<SLIDES!!!>

REVIEW sameln
regexp
sed
grep
pipelining

Recall 4 stages of compilation:
Preprocessing (preprocessed code):
- include headers
- replace symbolic constants
Compilation (assembly code):
- Preprocessed code to assembly instructions specific to target processor architectire
Assembly (Object code)
- To object code. Output of actual instructor instructions
???

gcc -Wall shpppingList.c item.c shop.c -o shop
MAKEFILE example: (know how to write)
////
//To make the shop executable file:
all: shop //will be called with make all (this is our target for making); to make all, want to make shop executeable files
shop: item.o shoppingList.o shop.o //3 prerequisites to make shop
g++...
item.o: item.cpp item.hpp
g++ -g -Wall -c item.cpp
shoppingList.o: shoppingList.cpp shoppingList.hpp
g++...
shop.o: shop.cpp item.o shoppingList.o
...
FORMAT:
object file: prereqs
compiling command
///
diff/patch file structure REVIEW
REVIEW SLIDES
-p is to strip n number of leading slashes

python is an interpreted language, scripting lang
- execute directly and freely by interpreter
- no need for previous compilation
REVIEW shuf.py

REVIEW GDB
(gdb) break file1.c:6 (break at line 6)
(gdb) break my_function
(gdb) delete [breakpoint id]
Stack: push frame for func invoke, pop frame for func return, local vars, return addresses
Heap: dynamic memory allocation
Data types, pointers, structs, etc...
void *malloc(size)t size); - allocates size bytes and returns pointer to allocated memory

///FINAL EXAM SOL:///
Problem 1:

1) Since I know the commands that are used to make directories, mkdir. Also I know touch is used to make files. I would look at the man page for mkdir and touch, looking for tags and options that would allow me to do what I want to.

2) ^def\s[a-z|A-Z|_][a-z|A-Z|0-9|_]*\([a-z|A-Z|_][a-z|A-Z|0-9|_]*\):$

3) Use recipients public key to encrypt, they use their own private key to decrypt, and vice versa for you. For digital dignature, encrypt with your own private key and the signature is verified by the recipient with your public key.

Problem 2:

1) For Contact.txt: chmod 664 /projects/team-1/Contact.txt
For README: chmod 640 /projects/team-1/README (assuming that no one else is allowed to access the README)
For MeetingNotes.txt: chmod 660 /projects/team-1/MeetingNotes.txt

2) git log //to get log of all commits
//From there, locate the commits based on date from last week and get commit IDs
git diff <commit id> HEAD > patchFileN.diff //for N different commits, so that you have various different patches
//Make multiple branches off of that point to test each patch individually
git format-patch -1 <commit ID> --stdout > patchFileN.diff //this is a better way, expecially for below

***A patch created with git format-patch will also include some meta-information about the commit (committer, date, commit message, ...) and will contains diff of binary data. Everything will be formatted as a mail, so that it can be easily sent. The person that receive it can then recreate the corresponding commit with git am and all meta-data will be intact. It can also be applied with git apply as it is a super-set of a simple diff.

A patch crated with git diff will be a simple diff with context (think diff -u). It can also be applied with git apply but the meta-data will not be recreated (as they are not present).

In summary, git format-patch is useful to transmit a commit, while git diff is useful to get a diff between two trees.***

3) git format-patch -1 HEAD(or current commit ID) --stdout > formatted-patch.txt //this formats it into a mail format that will allow all recipients to receive and apply the commit in similar conditions
git add foo.c
git commit -m "Update to foo.c"
git push -u origin master
git pull origin master //for all else to get most updated version of the code, pull from remote master to local origin/

Problem 3:

1) shell script that takes in 2 arguements $1 and $2

```
#!/bin/sh
school1=$1
school2=$2
cat nba.txt | grep $school1 > school1.txt
cat nba.txt | grep $school2 > school2.txt
N1=`cat school1.txt | wc -l`
N2=`cat school2.txt | wc -l`
if [ $N1 -gt $N2 ]
then
    echo "0"
    exit 0
else
    echo "1"
    exit 1
fi
```

2)

```
#!/bin/sh

cat nba.txt | grep '/' > players.txt
cat players.txt | sed 's/.*\t//g' > names.txt
```

3)

```
#!/bin/sh

cat nba.txt | grep 2003 | sed 's/^\([^\t]*\t\)\{4\}//g' | sort -k1 -nr  > scorers.txt
```

Problem 4:

1)

```
a_list.append(tuple((3, 4)))
```

***tuple(my_str.split(';')[:-1])
('str1', 'str2', 'str3')
You split the string at the ; character, and pass all off the substrings (except the last one, the empty string) to tuple to create the result tuple.***

2)
Potential error/leak in information: It doesn't seem the information is at all encrypted, adn you only need username to create a "valid" query to the database.

3)
Dynamically linked in Makefile, so updates will not have to be recompiled and so it should be safe in that regard

4)
???