SHELL SCRIPT 101

CSC 60

SHELL COMMANDS	
chsh	change shell , do a cat /etc/shells If you don't like the shell, you can change using this shell.
ls	list files, ls –F, ls –l , ls –r, ls -t
cd	change working directory to HOME
cd ..	change working directory to parent
cd /	cd to root
cd /usr/include	cd to /usr/include , this is absolute path
cd ../../sys	change to grandparent folder and then to sys folder, this is relative path
cd /	change working directory to / , / symbol for root
cd ~	~ means HOME
.	current folder one dot
..	parent folder two dots
cp file1 file2	copy file1 to file2
mv file1 file2	rename file1 to file2
mkdir cprog	make directory cprog
mkdir p1 p2 p3	make directory p1 p2 p3
rmdir p1 p2 p3	remove directories p1 p2 3, but they should be empty
rm file	remove file
rm –rf cprog	rm recursively and forcefully starting from cprog
touch first.c	set the modified time to current time. or create first.c
cat file1 file2 > file3	concatenate file1 file2 and store them in file3
cat file1 file2	display contents of file1 followed by file2 on screen
chmod	change permissions rwx for users , groups and others
chgrp	change group
chown	change ownership of a file
passwd	change password
stat file	display information about file or file status
which cmd	displays the path of the command. or locate a command
whereis	
cut filename –d' ' -fn	trims each line using delimiters and field numbers
head	display only the top 10 lines (this is default) -n 5 means display the top 5 lines
diff	
sort	sort –nrku

	n means sort numerically, r means reverse order k column number u means remove duplicates
tail	
date	see various fomats
pwd	print working directory
cal	calendar
more file_name	displays the contents of a file page by page, press q to quit, space bar to next page.
less	
od	octal dump
screen	utility to launch multiple shell sessions in one session. beyond the scope of this lecture.
sleep n	suspend the process for n seconds
split	
tee	copies standard input to standard output and one or more files
join	
uname	print the information about the operating system
spell	
top	display tasks running in the system . press q to quit
tr	translate
umask	set mask , used to set permissions.
wc	character , word and line count
exit	
shift	shift positional parameters
command1 | command2	| pipe
clear	clear the screen
jobs	
ln	ln can create both hardlink and softlink
alias	
bc	calculator
strace command	shows the entire trace of the execution of the command. very useful to trace system calls
test	-f -d -r -w -x -e -s(ize)
seq	seq start step end
2>	strerr
>>	append to a file ex a.out >> output.txt
>	redirect output ex a.out > output.txt
<	redirect input ex cat < scope.c
|	pipe
&	run the program in background
fg	bring the suspended job to the foreground
read -p "Enter your age" age	read input data from the user, it copies in $age SHELL also stores in $REPLY variable
^	beginning of line
$	end of line
xargs	what is the difference between find and

	xargs

LOGIN PROCESS OF BASH

Some notes: http://www.gnu.org/software/bash/manual/html_node/Bash-Startup-Files.html

An interactive shell generally reads from and writes to a user's terminal. When Bash is invoked as an interactive login shell, it first reads and executes commands from the file /etc/profile, if that file exists. After reading that file, it looks for ~/.bash_profile, ~/.bash_login, and ~/.profile, in that order, and reads and executes commands from the first one that exists and is readable.

```
if [ -z "$PS1" ]; then
        echo This shell is not interactive
else
        echo This shell is interactive
fi
```

test command has -z options means the length of the string is zero

When an interactive login shell exits, or a non-interactive login shell executes the exit builtin command, Bash reads and executes commands from the file ~/.bash_logout, if it exists.

find command to search for files

find ~ -name cmds.txt		
find `pwd` -name "*.c"		
find / -name "*.c"		
find `pwd` -user Ankar -name "*.c" | wc -l		
find `pwd` -user sankar | wc -l		
find ~ -type f -name "*.sh" -mtime -3	type f means find files	
find ~ -type f -name "*.c" -atime +3 -maxdepth 2	modified in 3 days , limit to depth to 2 , accessed	
find ~ -type f -name "*.c" -ctime +3	look for *.c files changed more than 3 days	
find `pwd` -ctime -2 -name "*.sh"	changed in the past two days	
find . and find . –print will yield same results.	-print is optional	
find . -exec wc -l {} \;	execute wc –l on each file, {} means substitute each file , ; means each file	
find . 2> /dev/null -exec wc -l {} \;	send stderr to /dev/null	
find c_programs cprog -exec wc -l {} \;	find in directories c_programs and cprog	
find . –size +1M	find files which are more 1M file, +1k means kilpbytes. +1G means gigabytes	
find . -size +1M -exec rm {} \;	rm files which are more 1MB	

grep to search strings in files	
grep main .	search text=main in files in the current directory
grep -r main c_programs cprog	search recursively text=main in c_programs and cprog dirs
grep -Ri main c_programs cprog	ignore case, recursively, there is a difference between r and R
grep -R "main ( )" c_programs cprog	search main ( ) , enclosed in double quotes
grep ^- filename	search all lines starting with - , ^ means begin of a line
grep sh$ filename	search all lines ending with sh, $ means end

SHELL PREDEFINED VARIABLES	
echo $SHELL	prints the shell running
echo $PWD	prints the current working directory
echo $HOME	prints the HOME directory
echo $HOSTNAME	prints the computer name
echo $HISTFILE	prints the list of previously issued commands
echo $BASH_VERSION	prints the version of BASH installed
echo $PPID	prints the parent process ID
echo $UID	
echo $RANDOM Each time this parameter is referenced, a random integer between 0 and 32767 is generated. Assigning a value to this variable seeds the random number generator.	prints a random number
echo $PS1 (see note below)	stores the string to be displayed for the prompt
echo $$	prints the process ID
echo $#	print the number of arguments passed to the script
echo $?	print the value returned by the last command
echo $PS2	
echo $PS3	

date and various formats to print	
prints date **date +%a:%A:%D:%T:%Z:%Y:%F**	**%Y means year in YYYY** **%Z zone** **%T in 24 hour format** **%F means time in YYYY-MM-DD** **%D means yy/mm/dd format** **%a means 3 letter weekday, Mon, Tues,….** **%A means print weekday Monday, Tuesday,**
date –u	**means universal UTC format**

TAR Commands to archive	
tar cvf file.tar file1 file2 folder1	**creates tar c=create, v=verbose, f=next is filename**
tar xvf file.tar	**extract file.tar, x=extract**
gzip	
gunzip	

Commands for Compiling	
make	
gcc source file	**-E means preprocess** **-g means store debugging symbols** **-std=C89, -std=C99 for compiling with C89 standards**
ldd executive_file	**dynamic loading dependencies**
nm executive_file	**name symbols**
gdb executive_file	**gnu debugging tool**
objdump	**objdump the object file**
readelf -S	**read elf format and print the sections**

SED – Stream editor	
sed "s/string_source/string_destination/g"	s is an option with sed to change strings. The format is /sourse/destination/g g means global. g is needed if you need to change all occurrences of source string
	You can even invoke stream editor within vi
sed 's/sity/city/g' city.data	substitute sity with city in the entire file city.data
sed 's/pattern/topattern/Ig' filename	g means global, I means ignore case
sed '/Rock/d' filename	remove all lines containing Rock

AWK programming - record processing	
awk 'BEGIN { variables } conditions {statements} END {statements}	
ls -l | **awk 'BEGIN { d_count = 0 } /^d/ { d_count++ ; } END { print d_count }** '	count directories
cat /etc/passwd | awk 'BEGIN { user_count = 0 } { user_count++ ; } END { print user_count } '	count users
ls -l | **awk 'BEGIN { f_count = 0 } /^-/ { f_count++ ; } END { print f_count }** '	count files

Misc commands	
env	**print environment variables**
set	
unset	
export	**you can save the variables in .profile to store permanently. export COURSE=CSC60**
pushd directory_name	
popd	
baseline	
w	**list of users logged in**
who	
finger	
whoami	**prints the id of the user logged in**
ac -p	**print statistics about users' connect time**
man command	**displays the information about a command. Press space bar to read more pages. q to quit**
hostname	**prints the computer name,**
ping ip_address	**communicates with the machine with IP address. Mostly used for checking if the remote machine is alive or shutdown.**
fg	**brings a process to foreground**
bg	**takes a suspended process to background**
kill -Number process_id	**sends signals to the process ID. kill -9 2356 sends procesd 2356 signal to terminate**
ps	**-e –a –l –f**
du -ah	**h means human readable, a means all files**
df	**displays disk usage of mountable devices**
history	
free	
ssh	
ftp	
crontab	
shutdown	**don't even think about shutting down the system**
mount	

Change Prompt using PS1	
export PS1="\u@\h \w:\!:\T>"	\u means username
	\h means hostname
	\w working directory
	\! history number of the command
	\T current time in 12 hour hh:mm:ss format
	\W basename of the folder without the path
export PS1="\u@\h \w:\!:\t>	\t current time
$PS2, $PS3, $PS4 beyond the scope of this course, You can google it to know it more. But it is not very important	
PROMPT_COMMAND	
export PROMPT_COMMAND="echo ready for	Bash executes the command stored in

next command"	PROMPT_COMMAND before executing PS1.

UMASK settings	
default permissions is set to 666 for files and 777 for directories. umask is set during login time. When a file is created, this umask is negated (bit complimented) and ANDed with the default permissions (666)	say umask is 0077 touch a.txt 0666 & (7700) = 0600. So a.txt will have rw for owner and denies permissions for groups and others
	say umask is 0002 touch b.txt 0666 & (7775) = 0664 , so b.txt will have rw for owner and group, read only for others

GIT (https://www.youtube.com/watch?v=HVsySz-h9r4)	
git init	
git status	
git reset	
git diff	
git remove -v	
git config --help	
git log	
git add <file>	
git add -A	
touch .gitignore	add files to ignore
git commit –m "commit message"	
git --version	

Shell script

User defined stings and variables		
echo "Hello World"	prints Hello World	
greetings="Hello World"	assigns Hello World to variable	
echo $greetings	prints Hello World	
first=kobe	no space around = sign	
first="kobe"	assigns value kobe	double quote is recommended
last="Bryant"	assigns Bryant to variable last	
echo "First name = $first Last Name = $last"	prints **First name = kobe Last Name = Bryant**	
full_name="First name = $first Last Name = $last "	assigns **First name = kobe Last Name = Bryant** to full_name	
echo $full_name	prints value of full_name	
name=$first$last	assigns kobeBryant	concatenates
name=$firstname	there is no variable named firstname, so it assigns null	

meaning of && || ;

when a command successfully executed, it returns 0. Returning zero means true in BASH. If there are errors, the return status is non-zero, which means false. Kind of different from C

cmd1 && cmd2 means execute cmds2 only after cmd1 is successfully executed.
cmd1 || cmd2 means execute cmd1. If it is success, don't execute cmd2. Otherwise execute cmd2
cmd1; cmd2 means execute cmd1. then execute cmd2 no matter what the return status of cmd1

date && ps	both will be executed
data && ps	data is not a command, it returns non-zero return status. So ps will not be executed
data || ps	data is not a command, ps will be executed due to ||
data ; ps	execute ps after data