CPSC 4240/6240 Discovery 1 50 points

Instructions:

This homework is done *individually*, not in teams. You are allowed to use search engines, textbook/s, lecture notes, and any other non-human sources you wish. BUT you are *not* allowed to consult others, or help other students with their work, give hints or solutions to anyone in the class. The reason it is called a *Discovery* is because you will do some hands-on exercises and show what you have accomplished. Each exercise may have several questions, so please read carefully. Please do not take shortcuts. Thoroughly explain what you have discovered. This is not the case when "less is more".

You may need to take a screenshot or two (no, not with your phone, with your keyboard) to illustrate your work. All screenshots should be clearly legible and illustrate without a doubt what you are doing. You can open them in an image editor and trim off the parts you do not need, so that they take less space. Insert them when answering the question, do not submit them separately as image files. Make space between the questions in this document and type your answers there. Please do not type in red.

Computer Science is attention to detail, so let's make your work look nice and professional, even though *it does not have to conform to MLA, APA, or ACM format.*

What and how to submit

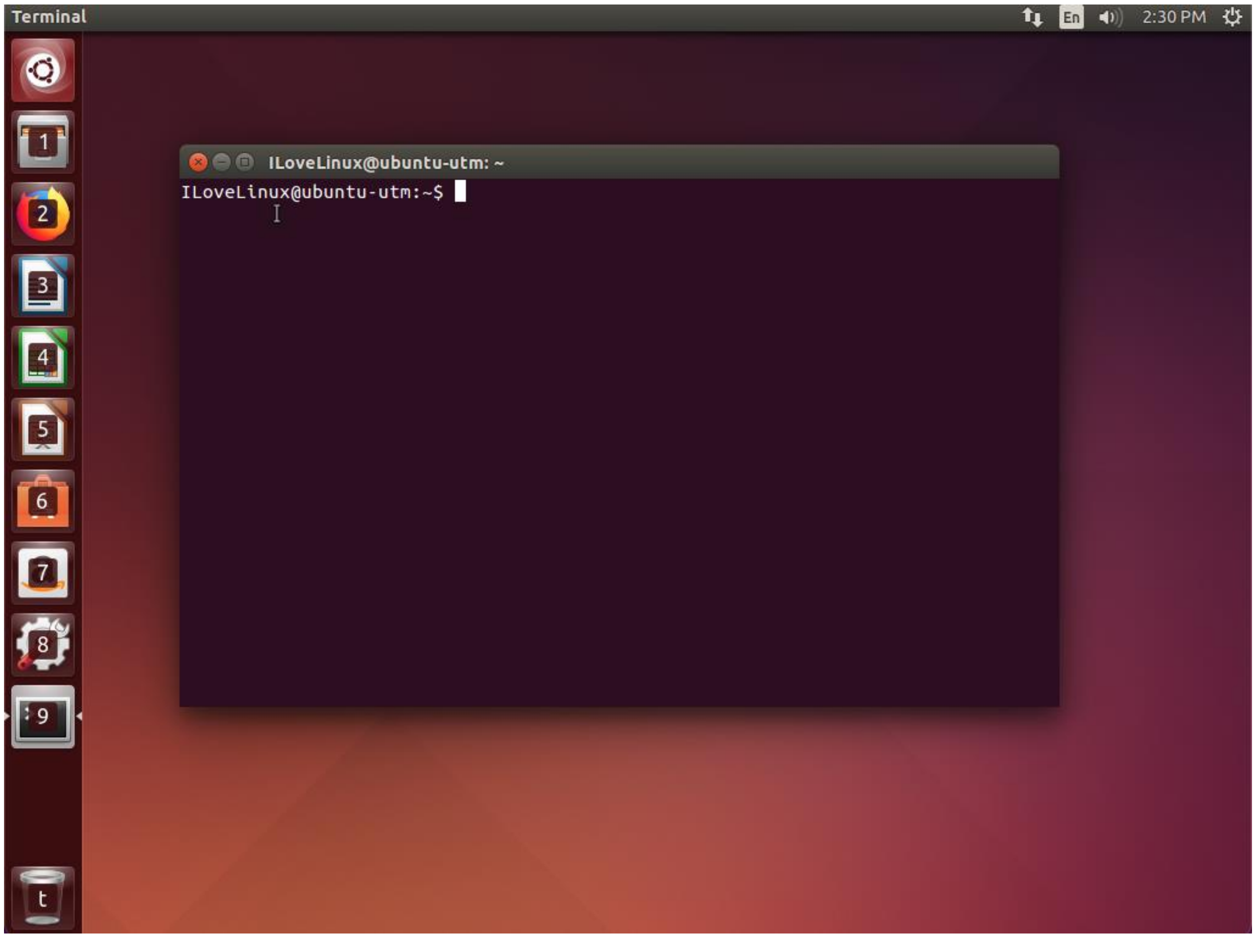
Submit your .pdf file to canvas before the due date. All screenshots illustrating your work should be part of the document, not submitted separately.

Grading and Points

Every question indicates how many points it is worth. 4000-level and 6000-level are graded differently, with points indicated as (x/y), where x is 4240 and *y* is 6240.

Exercises

1. This one is a fun and useful exercise. Change the login prompt that will last across boots, so that instead of the default *user@user-VirtualBox:~$* (or whatever your default prompt is) it shows: *ILoveLinux:~$* (you can change it back afterwards) (6/5)



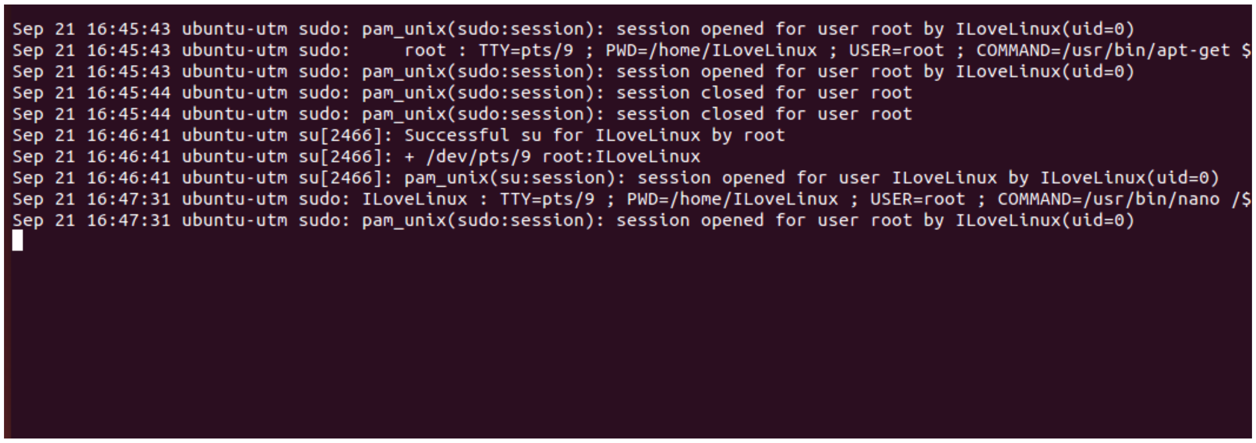
```
ILoveLinux@ubuntu-utm:~$
```

2. Please explore SetUID/SetGID programs. What are they, and what is special about them? Find two such programs that run on Linux (other than passwd). Are these programs a security risk? Why yes, or why not? (7/6)

   SetUID and SetGID are permission bits in executable files. When they are set in said files and the kernel runs, the effective UID or GID of the resulting process is changed to the UID or GID of the file containing the program image rather than the UID and GID of the user running the command. This is special because it promotes the user's privileges for the execution of that command. So users may execute programs with a level of access that matches the user who owns the file. SetGID is the equivalent of SetUID but for groups. Some example programs that have these bits are *sudo and crontab* has SetGID. However, programs that run SetUID are a security risk, especially if they run SetUID to root. This is because the files with SetUID and SetGID may give an unauthorized user root access or even access to run a program in another user's name.

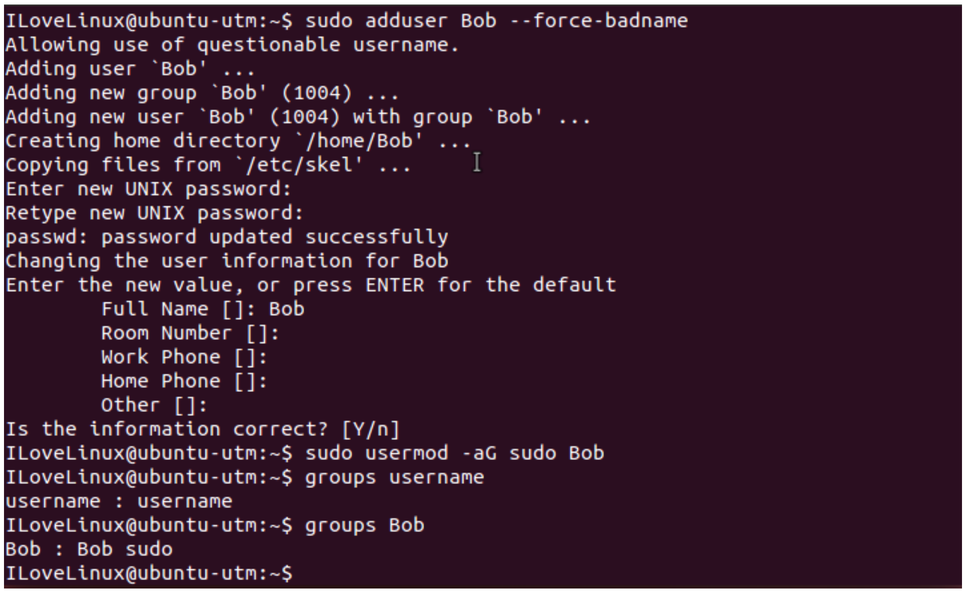
3. Run several commands with sudo. What information is being logged, and if it is, where? Show a screen shot of the log page. Explain what the entry contains. Now create a privileged user Bob, su to that user and run some privileged commands. What is logged and where? Please insert a couple of screenshots to illustrate what you are doing here. (7/6)

   In Ubuntu all sudo activities are logged in the /var/log/auth.log file. This file logs all of the commands executed in the terminal along with which user executed the command and the date and time when it was executed. It also shows when sessions were opened and closed by a user. Below is a screenshot of the log page where I executed some sudo commands.



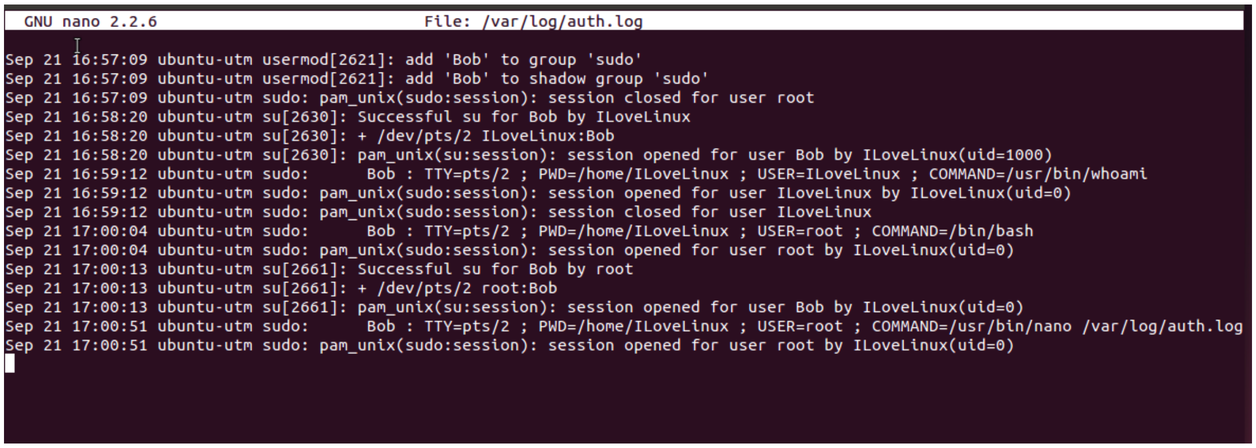
```
Sep 21 16:45:43 ubuntu-utm sudo: pam_unix(sudo:session): session opened for user root by ILoveLinux(uid=0)
Sep 21 16:45:43 ubuntu-utm sudo:     root : TTY=pts/9 ; PWD=/home/ILoveLinux ; USER=root ; COMMAND=/usr/bin/apt-get $
Sep 21 16:45:43 ubuntu-utm sudo: pam_unix(sudo:session): session opened for user root by ILoveLinux(uid=0)
Sep 21 16:45:44 ubuntu-utm sudo: pam_unix(sudo:session): session closed for user root
Sep 21 16:45:44 ubuntu-utm sudo: pam_unix(sudo:session): session closed for user root
Sep 21 16:46:41 ubuntu-utm su[2466]: Successful su for ILoveLinux by root
Sep 21 16:46:41 ubuntu-utm su[2466]: + /dev/pts/9 root:ILoveLinux
Sep 21 16:46:41 ubuntu-utm su[2466]: pam_unix(su:session): session opened for user ILoveLinux by ILoveLinux(uid=0)
Sep 21 16:47:31 ubuntu-utm sudo: ILoveLinux : TTY=pts/9 ; PWD=/home/ILoveLinux ; USER=root ; COMMAND=/usr/bin/nano /$
Sep 21 16:47:31 ubuntu-utm sudo: pam_unix(sudo:session): session opened for user root by ILoveLinux(uid=0)
```

Here is me creating the new privileged user Bob:



```
ILoveLinux@ubuntu-utm:~$ sudo adduser Bob --force-badname
Allowing use of questionable username.
Adding user `Bob' ...
Adding new group `Bob' (1004) ...
Adding new user `Bob' (1004) with group `Bob' ...
Creating home directory `/home/Bob' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for Bob
Enter the new value, or press ENTER for the default
        Full Name []: Bob
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n]
ILoveLinux@ubuntu-utm:~$ sudo usermod -aG sudo Bob
ILoveLinux@ubuntu-utm:~$ groups username
username : username
ILoveLinux@ubuntu-utm:~$ groups Bob
Bob : Bob sudo
ILoveLinux@ubuntu-utm:~$
```

After running a few privileged commands as Bob, we can now go back and check the same log as before. Just as before it logs the same information, but now we can see the commands that were executed as Bob.



```
 GNU nano 2.2.6                            File: /var/log/auth.log

Sep 21 16:57:09 ubuntu-utm usermod[2621]: add 'Bob' to group 'sudo'
Sep 21 16:57:09 ubuntu-utm usermod[2621]: add 'Bob' to shadow group 'sudo'
Sep 21 16:57:09 ubuntu-utm sudo: pam_unix(sudo:session): session closed for user root
Sep 21 16:58:20 ubuntu-utm su[2630]: Successful su for Bob by ILoveLinux
Sep 21 16:58:20 ubuntu-utm su[2630]: + /dev/pts/2 ILoveLinux:Bob
Sep 21 16:58:20 ubuntu-utm su[2630]: pam_unix(su:session): session opened for user Bob by ILoveLinux(uid=1000)
Sep 21 16:59:12 ubuntu-utm sudo:      Bob : TTY=pts/2 ; PWD=/home/ILoveLinux ; USER=ILoveLinux ; COMMAND=/usr/bin/whoami
Sep 21 16:59:12 ubuntu-utm sudo: pam_unix(sudo:session): session opened for user ILoveLinux by ILoveLinux(uid=0)
Sep 21 16:59:12 ubuntu-utm sudo: pam_unix(sudo:session): session closed for user ILoveLinux
Sep 21 17:00:04 ubuntu-utm sudo:      Bob : TTY=pts/2 ; PWD=/home/ILoveLinux ; USER=root ; COMMAND=/bin/bash
Sep 21 17:00:04 ubuntu-utm sudo: pam_unix(sudo:session): session opened for user root by ILoveLinux(uid=0)
Sep 21 17:00:13 ubuntu-utm su[2661]: Successful su for Bob by root
Sep 21 17:00:13 ubuntu-utm su[2661]: + /dev/pts/2 root:Bob
Sep 21 17:00:13 ubuntu-utm su[2661]: pam_unix(su:session): session opened for user Bob by ILoveLinux(uid=0)
Sep 21 17:00:51 ubuntu-utm sudo:      Bob : TTY=pts/2 ; PWD=/home/ILoveLinux ; USER=root ; COMMAND=/usr/bin/nano /var/log/auth.log
Sep 21 17:00:51 ubuntu-utm sudo: pam_unix(sudo:session): session opened for user root by ILoveLinux(uid=0)
```

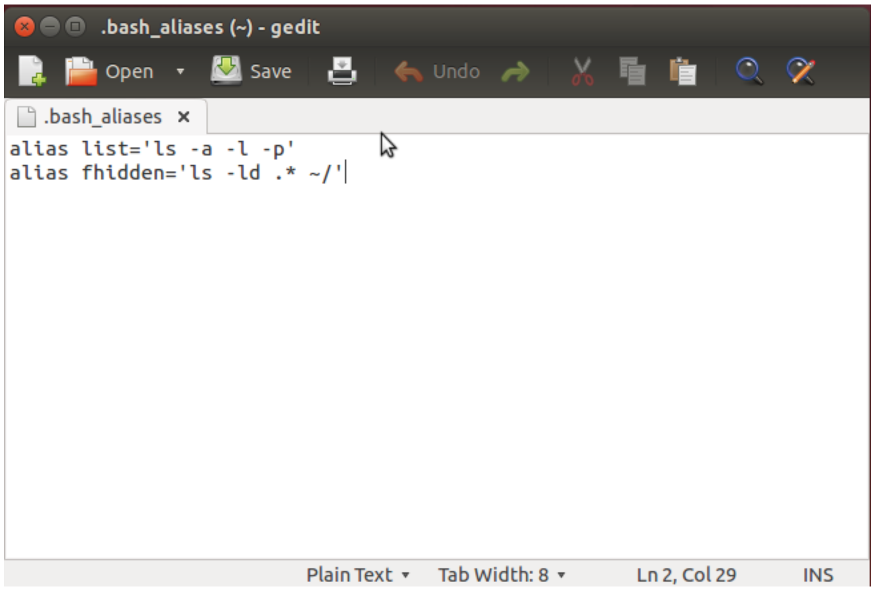
4. Research aliases. Create aliases that persist across boots that do the following: (6/5)

    a. Alias *list* will list all files/subdirectories in the current directory (should work in any directory, so do not hard-code the path), even hidden files that start with a dot, in long format, with a slash indicating if an entry is a directory.

    b. Alias *fhidden* lists only hidden files in your home directory in long format

    c. Place these aliases into the appropriate file, so that they work every time you boot the system. What file is this? Open that file in an editor, take a screenshot and insert below.

In order to create aliases that persist across boots, we have to put the aliases in the .bash_aliases file for later versions of Ubuntu, for other versions the alias can go into .bashrc. After creating the .bash_aliases file (my .bashrc file contained setup information allowing me to do that), we can add our new aliases to that file.

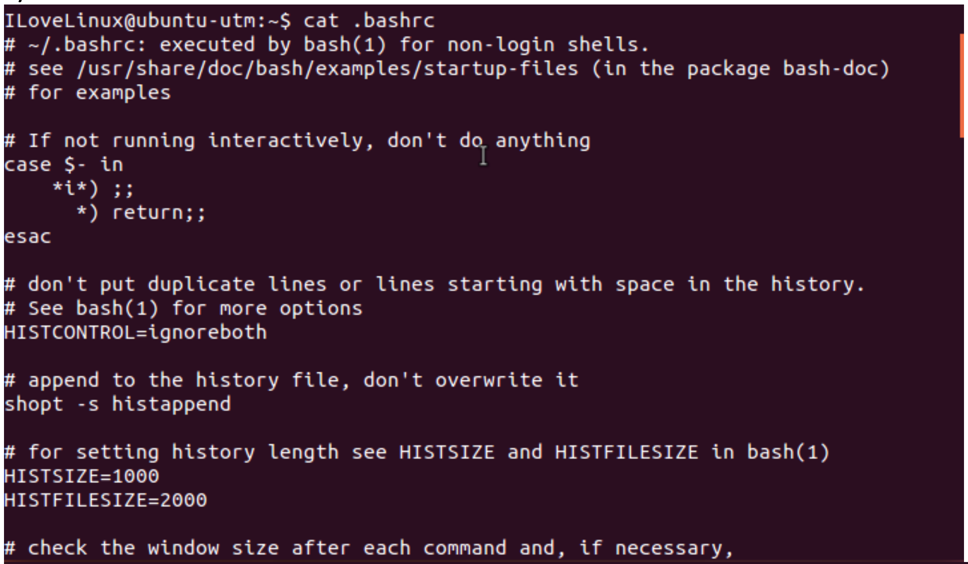


```
alias list='ls -a -l -p'
alias fhidden='ls -ld .* ~/'
```

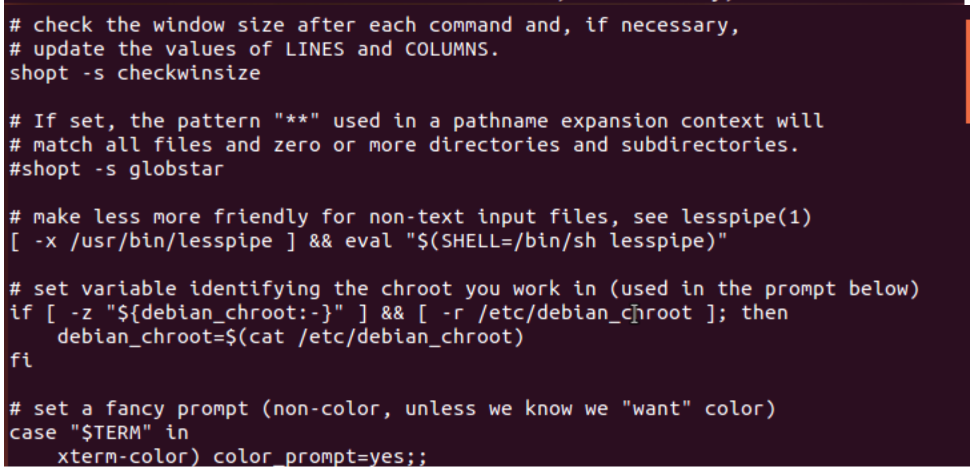
5. Investigate these files: *.profile*, *.bashrc*, and *.bash_profile*. Why do they start with a dot? What is the role of each of these files, and when do they get executed. What are the contents of those files? Screenshots please. (8/6)

Each of these files starts with a dot because they are hidden files in Linux.

.bashrc is a bash shell configuration file and it gets executed every time a user starts a fresh terminal session on their system. Its role is that it contains the set of data that defines all of the configurations for a terminal session. The interactive non-login shell executes the .bashrc file. The contents of the file contain the configurations for the interactive terminal session on the system. Below are some screenshots:

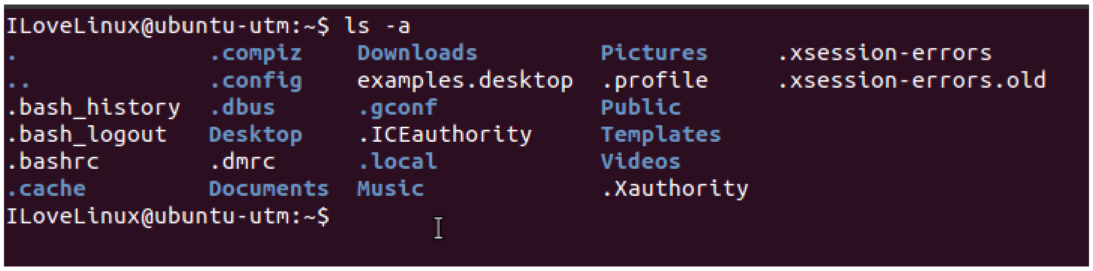


```
ILoveLinux@ubuntu-utm:~$ cat .bashrc
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
    *i*) ;;
      *) return;;
esac

# don't put duplicate lines or lines starting with space in the history.
# See bash(1) for more options
HISTCONTROL=ignoreboth

# append to the history file, don't overwrite it
shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000

# check the window size after each command and, if necessary,
```



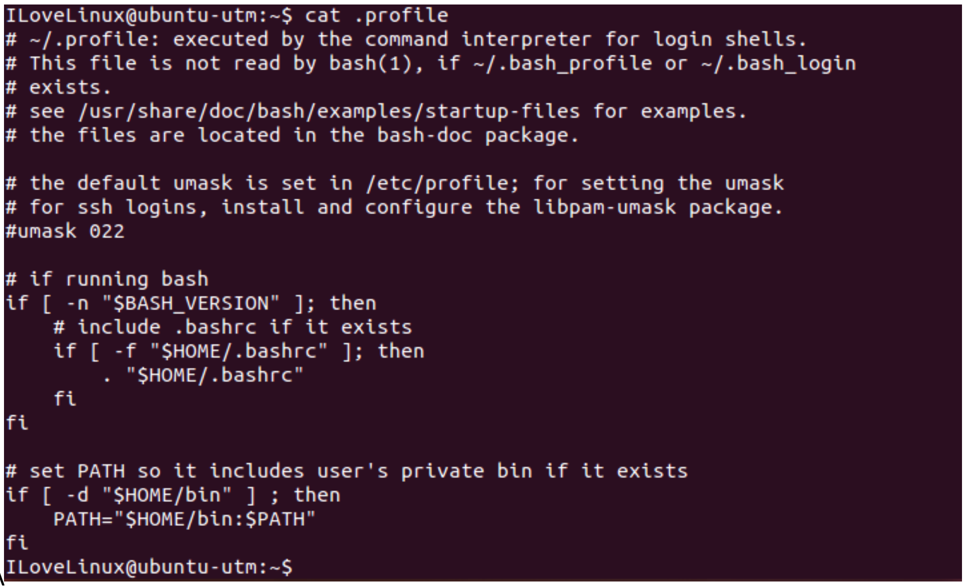
```
# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
shopt -s checkwinsize

# If set, the pattern "**" used in a pathname expansion context will
# match all files and zero or more directories and subdirectories.
#shopt -s globstar

# make less more friendly for non-text input files, see lesspipe(1)
[ -x /usr/bin/lesspipe ] && eval "$(SHELL=/bin/sh lesspipe)"

# set variable identifying the chroot you work in (used in the prompt below)
if [ -z "${debian_chroot:-}" ] && [ -r /etc/debian_chroot ]; then
    debian_chroot=$(cat /etc/debian_chroot)
fi

# set a fancy prompt (non-color, unless we know we "want" color)
case "$TERM" in
    xterm-color) color_prompt=yes;;
```

.bash_profile is another bash shell script file, but it gets executed every time a user logs into a system. It contains the data that defines all of the configurations for the current login shell for the user. Each user has their own .bash_profile file that stores all configurations for the user. Its role is to mainly be used to set up custom environment variables for different users. It is used in the interactive login shell. The contents of the file are the configurations for the current login shell for the particular user. My system did not contain a bash_profile file.



```
ILoveLinux@ubuntu-utm:~$ ls -a
.               .compiz    Downloads          Pictures      .xsession-errors
..              .config    examples.desktop   .profile      .xsession-errors.old
.bash_history   .dbus      .gconf             Public
.bash_logout    Desktop    .ICEauthority      Templates
.bashrc         .dmrc      .local             Videos
.cache          Documents  Music              .Xauthority
ILoveLinux@ubuntu-utm:~$
```

.profile is a file that can hold the same configurations as .bash_profile or .bash_login. .profile is executed by Bash and controls the prompt appearance, keyboard sounds, shells to open, and individual profile settings that would override variables set In /etc/profile file. .profile is executed when .bash_profile or .bash_login is not present in the home directory when there is an interactive shell login. Like .bashrc it also controls variables for how to configure the user's environment.

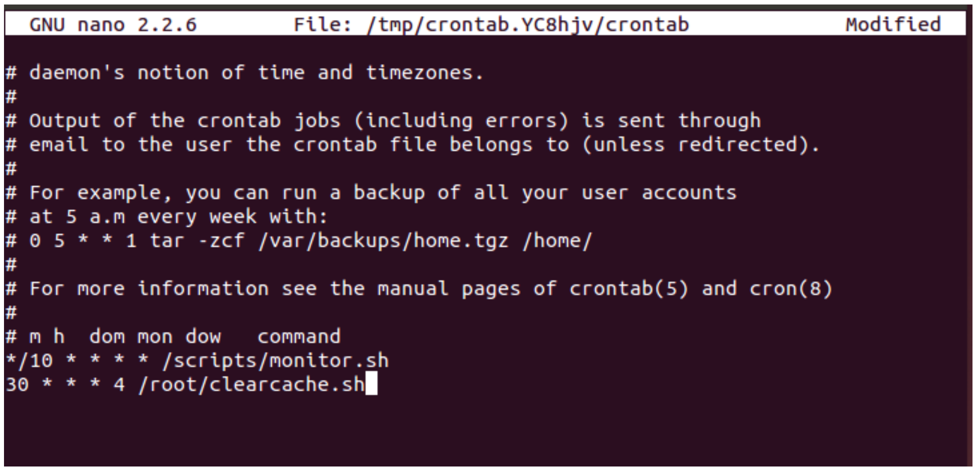


```
ILoveLinux@ubuntu-utm:~$ cat .profile
# ~/.profile: executed by the command interpreter for login shells.
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login
# exists.
# see /usr/share/doc/bash/examples/startup-files for examples.
# the files are located in the bash-doc package.

# the default umask is set in /etc/profile; for setting the umask
# for ssh logins, install and configure the libpam-umask package.
#umask 022

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
ILoveLinux@ubuntu-utm:~$
```

\

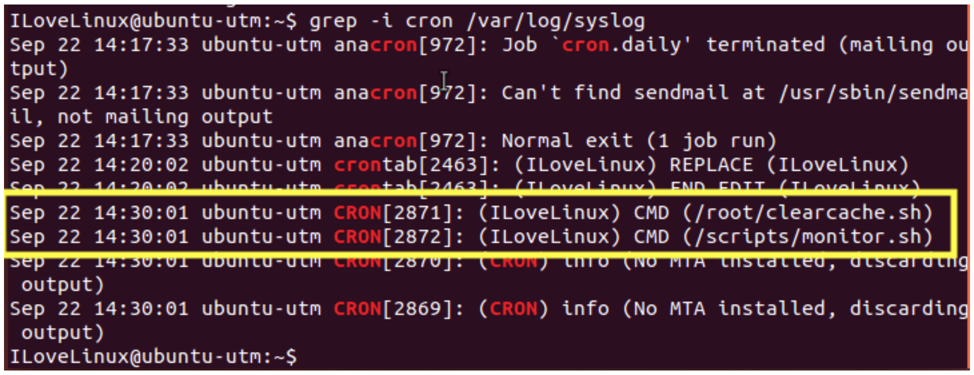
6. Please research two processes - cron and anacron. What are their functions? Now figure out how to schedule periodic tasks using cron, a good one to try is periodic backups of some directory. You can schedule it to be done daily, or weekly, your choice. Schedule two different tasks of your choice to run at a certain interval. Show a screenshot of how/where you have accomplished it and show proof that it worked. (8/8)

   Cron and ancron are both used to automatically run roccuring jobs at scheduled times. Cron runs scheduled jobs at specific intervals but only when the system is running at that moment. Anacron runs the scheduled job even when the computer is off. Once you turn on the system, anacron runs the missed jobs. For my two tasks, I elected to clear the cache every 30 minutes on Thursdays and to perform monitoring every 10 minutes. I scheduled these cron jobs within the crontab file which is in the tmp directory.



```
  GNU nano 2.2.6          File: /tmp/crontab.YC8hjv/crontab             Modified

# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
*/10 * * * * /scripts/monitor.sh
30 * * * 4 /root/clearcache.sh
```

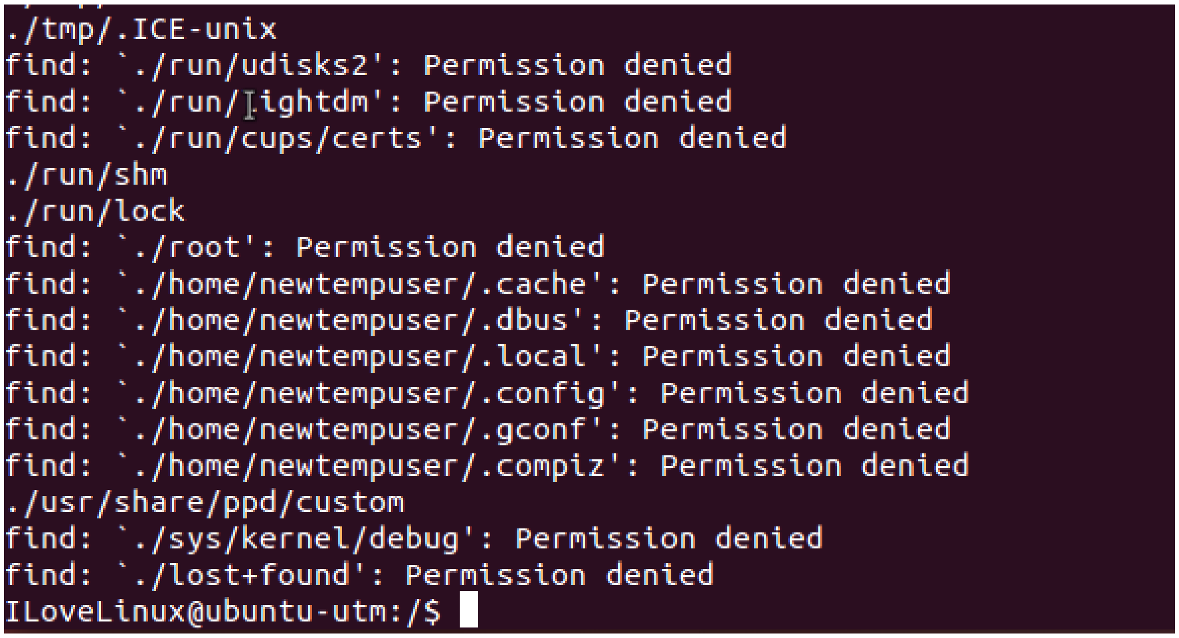
In order to check that the cron jobs ran, we need to open the cron log and then we can see what jobs ran and when. The cron log is located in the system log file syslog in the /var/log/ directory. Looking in that file at a 30 minute mark in the hour, we can see where the 2 cron jobs were executed and I have highlighted the logging of those jobs in a yellow box below.



```
ILoveLinux@ubuntu-utm:~$ grep -i cron /var/log/syslog
Sep 22 14:17:33 ubuntu-utm anacron[972]: Job `cron.daily' terminated (mailing ou
tput)
Sep 22 14:17:33 ubuntu-utm anacron[972]: Can't find sendmail at /usr/sbin/sendma
il, not mailing output
Sep 22 14:17:33 ubuntu-utm anacron[972]: Normal exit (1 job run)
Sep 22 14:20:02 ubuntu-utm crontab[2463]: (ILoveLinux) REPLACE (ILoveLinux)
Sep 22 14:20:02 ubuntu-utm crontab[2463]: (ILoveLinux) END EDIT (ILoveLinux)
Sep 22 14:30:01 ubuntu-utm CRON[2871]: (ILoveLinux) CMD (/root/clearcache.sh)
Sep 22 14:30:01 ubuntu-utm CRON[2872]: (ILoveLinux) CMD (/scripts/monitor.sh)
Sep 22 14:30:01 ubuntu-utm CRON[2870]: (CRON) info (No MTA installed, discarding
 output)
Sep 22 14:30:01 ubuntu-utm CRON[2869]: (CRON) info (No MTA installed, discarding
 output)
ILoveLinux@ubuntu-utm:~$
```

7. Please research the sticky bit and explain what it is used for. Show some examples of how/where it would be useful. (8/8)

   The sticky bit is another permission bit that is set on a file or directory that lets only the owner of said file or directory or the root user delete or rename the file. The sticky bit is commonly used in the /tmp directory. Files in /tmp are frequently created for different users during normal operation and if users could delete each other's /tmp files it could be very damaging to the workings of different applications. Another example would be if I have a folder and two users on my system and both of those users need to add files to that folder. By using the sticky bit, we can ensure that one user cannot go and delete another user's file. Below are some other directories and files with the sticky bit permission:



```
./tmp/.ICE-unix
find: `./run/udisks2': Permission denied
find: `./run/lightdm': Permission denied
find: `./run/cups/certs': Permission denied
./run/shm
./run/lock
find: `./root': Permission denied
find: `./home/newtempuser/.cache': Permission denied
find: `./home/newtempuser/.dbus': Permission denied
find: `./home/newtempuser/.local': Permission denied
find: `./home/newtempuser/.config': Permission denied
find: `./home/newtempuser/.gconf': Permission denied
find: `./home/newtempuser/.compiz': Permission denied
./usr/share/ppd/custom
find: `./sys/kernel/debug': Permission denied
find: `./lost+found': Permission denied
ILoveLinux@ubuntu-utm:/$
```

Graduate students have one additional exercise:

8. How does SHA-512 algorithm work? Please include a brief explanation below, half a page to a page should be enough. What makes is secure? (0/6)

Extra credit question for everyone (5/5)

File */etc/security/limits.conf* allows to set limits to various user parameters, including *nproc* - number of processes that user can create. It allows to specify whether this is a hard or a soft limit. What does it mean – hard or soft limit? What is the difference between the two?

Write a small program with a fork() system call in an infinite loop. What happens? Now set a hard limit for your user to some number of processes (20?). Execute a fork bomb and see what happens. Did it prevent a fork bomb? Please include a screenshot of the changes you made to that file.

(Linux is thy friend. Love thy Linux)