssh key so you can get in.

MAKE SURE ITS A FRESH NEW SERVER! You might have files on your old machine that will cause errors in today's lecture. Just delete your old machine and create a new one to avoid problems.

Then:

```
root@machine$ apt update
root@machine$ apt install sqlite3 # well use this later.
root@machine$ git clone
   https://github.com/melvyniandrag/LinuxClassRepo.git
#
```

We'll be looking at the materials in the following directories

- Lectures/Week07_SQLite_cron_Backups
- Homework/Week07

3 Review

At this point, we're 7 weeks (nearly 2 months!) into your Linux career. You may be realizing that learning Linux is something of a lifestyle - this will take lots of practice! Soon we'll start doing some real projects on our Linux machines like configuring various types of servers, but let's make sure that we know basic things like create files, move and modify files, and navigate our linux machine with ease. I expect you to understand these concepts right now:

`should use this right margin for lecturer notes - what to draw on the board, what leading questions to ask students, relevant anecdotes, etc.`

1. create a hidden file
2. create a directory
3. delete a file
4. delete a directory
5. add some text to a file with vim.
6. what are some text editors besides vim?
7. why are we using vim in this class and not those (answer : I like vim and it's my class. If you like nano or pico or emacs, then use that on your own time! In this class we're learning vim.
8. ls
9. ls -a
10. ls -l
11. chmod
12. How to execute a file with "./"
13. cp
14. cp -r
15. mkdir -p
16. mv
17. rm
18. rm -r
19. what's the sudo group?
20. who is root?
21. how to install software on debian with apt?
22. what is the keyboard shortcut to send SIGINT (terminate a process, not pause it)?
23. keyboard shrtcut to send SIGTSTP (stop aka pause a process)?
24. signal to restart a process?
25. grep
26. basic notion of a regular expression. Could you find this pattern in a file: letter,number,letter,number,letter,number? e.g.could you use grep to find "a1b2c3" and "z8t5r2"?
27. difference between a job and a process?
28. what is your favorite version control software?
29. what is a pipe?
30. to ssh into a server you need to put your ssh key in which file?
31. purpose of setuid programs? (Allow a non root user to run a program as if he/she were root).

There's more to know, I just spent a minute thinking about the semester - the definitive source of what you should know is in the lecture materials from the previous six weeks.

4 cron

cron is used for scheduling jobs on your computer. You can modify a thing called a **crontable** to tell your computer to do thing at certain time(s) in the future.

All you have to do is add some command(s) to the crontable and then yuou're good to go! The computer will do the rest of the work! Pretty cool stuff. Your phone does stuff like this already - What'sApp usually backs up your messages and images to the cloud at night time, I don't know what Google Photos and iCloud do, but they probably do something similar or offer you a mechanism to do it.

We are using vim in this class, so we'll want to make sure that the crontable is opened with vim. Make sure to perform the following step to set your default editor to vim.

```
# note I haven't tested this command yet,
# make sure I've written it right please, students!
# I don't have internet now and i'm just going by memory
root@digitalOcean$ apt install vim
root@digitalOcean$ update-alternatives --config editor
# then choose vim or vi
# if you have multiple choices for vim, choose vim.basic
```

Now we are ready.

View your cron table with *crontab -l*, edit the crontab with *crontab -e*. Choose vim if prompted to select which text editor to use. It may say vim.tiny, vim.basic. To be honest in all these years I've just chosen a random one. I should learn the distinction.

To schedule jobs, you add rows of the form

m h dom moy dow command

where the above represent:

- m - minute (0-59),
- h - hour (0-23),
- dom - day of the month (1-31),
- moy - month of the year (1-12),
- dow - day of the week (0-6, 0=Sunday)
- command - bash command

For example, to echo "hello" into a file on March 4th, every minute of the 19th hour of the day, you will add the following line to your crontab, as the root user.

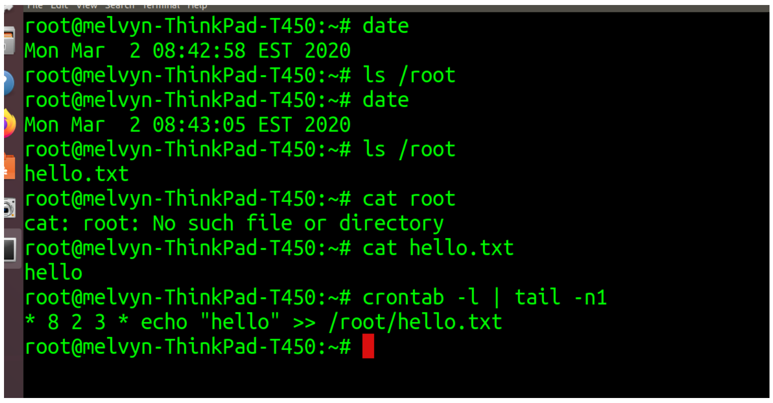
```
* 19 4 3 * echo "hello" >> /root/hello.txt
```

Note that the above command says * for minutes (aka any minute) 19 for hours (if the hour is 19), 4 for day (if the date of the month is 4), 3 for month (March) and * for dow (any day of the week, could be monday, tuesaday, etc.).

Maybe belabor this point a bit. The m h dom etc. stuff can be confusing at first. Take a few minutes to discuss it with students to try and clarify it a bit.

You will note that these conditions are all met right now, today! If you add this to your crontab and wait a few minutes, then look at /root/hello.txt, you'll see it says "hello" a few times inside!

So now you know how to schedule jobs on your computer. See figure 1 to see me running the above cronjob on my computer.



```
root@melvyn-ThinkPad-T450:~# date
Mon Mar  2 08:42:58 EST 2020
root@melvyn-ThinkPad-T450:~# ls /root
root@melvyn-ThinkPad-T450:~# date
Mon Mar  2 08:43:05 EST 2020
root@melvyn-ThinkPad-T450:~# ls /root
hello.txt
root@melvyn-ThinkPad-T450:~# cat root
cat: root: No such file or directory
root@melvyn-ThinkPad-T450:~# cat hello.txt
hello
root@melvyn-ThinkPad-T450:~# crontab -l | tail -n1
* 8 2 3 * echo "hello" >> /root/hello.txt
root@melvyn-ThinkPad-T450:~# 
```

Figure 1: `Check the timestamps! There was not file called /root/hello.txt, but then magically a minute later it was there. In the image I show you my crontable entry.`

5 Backups with Cron

This was a silly example that I just gave you. More commonly you'll use cron to back up your computer.

5.1 tar

tar is used for creating archives, i.e. a single file containing a bunch of files. It's similar to a zip or rar file you may have seen at some time in your life. Heck, you may be using tar files everyday too, I don't know. Some folks were submitting me homeworks in the beginning of class as an archive - I can't remember if people were submitting zips, tars or what. In any event, let's create a little sample directory and we'll use tar to pack it into an archive.

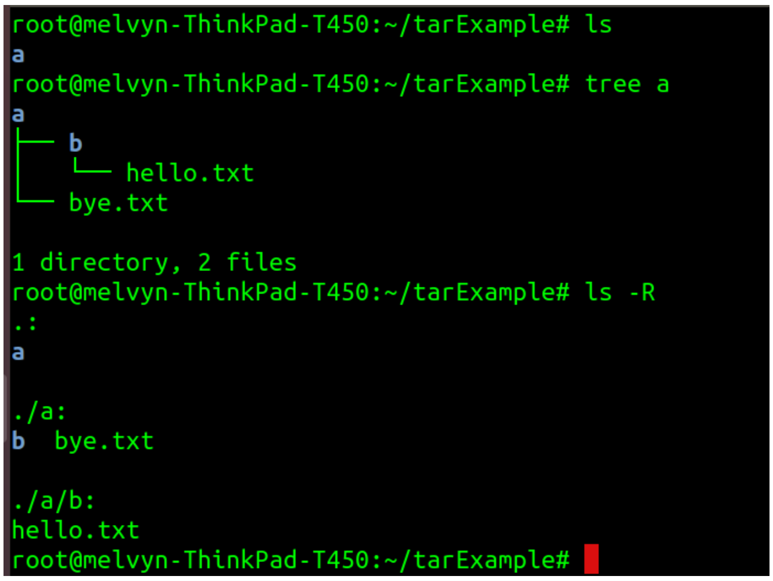
For the example I created the files / directories shown in figure 2

Now what I want to do now is create a tar archive from these files. Here's how you do it:

```
root@machine$ ls
a
root@machine$ tar -cvf a.tar a # create a.tar from directory a.
root@machine$ ls
a a.tar
```

To verify that it worked, let's unpack the archive in another directory.

```
root@machine$ ls
a a.tar
root@machine$ mkdir NewDirectory
```



```
root@melvyn-ThinkPad-T450:~/tarExample# ls
a
root@melvyn-ThinkPad-T450:~/tarExample# tree a
a
├── b
│   └── hello.txt
└── bye.txt

1 directory, 2 files
root@melvyn-ThinkPad-T450:~/tarExample# ls -R
.:
a

./a:
b  bye.txt

./a/b:
hello.txt
root@melvyn-ThinkPad-T450:~/tarExample# 
```

Figure 2: Files for this example. There is nothing special about these files and directories, we're just using them for the example. Also note I'm showing two ways to view files in directories - *tree* and *ls -R*

```
root@machine$ mv a.zip NewDirectory
root@machine$ cd NewDirectory
root@machine$ ls
a.tar
root@machine$ tar -xvf a.tar
root@machine$ ls
a.tar a
```

In the above examples you've seen how to create a .tar archive and how to unpack it. For the record, let's just see where the tar executable is on our system:

```
root@machine$ which tar
/usr/bin/tar
```

This output may vary. It might say '/bin/tar' or '/usr/bin/tar' or something else. This doesn't matter. Just make a mental note of what it says.

5.2 backup

I already showed you a cron job using echo. Now we'll just do a cronjob with tar. A subtle and important difference between tar and echo is that echo is a builtin command, and tar is not. If you are dying to know what that means, or finding my class too easy, then ask me after class and I'll explain the differnce. What matters to you and your grade, is that you are able to create a cron job with tar. And here is how you do it:

In this example, we will back up /home/yourname to a tar file. There are many ways to do this, but I'm going to show you this way to give your fingers some practice. To learn more about this, go on youtube, or talk to me when we both have free time.

Create a user for the coming example:

```
root@machine$ adduser YOURNAMEHERE
# then follow the instructions as last time
```

Add the following line to root's crontab to back up /home/melvyn every day at midnight.

NOTE: BELOW I PUT /USR/BIN/TAR - you need to put the output of 'which tar'. I don't know where tar will be on your system.

```
0 0 * * * /usr/bin/tar -cvf /root/melvyn-home.tar /home/melvyn
```

`Change the cron schedule to one that will be useful in class so students can see it in action. The time is probably around 7:40 pm. So set the time stamp to 42 19 * * * so that the tar command will run at 7:42 PM.`

AND OF COURSE YOU MUST HAVE CREATED A USER 'melvyn' for this to work otherwise there won't be any /home/melvyn to backup.

If you want to backup /home/jacqueline/importantfiles every tuesday at noon you would add the following to your crontab:

```
0 12 * * 2 /bin/tar -cvf /root/jackie.tar
   /home/jacqueline/importantfiles
```

The above line in the cron table says to run the tar command when:

- minute = 0
- hour = 12
- day of the month = any
- month of the year = any
- day of the week = 2 (Tuesday, because Sunday is 0, Monday is 1, etc.)

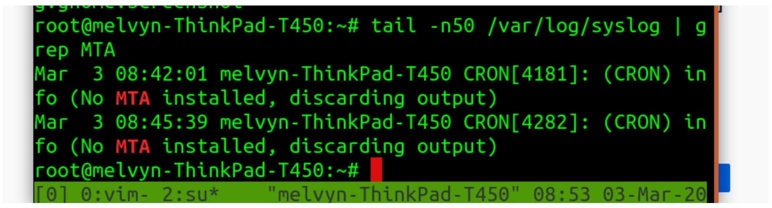
and note, once again, that we must provide the path to *tar* as *cron* doesn't know where *tar* is. *cron* did know where *echo* was in an earlier example, and I told you that this is because *echo* is a **builtin**. This is an important concept to know but there probably won't be any time to discuss more about builtins. If you want a list of all the commands you can use with cron without providing the path (as we do with tar), just type 'help' at the command prompt.

```
user@machine$ help
# a list of all shell builtins is provided.
# you will see echo in the list, but not tar.
```

5.3 Going Further

That's all you need to know to get started! If you want to go further you might want to know what happens if a cronjob fails. Whenever cron has output, be it error output or standard output, it wants to email you. You'll see a message in /var/log/syslog that looks like what I'm showing in figure 3

Also, you might want to know that .tar does not compress your files - it only creates an archive. It puts them all into a big box. But



```
root@melvyn-ThinkPad-T450:~# tail -n50 /var/log/syslog | g
rep MTA
Mar  3 08:42:01 melvyn-ThinkPad-T450 CRON[4181]: (CRON) in
fo (No MTA installed, discarding output)
Mar  3 08:45:39 melvyn-ThinkPad-T450 CRON[4282]: (CRON) in
fo (No MTA installed, discarding output)
root@melvyn-ThinkPad-T450:~#
[0] 0:vim- 2:su*    "melvyn-ThinkPad-T450" 08:53 03-Mar-20
```

Figure 3: `cron wants to send you emails telling you about your cron jobs. If you want to configure your machine to email you the status of your cron jobs, google 'MTA', 'cron' and 'postfix' and follow a tutorial.`

they aren't squished. So, if you have 1GB of files and you create a .tar from them - you'll have a .tar that's bigger than 1GB, because it has to contain all the uncompressed stuff plus some data specific to tar files. If you want to compress your files, try adding the -z option to your compression command, and name the file .tar.gz. Like this:

```
user@machine$ tar -xvzf compressedArchive.tar.gz
    directoryToCompress
```

Compression is an interesting topic, very important and lots to say about it, but we have a schedule to keep! If you're enjoying this class and don't feel like you're drowning, then definitely go google about .gz files or come talk to me about it or whatever.

Moving on, now we'll plant some seeds of knowledge that we'll see blossom next week. Let's talk about SQL.

6 Intermezzo

Potty break and discussion before we learn SQL.

7 SQL

7.1 Intro

Structured Query Language. As you've seen, you'll need to know a bit about a million different things to work on Linux. One such thing is SQL

SQL is a language used to poke through tables of data. It's the programming language you use to talk to databases.

It's a programming language, but not in the sense of bash or Java or Python or anything like that. It's more like a fancy way to go through tables of data. A database is like an excel file, but you don't look at it like a spread sheet. You look at it by making **Queries**.

`Do you know the language you use to make queries? (SQL)`

Sqlite3 is a small database software that is widely used in industry. Go online and look around - you'll see everyone either uses sqlite3, or at least has played with it a few times. Tonight is your night to poke at it and learn something.

there are other database programs out there - can you name some?.

`open question to the class. have students name some database programs they know. Access, MySQL, PostgreSQL,`

7.2 Murder Mystery

A while back I came across a cool game that is played with a little sqlite3 database. Let's have a look at it. It's a database that contains all the data you need, as an investigator ,to solve a murder mystery! Let's do it.

The Murder Mystery database is found in the class repository we cloned at the beginning of lecture. It is the sql-murder-mystery.db file in the Week07 lecture directory.

The only clue we have to start with is that a murder occured on Jan 15th, 2018 in SQL City and that we must start by looking at the crime scene reports from the police department's database.

Open the database like this:

```
user@machine$ sqlite3 sql-murder-mystery.db
```

Now we have opened the file. To list the tables in the database we do this:

Do you know what .db stands for in the file extension?

```
sqlite> .tables
# list of tables.
```

This will list all the tables. Of course the table is crime_scene_reports. To see what's in the table we do this:

Have students guess which table we need to look at to see the crime scene reports

```
sqlite> select * from crime_scene_report;
# many many results
```

you'll see the computer puke out an overwhelming amount of data. Now what I'll show you now is important, you will not come up with this idea on your own. These tables have columns. And every column has a header. . But we can't see the heade rinformation because it's pushed all the way to the top. To get the info we can do the following:

Consider drawing a table on the board with column headers

```
sqlite> .header on
sqlite> select * from crime_scene_report limit 1;
date|type|description|city
20180115|robbery|A Man Dressed as Spider-Man Is on a Robbery
    Spree|NYC
```

so we see that the table 'crime_scene_report' has columns 'date', 'type', 'description', and 'city'.

Now, you might want to know what all is in this database. So I'll give you a picture. See figure 4 to see the tables and the columns they contain. There are arrows and keys in the image. If you know what those mean, great. If you don't, it doesn't matter. Ignore them for now and next lecture when we set up a proper database server I'll explain it to you.

SQL students learned so far is SELECT * FROM tableName LIMIT #

We want the reports that happened on January 15th 2018 and looking at the date format can you guess how the date Jan 15th 2018 is written in the table?

wait for students to guess 20180115

So now we want to filter out the table results. We will enter a command like this:

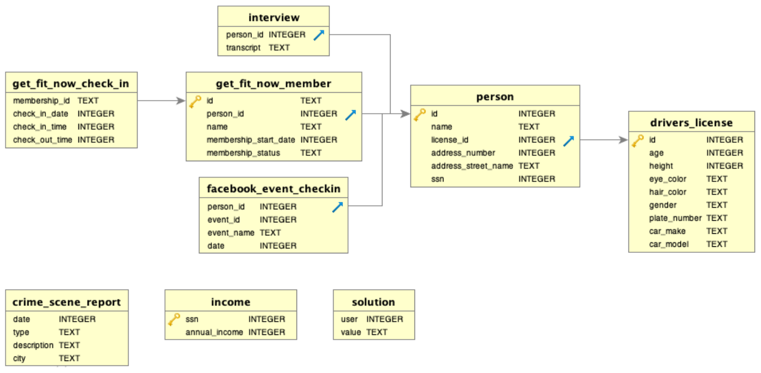


Figure 4: `Murder mystery database schema. schema is a fancy nerd word that means "the layout of the database" i.e. the columns and tables in it and the relationships between them.`

```
sqlite> select * from crime_scene_report where
date="20180115" and city="SQL City" and type="murder";

20180115|murder|Security footage shows that there were 2
    witnesses. The first
witness lives at the last house on "Northwestern Dr". The
    second witness, named
Annabel, lives somewhere on "Franklin Ave".|SQL City
```

We found the report!

On our paper we can write down what we know now - one witness lives in the last house on Northwestern Drive, and the other is named Annabel and she lives on Franklin Ave. Let's see which tables we have again.

```
SQL students
learned here is
SELECT * FROM
tableName WHERE
columnName="value"
AND
otherColumnName="value2"
... etc.
```

```
sqlite> .tables
#tables. Notice there is one called 'person' - this ought to
    have data about
people!
```

We want the rows from the table that match the data we have. What are the columns in the person table?

```
sqlite> select * from person limit 1;
id|name|license_id|address_number|address_street_name|ssn
10000|Christoper Peteuil|993845|624|Bankhall Ave|747714076
```

Let's first follow clue # 1 - the winess lives at the last house on Franklin Ave and is named Annabel:

```
sqlite> select * from person where
    address_street_name="Franklin Ave" and name
like "Annabel%";
id|name|license_id|address_number|address_street_name|ssn
16371|Annabel Miller|490173|103|Franklin Ave|318771143
```

You've learned another SQL command, the LIKE command with the "%" wildcard. In grep you match anything with "." . On the command line you match anything with "*". In sqlite3, you match anything with "%". You also learned the AND command. As you can imagine, there is also an OR command. Programming is a pain in the neck to learn. Thank goodness it is so interesting.

Shall we do another thing in SQL? Let's look at the other clue.

```
sqlite> select * from person where
    address_street_name="Northwestern Dr"
# a flood of facts
# a deluge of data
# an outpouring of observations
# we want the last house on the street! This is too much work
    to look through!
```

Remember, the new commands you learned are SELECT * FROM tableName WHERE column LIKE "something AND otherColumn="someValue"

If we were not clever people we would scroll through all this data and find the biggest house number. But we can do better than that because we are clever! What if there were one million rows here?!?! We would be searching for days. I'll show you one last SQL command.

```
sqlite> select * from person where
    address_street_name="Northwestern Dr" order
by address_number;
# lots of data as before, but now sorted by address_number!
# the biggest row corresponds to ole Morty Shapiro.
```

If we want to do even better we can order by descending and then just grab the first entry.

```
sqlite> select * from person where
    address_street_name="Northwestern Dr" order
by address_number desc limit 1;
id|name|license_id|address_number|address_street_name|ssn
14887|Morty Schapiro|118009|4919|Northwestern Dr|111564949
```

So now we've found the two witnesses. Now we need to find more information about the murder. Well, if you look at the schema image I've provided or you review the tables in the database by typing '.tables' as shown above, you will see there is an 'interview' table. Maybe the next thing to do is look there and see what the witnesses said about the crime. Solving this mystery will be your homework, so I'll leave it at that.

7.3 Play more with SQLite3

Let's run a few more queries before moving on, just so you get more practice.

Is there anyone named John in the person table?

```
sqlite> select * from person where name like "%John%";
#look at results
```

Is there anyone driving a Ford?

```
sqlite> select * from drivers_license where car_make="Ford";
#look at results
# if that doesn't work type
# select * from drivers_license where car_make like "%ord%";
```

What is the largest date in the database?

```
sqlite> select * from crime_scene_report order by date desc
    limit 1;
#look at result
```

7.4 Recap

Let's all agree on what we learned. We've been just playing around with sql for a few minutes. What are the sqlite3 commands you've learned?

1. select * from table;
2. select * from table where column="value";
3. select * from table where column="value" and column2="value2";
4. select * from table where column like "%value%";
5. select * from table limit #;
6. select * from table order by column;
7. select * from table order by column asc;
8. select * from table order by column desc;

Now I should tell you a cool thing about SQL. The commands are not case sensitive! So you can write "select" or "SELECT", "from" or "FROM", "order by" or "OrDeR bY". In fact it's good practice to write the SQL commands in CAPS to differentiate them from data. Sorry to have led you astray in the last half hour or so. So the above commands can be run as

1. SELECT * FROM table;
2. SELECT * FROM table WHERE column="value";
3. SELECT * FROM table WHERE column="value" AND column2="value2";
4. SELECT * FROM table WHERE column LIKE "%value%";
5. SELECT * FROM table LIMIT 5; # or any number
6. SELECT * FROM table ORDER BY column;
7. SELECT * FROM table ORDER BY column ASC;
8. SELECT * FROM table ORDER BY column DESC;

In case I didn't mention it before - ASC means ascending, or the numbers get bigger as you go on. DESC means descending, or the numbers get smaller as you go on.

7.5 End SQL Statements with ;

In the many SQL statements we've run so far you'll have noticed that they all end with a semicolon. This is obligatory.

Don't do this:

```
sqlite> select * from person
```

sqlite will sit and wait for you to provide a semicolon.

7.6 Running sqlite3 commands from a script

You don't have to use the command line to type in commands slowly. You can pass scripts to sqlite3. Here's how you do it:

```
user@machine$ ls
sql-murder-mystery.db script1.sql script2.sql
user@machine$ cat script1.sql
SELECT * FROM person LIMIT 1;
user@machine$ sqlite3 sql-murder-mystery < script1.sql
# script output
user@machine$ cat script2.sql
SELECT * FROM crime_scene_report LIMIT 2;
user@machine$sqlite3 sql-murder-mystery < script2.sql
# output from other script
```

As an aside: We already learned what the < operator means in bash - that's stdin! So you can feed scripts to sqlite3 via stdin. In case you forgot and want to refresh your memory, stdin corresponds to 0, stdout is 1 and stderr is 2.

7.7 Conclusion

We haven't solved the mystery yet! In this lecture we've used the command line interface to sqlite3. I'm showing you sqlite3 because it's wildly popular, it's used in everything from national defense software to video games to websites to embedded applications. And it's super easy to use! You can just fire up the command line and go!

I found this database on the internet a while back. This game was written by students at the (University of Nebraska? Can't rmember). If you want to show this to your friends you can find the original implementation online here: `https://mystery.knightlab.com/`. The database and all the website code is online -can you guess where??? GITHUB! Here's the link, this is where I stole the database for tonight's lecture. `https://github.com/NUKnightLab/sql-mysteries`

Also, if you have Android you can download a crappy implementation of the game I wrote while experimenting with Android: `https://play.google.com/store/apps/details?id=com.ballofknives.sqlmurdermystery`

Your homework will be to solve the murder mystery and send me all the .sql scripts you wrote to figure it out.

8 Break

Another 5 minute break before we move on to discussing the exam.

9 Exam

Let's wrap things up with a discussion of the exam. .

Your midterm will be half programming and half multiple choice exam. You all will write the exam! I want you all to write 3 good multiple choice Linux questions as I've sketched out in the exam directory. Questions about the various commands you know. The thing is, I need you all to make sure that your questions are unique. We learned alot in this class.

Show the class the sketch of the exam I've been working on in the Midterm/SketchOfMidterm directory.

I don't want any duplicate questions, but orchestrating a software project between 30 people is very difficult. So just follow the test instructions and I'll look through your PRs and whatever duplicate questions I see I'll just fix them up myself.

To make this easier, assign topics to people in class. For example, assign Lesly chmod, Jackie ls, Raheem vim, etc.

I've done a few things in this script. Let's make sure to discuss them as we look through the exam.

9.1 bash functions

bash is a programming language and has functions just like all the other languages you know.

In bash a function looks like this:

```bash
run_exam(){
  echo "the function body goes here"
}
```

this is very similar to a java function that looks like this:

```java
public void run_exam(){
  System.out.println("the function body goes here");
}
```

a surprising thing is that you call the function without parentheses. for example consider the following:

```bash
#!/bin/bash

run_exam(){
  echo "function body here"
}

# call the function.
# Note I say run_exam and not run_exam()
run_exam
```

this is different from languages like Java - in those kind of languages you need parentheses.

9.2 read

If you want to read user input to a bash program you can use *read*. Look at MelvynDrag/examQuestions.sh to see my use of read (this is how you will use read).

9.3 source

If you want to import one script into another in bash you type *source scriptName.sh*. In Java you would say *import java.util.Scanner* or something like this. Well, in bash you say 'source'.

9.4 case

bash has a *case* statement. This is like Java's *switch/case*. Look at MelvynDrag/examQuestions.sh to see how it's used. Pretty cool how you end a *case* with *esac*. This is like how bash ends if statements - *fi*.

9.5 final note to lecturer

I guess those are the few new things I've had to use in the exam. In class look over the exam with the class, make sure the instructions are clear and take questions. If time, wait and make sure every student can write at least one exam question.

10 Did anyone get the PR?

Did anyone make a PR on the wicked cool shell scripts repo? It's an easy target. If anyone wants it, raffle it off in class. It would be annoying if the repo maintainer got a flood of PRs about the same issue out of the blue.