ECS 40 **UNIX Lecture Notes**

I. UNIX commands you should already know.
   A. **cat** *{fileName}** List contents of file(s).
   B. **cd** *[directoryName]* Change directory.
   C. **cp** *sourceFilename destinationFileName* Copies a file.
   D. **cp {***sourceFilename}+ destinationDirectory* Copies file(s) into a directory.
   E. **lpr** –h [-P*printerName*] *{fileName}+* Prints file(s). –h = without header page.
   F. **ls** –adlR *{filename}* {directoryName}** Lists information about files. –a = all files including those beginning with a period. –d = list directory entries instead of contents. –l = long format for each file. –R = recursively list contents of directory(s).
   G. **mkdir** *{directoryName}+* Creates directory(s).
   H. **mv** *oldFileName newFileName* Renames a file or directory.
   I. **mv** *{fileName}+ directoryName* Moves listed files into the directory.
   J. **pwd** List current working directory.
   K. **rm** –irf {fileName}* Removes file(s). -i = prompts. –r = recursively. –f inhibit prompts.
   L. **rmdir** *{directoryName}+* Removes empty directories.

II. Simple UNIX utilities
   A. **chmod** –R *change {fileName}+* Changes permissions of file(s). –R = recursively. *Change* can be an octal or of the form *cluster*[+-=]*permissions*, where *cluster* can be *u* (user) , *g* (group), *o* (others), or *a* (all); and *permissions* can be *r* (read), *w* (write), or *x* (execute).
   B. **diff** *filename1 filename2* Compares files line by line and produces a list of changes to *filename1* to make it identical to *filename2*. No output indicates that the files are identical. Three kinds of changes listed by diff: additions, deletions, and changes.
      1. Additions
         firstStart **a** *secondStart, secondStop*
         > lines from the second file to add to the first file.
      2. Deletions
         firstStart, firstStop **d** *lineCount*
         < lines from the first file to delete.
      3. Changes
         firstStart, firstStop **c** *secondStart, secondStop*
         < lines in the first file to be replaced

         > lines in the second file to be used for the replacement
         ---
   C. **head** –n {*fileName}** Displays the first *n* lines of file(s). If no *n*, then it defaults to 10.
   D. **less** *{fileName}** List contents of files(s) one screen at a time.
   E. **ln** –s *{filename}+ [directoryName]* Creates a hard link(s) to file(s) in the current directory unless *directoryName* is supplied. –s = create soft (symbolic) link(s), which can refer to files in another file system.
   F. **sort** -tc -r –k *start_field* [, *end_field*] { *fileName}** = Sorts a file in ascending or descending order.
      1. -tc = specify field separator, c, instead of white spaces.

2. -r = descending order.
3. Sort field numbering starts at one. •If no *endField*, then all fields following the *startField* are used for sorting.
4. `sort -t, -r -k 3,4 test.csv` will sort the lines of test.csv in descending order based on the 3rd through 4th fields which are separated by commas.

G. **tail** –n {*fileName}** Displays the last *n* lines of files(s). If no *n*, then it defaults to 10.

H. **wc** –lwc {*fileName}** Counts the number of lines (-l), words (-w), and/or characters (-c) of files(s).

III. Regular expressions used in grep, awk, sed, and vi.

A. . = single character. In filename substitution this a "?"
B. [..] = any single character in the bracket.
C. * = zero or more
D. ^ = beginning of line.
E. $ = end of line
F. \ = The meaning of a metacharacter is inhibited.
G. Extended regular expressions. Used by awk and grep -E
   1. c+ matches one or more occurrences of character c
   2. c? matches zero or more occurrences of character c
   3. *expression1* | *expression2* matches *expression1* or *expression2*
   4. (*expr1* | *expr2*) *expr3* matches *expr1 expr3*, or *expr2 expr3*

IV. UNIX Utilities.

A. **grep** -ilnvx *pattern { fileName}** = Displays all lines in the file list that contain the pattern.
   1. if no filename then standard i/o
   2. -i = ignore case.
   3. -l = just list of files displayed.
   4. -n = line numbers.
   5. -v = display all lines that don't match the pattern.
   6. -x = line must be an exact match.
   7. Examples

```
% cat file1.cpp
#include <stdio.h>
% grep "std" file1.cpp
#include <stdio.h>
% grep "Std" file1.cpp
% grep -i "Std" file1.cpp
#include <stdio.h>
% grep -x "std" file1.cpp
% grep -x "#include <stdio.h>" file1.cpp
#include <stdio.h>
% grep -x "#include <Stdio.h>" file1.cpp
% grep -xi "#include <Stdio.h>" file1.cpp
#include <stdio.h>
% grep -xin "#include <Stdio.h>" file1.cpp
1:#include <stdio.h>
% grep -lxin "#include <Stdio.h>" file1.cpp
file1.cpp
```

B. **find** *pathList expression* = Recursively descends through *pathList* and applies *expression* to every file.

1. -name *pattern* = True if the file's name matches *pattern*, which may include shell metacharacters *, []. ?
2. -print = prints out the name of the current file and returns true if **previous** criteria have been met.
3. -type *ch* = True if the type of the file is *ch* (f = ordinary file, d= directory)
4. -exec *command* = True if the exit code from executing *command* is 0. *Command* must be terminated by an escaped semicolon (\;). If you specify {} as a command line argument it is replaced by the name of the current file.
5. -perm octal
6. -size [+-]size_int[c] if c then size in characters, else in 512 byte blocks. + = greater than - = less than specified size
7. -o Short circuiting or. Use \(and \) to surround the clause.
8. [-a] = Short circuiting and. Default, so not needed.
9. find . \(-name '*.c' -o -name '*.txt' \) –print
10. Examples

```
% find .
.
./hw2
./hw2/file2.cpp
./hw2/file1.cpp
./file1.cpp
%
% find . -print
.
./hw2
./hw2/file2.cpp
./hw2/file1.cpp
./file1.cpp
%
% find . -name file1.cpp
./hw2/file1.cpp
./file1.cpp
%
% find . -type d
.
./hw2
%
% find . -type f
./hw2/file2.cpp
./hw2/file1.cpp
./file1.cpp
%
% find . -exec cat {} \;
cat: .: Is a directory
cat: ./hw2: Is a directory
#include <stdio.h>
#include <iostream.h>
#include <stdio.h>
%
%
```

```
find . -exec cat {} \; -print
cat: .: Is a directory
cat: ./hw2: Is a directory
#include <stdio.h>
./hw2/file2.cpp
#include <iostream.h>
./hw2/file1.cpp
#include <stdio.h>
./file1.cpp
%
% find . -print -exec cat {} \;
.
cat: .: Is a directory
./hw2
cat: ./hw2: Is a directory
./hw2/file2.cpp
#include <stdio.h>
./hw2/file1.cpp
#include <iostream.h>
./file1.cpp
#include <stdio.h>
%
% find . -exec grep "stdio.h" {} \; -print
grep: .: Is a directory
grep: ./hw2: Is a directory
#include <stdio.h>
./hw2/file2.cpp
#include <stdio.h>
./file1.cpp
%
% find hw2 -exec grep "stdio.h" {} \; -print
grep: hw2: Is a directory
#include <stdio.h>
hw2/file2.cpp
%
```

C. **tar** -cxtzZvf *[tarFileName] fileList*
   1. -c = create
   2. -f = filename
   3. -t = generates a table of contents.
   4. -v = verbose mode
   5. -x = extract mode
   6. -z = Compress (with -c) or decompress (with –x) using gzip.
   7. -Z = Compress (with –c) or decompress (with –x) using compress.`tar -czf Old40Files.tar.gz 40/f02 40/s02` would create an archive file named `Old40Files.tar.gz` that would contain all of the files in `40/f02` and `40/s02` directories. After being created the file would have been compressed by gzip.

V. Shells = a program that is an interface between a user and the raw operating system.
   A. Built-in commands vs. executable files. **cd** and **umask** are built-in commands. Use **`which`** utility to determine directory that contains the utility.
   B. Types and Selecting
      1. Bourne ($), Korn ($), C(%), tcsh(%) shells. echo $SHELL /pkg/bin/tcsh
      2. chsh doesn't work in CSIF; Ctrl-D doesn't work with tcsh.
      3. Start with sh, ksh, csh, tcsh

C. Shell Variables
   1. Hold values in string format
   2. Local variables only exist in the shell in which they are defined.
   3. Environment variables are passed on to subshells.
      a. To see them use printenv

```
% printenv  // edited by Sean
LESSOPEN=|/usr/bin/lesspipe.sh %s
HISTSIZE=1000
HOSTNAME=pc7.cs.ucdavis.edu
LOGNAME=davis
HISTFILESIZE=1000
MAIL=/var/mail/davis
TERM=xterm
HOSTTYPE=i386-linux
PATH=/home/davis/bin:.:/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/b
in:/usr/local/bin:/pkg/bin:/usr/local/jdk1.2.2/bin:/altpkg/java/b
in
HOME=/home/davis
SHELL=/pkg/bin/tcsh
USER=davis
QTDIR=/usr/lib/qt-2.1.0
DISPLAY=:0
LANG=en_US
OSTYPE=linux
SHLVL=1
LS_COLORS=no=00:fi=00:di=01;34:ln=01;36:pi=40;33:so=01;35:bd=40;3
3;01:cd=40;33;01:or=01;0m=01;35:*.png=01;35:*.tif=01;35:
COLORTERM=
VENDOR=intel
PWD=/home/davis/40/temp
GROUP=users
HOST=pc7.cs.ucdavis.edu
```

      b. PATH is the list of directories that are searched for executables.
      c. Uppercase are copies of lowercase. Lowercase are passed to subshells.
      d. Uppercase Vs Lowercase Examples

```
% echo $user $USER $group $GROUP
davis davis users users
% set user = fred
% echo $user $USER $group $GROUP
fred fred users users
% set USER = derf
% echo $user $USER $group $GROUP
fred derf users users
% setenv GROUP smith
% echo $user $USER $group $GROUP
fred derf smith smith
% set GROUP = jones
% echo $user $USER $group $GROUP
fred derf smith jones
% tcsh
% echo $user $USER $group $GROUP
fred fred smith smith
% set user = sean
% setenv GROUP ECS
% echo $user $USER $group $GROUP
sean sean ECS ECS
```

```
% ps
  PID TTY          TIME CMD
 9327 pts/0    00:00:00 tcsh
 9361 pts/0    00:00:00 tcsh
 9382 pts/0    00:00:00 ps
% exit
exit
% ps
  PID TTY          TIME CMD
 9327 pts/0    00:00:00 tcsh
 9383 pts/0    00:00:00 ps
% echo $user $USER $group $GROUP
fred derf smith jones
   %
```

D. Use of \ to extend a line. Don't follow with a space.

```
% echo this is just a very\
? long line
this is just a very long line
% echo test again\
? this
test again this
% echo test again \
? another
test again another
```

E. Filename substitution
   1. Wild cards are *, ?, [ ]
   2. Done by the shell not the utilities themselves. Thus all programs can use it anywhere on a command line..
   3. Act of pattern replacement is called globbing. Use "set noglob" to stop globbing. Use "unset noglob" to start globbing again. Use single or double quotes to stop globbing.

F. Quoting
   1. Grave accents: A command surrounded by grave accents " ` " not apostrophe or single quote " ' " is replaced by its standard output which is called "command substitution".
   2. Single quotes inhibit wildcard replacement, variable substitution, and command substitution.
   3. Double quotes inhibit wildcard replacement only
   4. Example session:

```
% ls
src           testfile      testfile.txt
% echo $home ls *.txt
/home/davis ls testfile.txt
% echo `$home` `ls *.txt`
/home/davis: Permission denied.
testfile.txt
% echo '$home' 'ls *.txt'
$home ls *.txt
% echo "$home" "ls *.txt"
/home/davis ls *.txt
```

G. Redirection
   1. Standard input is the default input data stream. Normally attached to the terminal.
   2. Standard output is the default output data stream. Normally attached to the monitor.
   3. Standard error is another output data stream used for error messages. Normally attached to monitor.

4. Redirection metacharacters enable the user to change the connections of these streams.
   a. > = write stdout to file (must not exist); >> = append stdout to file (file must exist); < = read stdin from a file; **>!** = writes even if file exists; **>&** = write stdout and stderr to file (must not exit); **>&!** = writes stdout and stderr to file even if file exists; | = sends the output of one process to the input of another process.
5. Examples
   a. Normal standard out standard error with no errors.

```
% find . -name ecs40 -print
./News/ecs40
./public_html/ecs40
./ecs40
```

   b. Normal with errors.

```
% chmod 0000 ecs50
% !find
find . -name ecs40 -print
./News/ecs40
./public_html/ecs40
find: ./ecs50: Permission denied
./ecs40
```

   c. Redirected standard out with errors.

```
% find . -name ecs40 -print > findstdout
find: ./ecs50: Permission denied
% cat findstdout
./News/ecs40
./public_html/ecs40
./ecs40
```

   d. Redirected standard out with errors. again

```
% find . -name ecs40 -print > findstdout
findstdout: File exists.
```

   e. Redirected standard out with errors again with overwrite:

```
% find . -name ecs40 -print >! findstdout
find: ./ecs50: Permission denied
```

   f. Redirected standard out and standard error with errors

```
% find . -name ecs40 -print >&! findstdout
% cat findstdout
./News/ecs40
./public_html/ecs40
find: ./ecs50: Permission denied
./ecs40
```

   g. Redirected standard out with errors using append.

```
% find . -name ecs40 -print >> findstdout
find: ./ecs50: Permission denied
% cat findstdout
./News/ecs40
./public_html/ecs40
find: ./ecs50: Permission denied
./ecs40
./News/ecs40
./public_html/ecs40
./ecs40
```

h. Redirected standard out and standard error to two different files

```
% rm find*
% (find . -name ecs40 -print > findstdout) >&
findstderr
% cat findstdout
./News/ecs40
./public_html/ecs40
./ecs40
% cat findstderr
find: ./ecs50: Permission denied
```

H. Piping redirects stdout of process to stdin of next process.

1. Example of piping:

```
% cat grades.txt
ID      Final   Total   Pct     Grade
Maximum 320     979     100
578945  231     852     87      B
514100  281     848     87      B
958134  285     942     96      A
71570   0       131     13      F
extra line to show sorting
% cat grades.txt | sort
514100  281     848     87      B
578945  231     852     87      B
71570   0       131     13      F
958134  285     942     96      A
ID      Final   Total   Pct     Grade
Maximum 320     979     100
extra line to show sorting
% cat grades.txt | sort | wc
7 34 152
```

2. Example of piping and quoting:

```
% find . -name e* -print
find: paths must precede expression
% find . -name "e*" -print
./News/ecs40
./public_html/ecs40
./public_html/ecs50
./public_html/ecs50/echo.csp
./.dt/errorlog.older
./ecs50
./ecs50/cusp/explode.exe
./ecs10
./ecs40
./CUSP/explode.exe
./ecs40.tar
./ecs30
% find . -name "e*" -print | sort
./.dt/errorlog.older
./CUSP/explode.exe
./News/ecs40
./ecs10
```

./ecs30
./ecs40
./ecs40.tar
./ecs50
./ecs50/cusp/explode.exe
./public_html/ecs40
./public_html/ecs50
./public_html/ecs50/echo.csp
%

VI. Interactive C Shell
- A. Startup
  1. Startup does /etc/csh.cshrc (not in CSIF) and /etc/csh.login then in user's home it does .cshrc and then .login.
  2. Each new shell after login just runs .cshrc
- B. Aliases
  1. Create an alias**:** **alias** *word string.*
  2. Shows the associated alias: **alias** *word.*
  3. Show all aliases: **alias**
  4. Remove all aliases that match a pattern: **unalias** *pattern*
- C. History
  1. List past command lines: **history**
  2. !! = last command
  3. !*number* = replaced with specified event number
  4. !*prefix* = replaced with last command that started with prefix

VII. All Shell Scripts
- A. rc (run command) script, ~/.bashrc, is executed ever time an interactive sub shell is created.
- B. Interpreter line is first line, begins with **#!**, and indicates the program to be used to run the script. If omitted then, the login shell will spawn a subshell to run the script.
- C. After writing in a text editor, you must explicitly set the executable bit(s) using chmod.
- D. **source** *filename* = has the current shell execute the script, e.g., **source .cshrc**

VIII. BASH Shell Scripts
- A. **read** can be used to read one or more variables interactively from the user.
- B. Parameters: **$0, $1 …. $*, or "$@"** is the complete set of command line arguments. **$#** is the number of elements in the argument list.
  1. **shift** = causes all of the positional parameters $1..$n to be renamed $2..$(n-1) and $1 to be lost. Note that $0 remains unchanged.

2. Parameter and shift example script:

```
[davis@lect15 notes]$ cat parameters.sh
#! /bin/bash

echo Number of arguments: $#argv
echo 'Zeroth argument, $0: ' $0
echo 'First argument, $1: ' $1
echo 'Second argument, $2: ' $2
echo 'All arguments, $*: ' $* ' $@:' $@
shift
echo 'New arguments: $0 = ' $0 ' $1 = ' $1
echo 'New all args = ' $@

[davis@lect15 notes]$ parameters.sh A B C D
Number of arguments: 4argv
Zeroth argument, $0:  parameters.sh
First argument, $1:  A
Second argument, $2:  B
All arguments, $*:  A B C D  $@: A B C D
New arguments: $0 =  parameters.sh  $1 =  B
New all args =  B C D
[davis@lect15 notes]$
```

C. Computation and String Handling:
   1. Assignment is simple, e.g., **x =5; y = 10**
   2. Use **(( ))** construct for numeric computation, e.g. **((z = x + y))**
   3. Use **${** *variable operator pattern*} construct for string manipulation.
   4. Given `bash-3.00$ filename=/home/davis/archive/hope.tar.gz`
      a. operators # = delete shortest segment that matches *pattern* at beginning of *variable*.
         `bash-3.00$ echo ${filename#*ho}`
         `me/davis/archive/hope.tar.gz`
      b. operator ## = delete longest segment that matches *pattern* at beginning of *variable*.
         `bash-3.00$ echo ${filename##*ho}`
         `pe.tar.gz`
      c. operator % = delete shortest segment that matches *pattern* at end of *variable.*
         `bash-3.00$ echo ${filename%ho*}`
         `/home/davis/archive/`
      d. operator %% = delete longest segment that matches *pattern* at end of *variable*.
         `bash-3.00$ echo ${filename%%ho*}`
         `/`

D. Boolean Expressions:
   1. **test** *expr1* operator *expr2*, or [*expr1* operator *expr2*], or [[*expr1* operator *expr2*]]
      a. numerical comparison operators are: -eq, -ne, -gt, -ge, -lt, and –le (less than or equal)
      b. Binary string comparison operators are: =, !=, and ==
      c. the [[ ]] construct allows wild cards
   2. **test** option *expr*, or [option *expr*]
      a. String options: -n (not null), -z (is null), and just the string itself tests if it is assigned and not null.

b. File attribute testing options: -f (regular), -r (readable), -w (writable), -x (executable), -d (directory), -s (exists and size > 0), and -e (exists), -z (exists but has a size of zero).

3. Logical Operators: && and ||

E. Control Structures

1. **if** *expr1*
   then # expr evaluates to true
   list1
   elif *expr2*
   then
   list2
   else
   list3
   fi

2. **case** *expr* in
   pattern1) *list* ***;;***
   pattern2 | pattern3) *list2* ***;;***
   ****)*** *defaultList*
   esac

3. **for** *variable_name* **in** *(Word_list)* ***; do***
   commandlist
   done

4. **while**(*expr)* ***; do***
   commandlist
   done

5. **break** and **continue** work within both for and while loops.

IX.Example scripts

```
[davis@lect15 notes]$ cat case.sh
#! /bin/bash

while [ $# -gt 0 ] ; do
  case $1 in
    *.tar.gz) echo $1 is gzipped tar ;;
    *.Z) echo $1 is compressed ;;
    *.cpp | *.c ) echo $1 is C based ;;
    parameters.sh) echo $1 is parameters.sh ;;
    *.tar) echo $1 is just tarred ;;
    *.tar.Z) echo $1 compressed tar ;;
    *) echo echo $1 is neither compressed nor tarred
  esac

  shift
done # while
```

```
[davis@lect15 notes]$ case.sh f.Z f.tar.Z f.tar f.tar.gz
f.Z is compressed
f.tar.Z is compressed
f.tar is just tarred
f.tar.gz is gzipped tar
[davis@lect15 notes]$ case.sh f.zip parameters.sh f.cpp
echo f.zip is neither compressed nor tarred
parameters.sh is parameters.sh
f.cpp is C based
[davis@lect15 notes]$

[davis@lect15 notes]$ cat compile.sh
#! /bin/bash

for arg in $@ ; do
  rm a.out 2> /dev/null

  if [ ! -r $arg ]
  then
    echo $arg is not readable
  else
    if [[ $arg == *.c ]]
    then
      if gcc $arg
      then
        a.out
      else
          echo $arg did not compile
      fi # if compiling worked
    elif [[ $arg == *.cpp ]]
    then
      g++ $arg
      if test -e a.out
        then
          a.out
      fi # if a.out exists
    else
      echo $arg is not a C or C++ file
    fi # if arg is .c
  fi # if readable
done
[davis@lect15 notes]$ compile.sh hello.c hello.cpp whoops.c bad.c hello.p
Hello, C world!
Hello, C++ world!
whoops.c is not readable
bad.c: In function 'main':
bad.c:5: error: syntax error before string constant
bad.c did not compile
hello.p is not a C or C++ file
[davis@lect15 notes]$
```

X. **sed** [*sed_command*] [-e *script*] [-f *scriptfile*] { *fileName* } *

A. Edits an input stream according to a script that contains editing commands.
B. Surround scripts on command line with single quotes.
C. Commands
   1. Address is either line number, or a regular expression / /. $ selects last line.
   2. Address range can be a single address, or a couple of addresses separated by commas. If two addresses then the action is applied to all lines between the addresses, inclusive.
   3. If no address, then applied to all lines.
   4. Append text after line(s). Must be done using a script file.

      sed *address* **a**
      text

```
[davis@lect15 notes]$ cat sedFile.txt
Line One
Line Two
Line Three
[davis@lect15 notes]$ cat sedAfile
2,3 a\
Stuff* \
Junk
[davis@lect15 notes]$ sed -f sedAfile sedFile.txt
Line One
Line Two
Stuff*
Junk
Line Three
Stuff*
Junk
[davis@lect15 notes
```

   5. Replace text with change command. Must be done using a script file.

      sed *addressRange* **c**
      text

```
[davis@lect15 notes]$ cat sedCfile
2,3 c\
Stuff* \
Junk
[davis@lect15 notes]$ sed -f sedCfile sedFile.txt
Line One
Stuff*
Junk
[davis@lect15 notes]$
```

   6. Insert text before line(s). Must be done using a script file.

      sed *address* **i**
      text

```
[davis@lect15 notes]$ cat sedIfile
2,3 i\
Stuff* \
Junk
```

```
[davis@lect15 notes]$ sed -f sedIfile sedFile.txt
Line One
Stuff*
Junk
Line Two
Stuff*
Junk
Line Three
[davis@lect15 notes]$
```

7. Delete text: **sed** *addressRange* **d**

```
a. sed '2,3 d' test1.hw
b. sed '/include/ d' test2.txt
c. sed '/if/,/endif/ d' testhw
```

8. Append the contents of the file name after the line: **sed** *address* **r** *filename*
   a. sed '4r test2.txt' testhw
9. Substitute the first occurrence of the regular express with str: **sed** *addressRange* **s**/*expr*/*str*/

```
hp9% ps
   PID TTY       TIME COMMAND
  1152 ttyp2     0:00 telnetd
  1187 ttyp2     0:00 ps
  1153 ttyp2     0:00 tcsh
hp9% ps|sed s/ttyp/whoops/
   PID TTY       TIME COMMAND
  1152 whoops2     0:00 telnetd
  1188 whoops2     0:00 ps
  1153 whoops2     0:00 tcsh
  1189 whoops2     0:00 sed
hp9% echo $path
/home/davis/bin /pkg/bin /bin /usr/bin /usr/sbin /usr/dt/bin /usr/bin/X11 . /alt
pkg/maxplus2/bin
hp9% echo $path | sed s/bin/binary/
/home/davis/binary /pkg/bin /bin /usr/bin /usr/sbin /usr/dt/bin /usr/bin/X11 . /
altpkg/maxplus2/bin
```

10. Substitute the every occurrence of the regular express with str: **sed** *addressRange* **s**/*expr*/*str*/**g**

```
hp9% echo $path | sed s/bin/binary/g
/home/davis/binary /pkg/binary /binary /usr/binary /usr/sbinary /usr/dt/binary /
usr/binary/X11 . /altpkg/maxplus2/binary
hp9%
```

XI. awk -Fc [-f awkfileName] program {fileName}*, where c is field separator if -F option

A. If the program is on the command line then surround it with single quotes.
B. Processes its input one line at a time applying user-specified awk pattern commands to each line with the format [condition] [{action}]
   1. If no condition then action is performed on all lines.
   2. If no action then the line is printed to standard out.
C. Built-in variables: NF = number of fields in a line; NR = contains the line number of the current line; $1 is first field, $2 is second field .. $NF is the last field.
D. Conditions
   1. BEGIN = triggered before first line ; END = triggered after last line
   2. Condition ranges: <first expression>, <second expression> then awk performs the action starting at the first line that satisfys the first expression, and stops when it reaches a line that satisifys the second expression.
   3. Patterns

a. /^well/ = starting with well
b. /dumb$/ = ending with dumb
c. /if*then/ = line with "if" followed by a "then" somewhere

E. Actions
1. if , if else, while, for, break, continue
2. next = skip remaining commands in program, start next cycle
3. exit = exit awk
4. var = expression.
5. print [list of expressions], eg. print $1 $3 $NF
6. printf format [, list of expressions], e.g. {printf "line# = %d, words = %d\n", NR, NF}

```
hp9% echo $path
/home/davis/bin /pkg/bin /bin /usr/bin /usr/sbin /usr/dt/bin /usr/bin/X11 . /alt
pkg/maxplus2/bin
hp9% ps | awk /ps/
  1194 ttyp2     0:00 ps
hp9% ps | awk ' BEGIN {print "first"} /5/ {print $2 $NF}'
first
ttyp2telnetd
ttyp2tcsh
hp9% ps | awk 'END {print "Done"} /5/ {print $1, $3, NF}'
1152 0:00 4
1153 0:00 4
Done
hp9% ps
   PID TTY       TIME COMMAND
  1152 ttyp2     0:00 telnetd
  1219 ttyp2     0:00 ps
  1153 ttyp2     0:00 tcsh
hp9% ps | awk '/5/ {printf"PID = %d line# = %d, shown# = %d\n", $1, NR, ++count}'
PID = 1152 line# = 2, shown# = 1
PID = 1153 line# = 4, shown# = 2
hp9%
hp9% ps | awk '/TIME/ , /52/ {for (i = 0; i < NR; i++) print i, $1}'
0 PID
0 1152
1 1152
hp9% ps | awk '/TIME/ , /52/ {for (i = 0; i < NR; i++) print i, $1}' | wc
3 6 20
hp9%
```

XII. Processes

A. Assigned unique Process IDs (PID) sequentially by OS.
B. Created by OS, and other applications to do specific tasks.
C. **ps** -eulf gets snapshot of current processes. e= every process; f = full format; l = long format, u = specify user.
1. PROCESS STATE CODES
a. D uninterruptible sleep (usually IO)
b. R runnable (on run queue)
c. S sleeping
d. T traced or stopped
e. Z a defunct ("zombie") process
f. W has no resident pages
g. < high-priority process
h. N low-priority task
i. L has pages locked into memory (for real-time and custom IO)

D. **top** provides ongoing total system process information and ps sorted by CPU Time

E. **ps Example**

```
[davis@lect15 ~] [davis@lect15 ~]$ find / -name hello >& /dev/null

Suspended
[davis@lect15 ~]$ ps
  PID TTY          TIME CMD
17613 pts/0    00:00:00 tcsh
17660 pts/0    00:00:00 find
17696 pts/0    00:00:00 ps
```

F. ps Long Format Examples

1. F = Process flags 100 = used super-user privileges 040 = forked but didn't exec
2. UID = Effective User ID
3. PPID = Parent Process ID
4. PRI = Priority. The smaller the value, the higher the priority.
5. NI = Nice
6. SZ = Size. Sizeof the process' data and stack in kilobytes.

```
[davis@lect15 ~]$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
davis    17613 17612  0 13:12 pts/0    00:00:00 -tcsh
davis    17660 17613  0 13:24 pts/0    00:00:00 find / -name hello
davis    17697 17613  0 13:40 pts/0    00:00:00 ps –f
[davis@lect15 ~]$ ps -l
  F S   UID   PID  PPID  C PRI  NI ADDR    SZ  WCHAN TTY          TIME CMD
100 S   696 17613 17612  0  72   0    -   722 10817b pts/0    00:00:00 tcsh
000 T   696 17660 17613  0  69   0    -   428 108e50 pts/0    00:00:00 find
000 R   696 17700 17613  0  74   0    -   635      - pts/0    00:00:00 ps
```

G. ps Showing Every Process

```
[davis@lect15 ~]$ ps –ef  //  shortened by Sean
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0  2002 ?        00:00:04 init [5]
root         2     1  0  2002 ?        00:00:00 [keventd]
root         3     1  0  2002 ?        00:00:00 [kapm-idled]
root         4     1  0  2002 ?        00:00:02 [kswapd]
root       393     1  0  2002 ?        00:00:00 syslogd -m 0
root       398     1  0  2002 ?        00:00:00 klogd -2
rpc        412     1  0  2002 ?        00:00:03 portmap
rpcuser    427     1  0  2002 ?        00:00:00 rpc.statd
root       660     1  0  2002 ?        00:00:07 /usr/sbin/sshd
lp         702     1  0  2002 ?        00:00:00 lpd Waiting
nobody     717     1  0  2002 ?        00:00:00 rpc.rusersd
nobody     729     1  0  2002 ?        00:00:00 rpc.rwalld
xfs        804     1  0  2002 ?        00:00:17 xfs -droppriv -daemon
root       829     1  0  2002 tty1     00:00:00 /sbin/mingetty tty1
root       830     1  0  2002 tty2     00:00:00 /sbin/mingetty tty2
root       835     1  0  2002 ?        00:00:00 /usr/bin/kdm -nodaemon
root     17612   660  0 13:12 ?        00:00:00 /usr/sbin/sshd
davis    17613 17612  0 13:12 pts/0    00:00:00 -tcsh
davis    17660 17613  0 13:24 pts/0    00:00:00 find / -name hello
davis    17698 17613  0 13:40 pts/0    00:00:00 ps –ef
```

H. ps –u Example

```
[davis@lect15 ~]$ ps -u nobody
  PID TTY          TIME CMD
  717 ?        00:00:00 rpc.rusersd
```

```
        729 ?        00:00:00 rpc.rwalld
```

I. top Example

```
[davis@lect15 ~]$ top  // shortened by Sean

  1:50pm  up 10 days, 15 min,  4 users,  load average: 0.00, 0.00, 0.00
57 processes: 55 sleeping, 1 running, 0 zombie, 1 stopped
CPU states:  0.1% user,  0.1% system,  0.0% nice,  0.4% idle
Mem:   126640K av,   99544K used,   27096K free,       0K shrd,    5356K buff
Swap:  530104K av,      40K used,  530064K free                   62936K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
17704 davis     16   0   968  968   772 R     2.8  0.7   0:00 top
    1 root       8   0   544  544   476 S     0.0  0.4   0:04 init
    2 root       9   0     0    0     0 SW    0.0  0.0   0:00 keventd
    3 root       9   0     0    0     0 SW    0.0  0.0   0:00 kapm-idled
    4 root       9   0     0    0     0 SW    0.0  0.0   0:02 kswapd
    5 root       9   0     0    0     0 SW    0.0  0.0   0:00 kreclaimd
    6 root       9   0     0    0     0 SW    0.0  0.0   0:00 bdflush
    7 root       9   0     0    0     0 SW    0.0  0.0   0:00 kupdated
    8 root      -1 -20     0    0     0 SW<   0.0  0.0   0:00 mdrecoveryd
  393 root       9   0   588  588   484 S     0.0  0.4   0:00 syslogd
  398 root       9   0  1080 1080   456 S     0.0  0.8   0:00 klogd
```

XIII. Signals

A. Programs must deal with events that must interrupt the regular flow of a program.

Macro	#	Default	Description
SIGINT	2	quit	interrupt (Ctrl-C)
SIGTRAP	5	dump	trace trap (used by debuggers)
SIGFPE	8	dump	arithmetic exception
SIGKILL	9	quit	kill (cannot be caught, blocked or ignored.)
SIGALRM	14	quit	alarm clock
SIGTERM	15	quit	termination (default for kill utility)
SIGUSR1	16	quit	user signal 1
SIGCHLD	18	ignore	child status changed.
SIGSTP	24	suspend	suspend. (Ctrl–Z)
SIGCONT	25	ignore	resume. (fg)

B. **trap** '*command_list' signal_list* has the BASH script execute the *command_list* when one of the signals is caught.
   1. An empty *command_list* has the script ignore the signal listed in the *signal_list*.
   2. Signals can be listed by number, e.g. 1 2 15, or name, e.g., HUP INT TERM

XIV. Process control

A. Ctrl-Z to suspend the process in the foreground.
B. **bg** to place suspended process in background.
C. **&** at end of command to place process immediately in background.
D. **jobs** displays a list of the shell's jobs.
E. **fg** [%job] resumes the specified job as the foreground process. If no job# then last job referenced.
F. **sleep** *seconds* Suspends a process for *seconds* seconds.
G. **wait** [pid] Suspends a process until one of its children (or a specific child if pid is provided) terminates.
H. **kill** [-*SignalID*] {PID or %job}+ to send a signal to a process. Default is SIGTERM which will terminate it.

I. Examples

1. Use of Ctrl-Z, bg, and fg.

```
% find / -name ecs >&! findfile
^Z
Suspended
% bg
[1]    find / -name ecs >& findfile &
% ps
   PID TTY       TIME COMMAND
  1291 ttyp2     0:00 telnetd
  1292 ttyp2     0:02 tcsh
  2371 ttyp2     0:00 ps
  2370 ttyp2     0:00 find
% fg
find / -name ecs >& findfile
^C
```

2. Use of &, jobs, and kill

```
[davis@lect15 ~]$ find / -name ecs > & ! findfile &
[1] 18308
[davis@lect15 ~]$ jobs
[1]  + Running                 find / -name ecs >& findfile
[davis@lect15 ~]$ fg
find / -name ecs >& findfile

Suspended
[davis@lect15 ~]$ find / -name ecs > & ! findfile &
[2] 18309
[davis@lect15 ~]$ jobs
[1]  + Suspended               find / -name ecs >& findfile
[2]  - Running                 find / -name ecs >& findfile
[davis@lect15 ~]$ ps
  PID TTY          TIME CMD
17613 pts/0    00:00:00 tcsh
18308 pts/0    00:00:00 find
18309 pts/0    00:00:00 find
18310 pts/0    00:00:00 ps
[davis@lect15 ~]$ kill %1 18309
[davis@lect15 ~]$ jobs
[1]  + Terminated              find / -name ecs >& findfile
[2]  + Terminated              find / -name ecs >& findfile
[davis@lect15 ~]$
```