CSCI 340

Lecturer: Dr. Simina Fluture

UNIX HOMEWORK - Part 1 – Handout

No submission yet is required.

INTRODUCTION:

During the course of CSCI 340 we are going to learn about different operating systems, their functionalities, and underlying structure. During the semester the discussion of Operating System Concepts will not be geared to any specific OS. At home most of you use a personal computer, which is preloaded with some version of WINDOWS, and applications for your daily work (i.e. Word Processing, Email, Chat, Internet Browsing). This homework deals with an Operating System that you don't regularly use (at least most of us), and familiarizes you with simple, basic features available in UNIX.

If there are any questions related to this tutorial please email the grader.

You already have a Unix account in ***venus.cs.qc.edu***. You can remotely login using *ssh* or *putty*.
If you don't have *ssh* in your computers, download the client application (the .exe file) from their site. It is useful to have it.
SSH: ftp://ftp.ssh.com/pub/ssh/
putty: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

---

UNIX systems are case sensitive; upper and lower case characters are treated differently.
All commands should be typed in lowercase characters.

When users login, they are placed in a **HOME** directory, which is a portion of the disk space reserved just for them.

UNIX asks for your password. Enter (type) the password you gave when registering for your account. The password will not be displayed.

UNIX checks the username and password you typed, and if correct, the shell will be invoked.
The shell is your interface to the system.
You might get a message similar to the following one:

Password:

```
Last successful login for sfluture: Sat Jun 23 12:49:55 2001 from qcunix1.acc.
Digital UNIX V4.0E  (Rev. 1091); Mon May  7 16:04:49 EDT 2001
```

1. Changing the password (`passwd`)

To each user in UNIX, a password is assigned. To maintain the general security of the system and, in particular, to keep your files secure on the system, you should change your password frequently. Use the passwd utility.

Type: `passwd` ↵

The utility asks for the new password (twice, to make sure you didn't misspell it). Type in the new password. Notice that it will not be displayed (for security reasons).
The user will be asked for the password for every new login session.

2. Getting help (`man, whatis`)

To get help in using any of Unix commands and their options, go to *Unix Reference Manual Pages.* The manual pages are comprised of multi-page, specially formatted, descriptive documentation for every command, system call, and library call in Unix. You can use the `man` command to view the manual page for a specific command.

Type: `man passwd` ↵

To stop between pages,

Type: `man passwd | page` ↵

You can replace `page` with `more`, `less`, `head` or `tail` to see only part of the display.

The head and tail commands:
The head command allows you to see the top part of a file. You may specify the number of lines you want, or default to ten lines. The tail command works like head, except that it shows the last lines of file.

Type: `head –2 "some file you have in your directory"` ↵
To get a short description of what any particular UNIX command does, use the `whatis` command.

Type: `whatis man` ↵

Get familiar with the meaning of other useful commands.
Type: `whatis man passwd finger df` ↵

3. Finding out who you are, what the name of your system is, etc.
(`whoami, who, hostname, uname, finger`)

Type:	`whoami` ↵	(displays your userid)
Type:	`who` ↵	(displays a list of people registered on the system)
Type:	`hostname` ↵	(displays the name of the host computer you have logged into)
Type:	`uname` ↵	(displays information about the operating system running the computer)
Type:	`finger` ↵	(displays information about users on a local or remote host)
Type:	`finger "last name"` ↵	(displays information about a specific user whose last name is entered in the field "last name")

4. Editing Text Files
Files can be edited using text editors (like pico, vi, vim, emacs). You should be familiar with at least one of the editors. If you are not yet, then it is your responsibility to get familiar with one of them. Generally you should choose the editor that you are most comfortable with, in terms of the way you prefer to work with the computer. Your choice of editor depends on the complexity and quantity of text creation and manipulation that you want to do. The more powerful editors such as **vi** and **emacs** are capable of handling complex editing tasks.

5. Viewing contents of a file (cat), Redirection (>, <, >>)

Unix filenames can be up to 14 characters long. Non-alphabetic characters can be used in any position of the filename. Files that begin with a dot are normally hidden when the directory is listed.

You can display the complete contents of one or more files on screen by using `cat` command. The `cat` command sends its output to the console screen. If no arguments are mentioned, `cat` copies from standard input to standard output.

Type: `cat .profile` ↵
The format of this command specifies that the `cat` utility is to use the file .profile as its input, and send the output to the console screen. If the file is larger than one page, the file contents quickly scroll off the display screen.

If the file to be viewed is larger than one page, you can use the `more` command.
Type: `more .profile` ↵

Standard files

There are default files from which a command reads its input and sends its output and error messages. In UNIX these files are known as standard input – the keyboard (stdin), standard output – display screen (stdout) and standard error (stderr). Those default files are opened for you already. Every command by default takes its input from the keyboard and sends its output and error messages to the display screen.

Suppose you want to edit a file, without using a text editor. You can use the `cat` command and output redirection (accomplished by symbol ">")

Type: `cat - > temp` ↵
This specifies that the input is from stdin (- means keyboard), and writes the output to the file called temp (> means dump into).

Type the following lines of text (ignore any typing mistakes, and do not use the cursor arrow keys).

> Small change got rained on by his own 38
> down by the arcade,

Press CTRL D to terminate text entry, and to return to the shell prompt.

The cat utility can also be used to join (concatenate) many files together into a single file.
Edit two files, named first and last. (You can use a text editor or the keyboard).
In first type your first name. For example: "My first name is Marian"
In last type your last name. For example: "My last name is Smith"

Type: `cat first last` ↵
Type: `cat first last > full ; cat full` ↵
Type: `cat last > full ;cat full` ↵

You can redirect without overwriting file contents (appending).
To append output, to the end of a file, replace the > with the >> operator. Also, you can append a file to another existing file.

Type: `cat last >> full; cat full ↵`

`Some command < somefile`

makes the shell read its input from file rather than keyboard.
Some command >`somefile`

causes the shell to place the output from the command in a file called "somefile".

Type: `cat < full; date > dateFile ; cat dateFile ↵`

Special Files

`.cshrc`	executed at each instance of the shell
`.history`	history list saved from previous login
`.login`	executed by login shell after .cshrc at login, for c shell
`.logout`	executed by login shell at logout
`.profile`	executed at login time, setup for shell if Bourne or Korn shell is the login shell

Organizing files and directories.
6. Determining and changing file access privileges (`ls –l, chmod`)

Type: `ls ↵` (displays a list of all the files in the current directory)

UNIX systems extend the power of commands by using special flags or switches. These switches are among the most powerful features of UNIX systems. Switches are preceded with a "-" symbol.

Type: `ls –a ↵` (the display also includes the hidden files)

Type: `ls –la ↵` (the switch l stands for long listing, and the a switch is for all files)

Unix might answer with the following display:

```
     ls -la
     total 162
-rw-------   1 a9cqc    users       5651 Jan 20 23:39 .pinerc
-rw-rw----   1 a9cqc    users          2 Sep 13 19:25 .popbull
-rwx------   1 a9cqc    users       1653 Sep 11 17:23 .profile
drwxr-xr-x   2 a9cqc    users        512 Jan 20 03:10 mail
-rw-------   1 a9cqc    users      47246 Jan 17 01:56 mbox
```

Let's explain what all this means.

```
  permission settings
   |          number of links to file
   |          |      owner of file
   |          |      |      group
   |          |      |      |      size of file in bytes
   |          |      |      |      |      date/time of last modification
   |          |      |      |      |      |             filename
   |          |      |      |      |      |             |
-rwx------      1      a9cqc   users 1653     Sep 11  17:23    .profile
```

The first part reveals the permission settings of the files (-rw-r--r--). These are coded as follows

d	directory
r	read
w	write
x	execute
-	no access

Note that there are three sets of permission settings.

```
specifies whether it is a directory entry
|      permissions for the owner of the file
|      |      permissions for group members
|      |      |      permissions for others
|      |      |      |
-      rw-     ---     ---
```

You can use the chmod command to change access privileges for your files.

Syntax: chmod[options] octal-mode file list
chmod[options] symbolic-mode file-list

Values for Symbolic Mode Components

Who	Operator	Privilege
u User	+ Add privilege	r Read bit
g Group	- Remove privilege	w Write bit
o Other	= Set Privilege	x Execute
a All		
ugo All		

Partial list of Symbolic to Octal Conversions

Symbolic	**Binary**	**Octal**
---	000	0
--x	001	1
-w-	010	2
-wx	011	3
r--	100	4
r-x	101	5

When you create a new file, it's given default permissions. You can control that with `umask` command. Bit patterns for `umask` and `chmod` are reversed, which means that by doing `chmod 777` somefile you give all

permissions to everyone and doing `umask 077` you take away all permissions from group and others for all the files that will be created after execution of `umask`.
Note: newly created files always have their execution bit turned off.

Examples:

`chmod 740 courses` sets access privileges for courses to read, write and execute for the owner, and read only for the group.

`chmod a+x sample` let everyone execute sample

You can use the `–F` option to identify directories, executable files, and symbolic links. The command displays an asterisk (*) after an executable file, a slash (/) after a directory, and symbol (@) after a symbolic link.

7. Copying, Moving, Linking and Removing Files (`cp, mv, rm`)

`cp` (copy)
Syntax `cp [options] file1 file2` copy file1 to file2.

Type: `cp full temp2` ↵

`mv` (move)
Syntax: `mv [options] file1 file2` move file1 to file2 or rename file1 as file2
`mv [options] file-list directory` move all the files in "file list" to directory

Type: `mv temp temp2`↵
This renames the file *temp* to *temp2*.

`rm` (remove) : The `rm` utility is used for erasing files and directories.

Type: `rm temp2` ↵
This removes temp2. Once a file is removed, it cannot be restored.

To cover situations where mistakes might occur, the switch –i, appended to this command, will request a Yes or No response before deleting the file.

Type: `rm –i temp1` ↵

NOTE that switches are written before the filenames. Answer Y to the prompt so that *temp1* is removed.

You can refer to any file by different names in different directories. Command `ln` creates a link which points to that file. Links are simply alternative names for the same file. Since link only points to file, deleting link doesn't affect the file. Any changes made to the file will be reflected when you access that file through any of the links. There are 2 types of links - hard and soft (symbolic) links. You can use a hard link to add a file to a directory
ln somefile some directory.

Hard linked files live in all the directories you link them to. If you delete a file from one of the directories it still exists in any other directories it was linked to.
A symbolic link contains the pathname of the file you wish to create a link to. Symbolic links can tie into any file in the file structure; they are not limited to files within a file system. Symbolic links may also refer to directories as well as individual files.

`ln –s somefile somedirectory`
will make soft link.

8. `pwd` (print working directory)
When a user logs in to a UNIX system, they are located in their own directory space. The `pwd` (print working directory) command displays the complete pathname of the current directory you are in. This is helpful when you want to know exactly where you are.

Type: `pwd` ↵

9. Creating, Removing and Changing Directories (`mkdir, rmdir, cd`)

Syntax (for creating) : `mkdir [options] dirnames`

If you want to create a new directory named CS241

Type: `mkdir CS241` ↵

You can confirm the creation of the new directory by using the `ls` commands.

Syntax (for removing): `rmdir [options] dirnames`

Syntax (for changing directories):

`cd dirname`	change current directory to the specified subdirectory
`cd ..`	change the current working directory to the parent directory
`cd /`	change to root directory
`cd`	change to your home directory

Unix uses a hierarchical file structure: root at the top, subdirectories below.
Standard directories in Unix include:
`/bin`, `/etc`, `/dev`, `/tmp`, `/usr`

When you login, Unix places you directly in your home directory.
$HOME or /usr/users/userid

File manipulation (continuation)
10. Determining file size

`wc` (word count)

This utility displays a count of the number of characters, words and lines in a file.

Syntax: `wc [options] file-list`

Commonly used options/features

`-c`	displays only the number of characters
-l	displays only the number of lines
-w	displays only the number of words

In your working directory edit a file named July_OH that will have the following content:

Office Hours for July 2000

Monday
    9:00 – 10:00 AM
    3:00 – 4:00 PM
Tuesday
    10:00 – 11:00 AM

Type: `wc July_OH ↵`

11. Comparing files (`diff`)

Diff is a very powerful and useful command. Sometimes you will need to compare two versions of source code or some other document to find out where they differ from each other. You can use the `diff` command to perform this task. The command compares two files and displays differences between them in terms of commands that can be used to convert one file to the other.

Syntax: `diff [options] [file1] [file2]`

Use the `man` command to read the options for `diff`, and how the command can be used.

Edit another file named October_OH that will contain the following:

Office Hours for October 2001

Monday
    9:00 – 10:00 AM
    4:00 – 5:00 PM
Wed
    10:00 – 11:00 AM

Type: `diff July_OH October_OH ↵`

12. Searching for command and files (grep, whereis, which, locate, uniq, tr)

`grep`: The `grep` command searches a file for a pattern. The following command

```
grep 'change' temp
```

Searches the file *temp* in the current directory for the text string *change*.

`whereis`: you can use the `whereis` command to find out whether your system has a particular command and, if it does, where it is in the file structure. You typically need to get such information when you are trying to execute a command that you know is a valid command but that your shell can't locate because the directory

containing the executable for the command isn't in the search path. It is a BSD command but most UNIX systems today have it.

Syntax: `whereis [options] [file-list]`

Output: absolute pathnames for the files containing binaries, source codes and manual pages for the commands in "file-list".

`which:` In a system that has multiple versions of a command, the which utility can be used to determine the location of the version that is executed by the shell that you are using when you type the command. The `which` command takes a command-list as argument and returns absolute pathnames for them to standard output.

`locate:` also searches for a specific file name.

`find` : you can use the find command to search a list of directories that meet the criteria described by the expression passed to it as an argument.

Syntax: `find directory-list expression`

Commonly used "criteria" in 'expression':

`-newer` `file`	search for files that were modified after "file"
`-user` `name`	search for files owned by the user name of ID
`-name` `pattern`	search for files that are specified by the 'pattern'
`-exec` `CMD`	the file being search meets the criteria if the command "CMD" returns 0 as its exit status.

Removing repeated lines:

`uniq` : is used to remove all but one copy of successive repeated lines in a file. The command is designed to work on sorted files.

Syntax: `uniq [options][+N][input-file][output-file]`

The input file does not change. If no output file is specified the output of the command is send to standard output. If no input-file is specified, the command takes input from standard input.

`tr` is a filter used to replace one or more characters in your files with other one or more characters.

Syntax:
```
tr[-cs]"string1""string2"
tr –s|-d[-c]"string1"
tr –ds[-c]"string1""string2"
```

Copy stdin to stdout with substitution or deletion of selected characters. The options specified and the "string1" and "string2" command arguments control translations that occur while characters and single-character collating elements are being copied.

Options:

- **-c** complement the set of characters specified by "string1"
- **-d** delete all occurrences of input characters specified by "string1"
- **-s** replace instances of repeated characters with a single character.

13. `cal` (calendar) :This command prints a calendar to the screen.

Piping

14. Piping commands:
This feature is extremely powerful. It means that the output of one utility can be fed directly into the input of another. The | symbol indicates pipelining in UNIX systems.
To illustrate how this works, consider the following command.

Type: `who |sort ↵`

This creates a pipe between the output of utility *who* and the input of utility *sort*. This command runs `who`, displays the users logged on the system, feeding the output into `sort` which prints the sorted list of users on the system to the standard output (your screen).
A pipe inter-connects two processes together.
Note: redirection connects processes and files.

Reading and Writing Shell variables

15. (`set, setenv`)
You can examine the names and values of your shell variables by using the `set` or `setenv` command. The `set` and `setenv` commands are used to assign a string to a variable. The `setenv` command declares and initializes a global variable, the `set` command declares and initializes a local variable.

Typing the `set` command will print out a list of all the current shell environment variables. The majority of these are loaded when you first log in. The first thing the shell does is execute the file .profile in the user's home directory!

Type: `set ↵`

Write down the values of the specified environment variables printed by the `set` command.

`home` .....

`user` .....

`uid` .....

`term` .....

You can assign values to variables.

Type: `set prompt = "CSCI241 :" ↵`

16. Echo can also be used to print out shell variables. (more about echo later)

Type: `set sample="I like this homework" ↵`
`echo $sample ↵`

You can remove specified variables from the environment. BUT BE CAREFUL!!!!!!!

Type: `unset name` ↵

Type: `echo $name` ↵

Processes

Process Attributes

17. ps (process status)

Every UNIX process has several attributes, including owner's ID, process name, process ID, process state, etc.

The `ps` command can be used to view the attributes of processes running on a system.

Type: `ps` ↵

Check a list of the possible displayed states for a process.
Hint: use man ps to check the state encoding.

For example:
R: Runnable

Type: ps –a ↵
displays information about processes executing on your terminal except the session leader (your login shell)

Commonly used options/features:

-l	display long list of the status report
-u *uidlist*	display information about processes belonging to the users with the UIDs in "uidlist"

For example, if you want information about your processes you can enter your username or your user id.

To find out your user id and your group id:

Type: id ↵

A shell is your interface to the Operating System. The system can support different types of shells. You can find out your login shell, the command interpreter that is invoked when you login, by checking the value of the environment variable "shell".
Some systems list the supported shells through the command:
chsh –l
Forbin doesn't allow the "-l" switch for the chsh command.

From one shell you can enter another shell and so on. The login shell remains the same but you, being inside of a different shell, can use commands specific to that shell. The login shell will change to the new shell only after you logout.

You can change the shell using the `chsh` command.

Type: chsh ↵

Try the following possibilities as your new shell: bash, csh, sh, ksh, zsh
After you changed the shell to another possible one:

Type: ps ↵

Observe the displayed output.
Check what the login shell is.
Log out.
Login.
Check again what the login shell is.
Change the shell to the default one.

18. Foreground and Background Processes (fg, bg, suspend)

When you type a command and hit <enter>, the shell executes the command and returns by displaying the shell prompt. While your command executes, you do not have access to your shell, and therefore cannot execute any commands until the current command finishes and the shell returns. In this case the command executes in the foreground, keeping control of the keyboard and display screen.

UNIX allows you to run a command so that while the command executes you get the shell prompt back, thus you can do other work. This capability is called running the command in the background.

Syntax: *commandname* (for foreground execution)
commandname & (for background execution)

Type: `ls -l > dirlist &` ↵

NOTE that the shell prompt immediately returns, and a number in brackets is printed. The & character signals to the shell that the command is to be run in the background.

The number returned by the shell, shown in brackets, is the job number for the process. A job is a process that isn't running in the foreground and is accessible only at the terminal with which it is associated.

The `fg` command allows you to bring a background process to the foreground.

Syntax: `fg [%jobid]`

When the `fg` command is executed without a "jobid", it brings the most recent job to the foreground.

Suspend

While running a command in the foreground, you may need to *suspend* it in order to go back to the shell, do something under the shell, and then return to the suspended process.
For example, let's say that you are in the middle of editing a C program file with vi and you need to see the process' status (more realistically let's say you need to compile another C file to see if some errors have been corrected), you can save changes to the file, suspend vi, execute ps, view the output of ps and return to vi.

You can suspend a process with <^Z>

Type: `ps` ↵

Observe the displayed output.

Type: `vi` ↵

Go in edit mode by pressing the "i" key.

Type the following:

```
#include <stdio.h>
#define SIZE 100

main (int argc, char *argv[ ])
{
```
…

To go in *command mode* press *Esc* key

Press <^Z>
~
~
~

observe the displayed message

Type: `ps` ↵

In general, if you want to see a list of all the suspended jobs

Type: `jobs` ↵

To quit vi from command mode

Type `:q!`

Sleep

Creates a process that waits for a specified time (expressed in seconds).

Type: `sleep 300&` ↵

Check the process status.
Check again the process status after more than 300 seconds. It should be gone.

19. **nice** changes the priority of a command (process).

Use the man command to learn about the nice command.

UNIX Daemons

A daemon is a system process running in the background. Daemons are used in UNIX to offer various types of services to users and handle system administration tasks. The print, e-mail, and finger services are provided via daemons.

ECHO

The echo command displays what follows.

Type: `echo "hello world"↵`

Sequential execution

Syntax: `cmd1; cmd2; …;cmdN`

Type: `date; echo Hello, World! ↵`

Parallel Execution

Syntax: `cmd1& cmd2&…cmdN&`

Type: `date& echo Hello, World! & uname; who ↵`

20. Abnormal termination of processes

You can terminate a foreground process by pressing <^C>

You can terminate a background process in one of two ways:
bring the process to the foreground by using the `fg` command and then pressing <^C> by using the kill command.

The primary purpose of the kill command is to send a system call (software interrupt) to a process.

To get a list of possible signals

Type: `kill –l↵`

The above command returned a list of all signals and their names (on some systems, numbers are not displayed)

Commonly used signal numbers:
Quit
Sure kill
Software signal (default signal number)

Syntax: `kill [-signal_number] PID`

Example:

```
forbin> ps
   PID TTY      S           TIME CMD
 20996 ttyp2    S        0:00.14 -tcsh (tcsh)
 25828 ttyp2    T        0:00.02 pico
forbin> kill -15 25828
forbin>
[1]    Terminated                    pico
```

Upon receiving a signal a process might accept the default action as determined by the UNIX kernel, ignore the signal, intercept the signal and take a user-defined action.

The kill command sends signal number 15 to the process which has the PID specified as an argument. The default action for this signal is termination of the process that receives it. The process can ignore the signal. In order to terminate a process that ignores signals, signal number 9, known as sure kill has to be sent.

Using a single kill command you can terminate a list of processes.
The kill command also works with job numbers (as well as with PIDs). You should check this fact.