CS213: Software Systems Laboratory

Autumn 2023-24

Koteswararao Kondepu

k.kondepu@iitdh.ac.in

Recap

- Definition of Unix
- Hierarchy - Shell, file system, permissions
- Types of Shell
- Tools. grep, find, head, tail, sort

r w x
4 2 1

1977 – 1983

Bash

ls -l

d

Outline

- Exercise on Bash commands
- Special Variables
- Process management
- Directory and File operations
- Loops

Exercise: Write a script to extract the usernames from the file /etc/group

cut command is used for text processing to extract a portion of text from a file by selecting columns.

```
NAME
       group - user group file

DESCRIPTION
       The  /etc/group  file  is  a text file that defines the groups on the system.  There is one
       entry per line, with the following format:

           group name:password:GID:user list
```

Usage: **cut <options> <filename>**

Common Options:

- **-c list** : Select only these characters
- **-b list** : Select only these byes
- **-f list** : Select only these fields.
- **-d delim** Use delim as the field delimiter character instead of the tab character
- **-s** : Do not print lines not containing delimiters

cut -f 4 /etc/group -d':'

```
#!/bin/bash
cut -d':' -f 4 /etc/group
cut -d':' -f 4 /etc/group | grep $USER
```

Exercise: grep

- Display all lines containing the term 'lib'?
- Showcase all lines that do not include the term 'lib'?
- Counting the total occurrences of lines containing 'lib'?
- Display all lines which contain 'lib' along with the line numbers?
- Display lines which do not contain 'lib', but only top 2 lines?

mylib.txt

How are lib

What are you doing Lib

How are lib2

Usage: grep [OPTION...] PATTERNS [FILE...]

Common Options:

-v : Select non-matching lines from FILE

-n : Select matching lines from FILE

-c : Print only a count of selected liner per FILE

-i -n -R

Lib

lib

```
#!/bin/bash
grep 'lib' mylib.txt
grep -v 'lib' mylib.txt
grep -c 'lib' mylib.txt
grep -n 'Lib' mylib.txt
grep -v 'lib' mylib.txt | head -n 2
```

Bash: Special Variables

Variable	Variable Description
$0	The filename of the current script.
$n	These variables correspond to the arguments with which a script was invoked where n is from 1,2....
$#	The number of arguments supplied to a script.
$*	All the arguments are double quoted. If a script receives two arguments, $* is equivalent to $1 $2
$@	All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2.
$?	The exit status of the last command executed
$$	The process number of the current shell. For shell scripts, this is the process ID under which they are executing.
$!	The process number of the last background command.

fg bg

Process Management

Terminates a process by sending a signal to it.

```
NAME
       pgrep, pkill - look up or signal processes based on name and other attributes

SYNOPSIS
       pgrep [options] pattern
       pkill [options] pattern

DESCRIPTION
       pgrep looks through the currently running processes and lists the process IDs which match the selection criteria to stdout.  All the criteria have to match.
       For example,

              $ pgrep -u root sshd

       will only list the processes called sshd AND owned by root.  On the other hand,

              $ pgrep -u root,daemon

       will list the processes owned by root OR daemon.

       pkill will send the specified signal (by default SIGTERM) to each process instead of listing them on stdout
```

Usage: **pkill <options> <pattern>**

Common Options:

-1 : To reload a process

-9 : To kill a process..

-15 : To gracefully stop a process..

kill

pkill -1

pkill 9

usa : ps -ef

ps -p

ps -aux | grep hello.sh

Try "kill" command for the similar kind of operations

-s

```
#!/bin/bash
Pkill -9 213
```

Date

Displays the current time according to the given format

```
NAME
       date - print or set the system date and time

SYNOPSIS
       date [OPTION]... [+FORMAT]
       date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]

DESCRIPTION
       Display the current time in the given FORMAT, or set the system date.

       Mandatory arguments to long options are mandatory for short options too.

       -d, --date=STRING
              display time described by STRING, not 'now'

       --debug
              annotate the parsed date, and warn about questionable usage to stderr

       -f, --file=DATEFILE
              like --date; once for each line of DATEFILE
```

Usage: **date <options> <format>**

Common Options:

-d : Used to display time described by STRING

-s : To set time described by STRING

-u : To display or set the UTC.

```
#!/bin/bash
sudo date -us "19 AUG 2023 09:20:34"
```

Cat

Reads data from the file and gives its content as output.

```
CAT(1)

NAME
       cat - concatenate files and print on the standard output

SYNOPSIS
       cat [OPTION]... [FILE]...

DESCRIPTION
       Concatenate FILE(s) to standard output.

       With no FILE, or when FILE is -, read standard input.

       -A, --show-all
              equivalent to -vET

       -b, --number-nonblank
              number nonempty output lines, overrides -n
```

print

Usage: **cat <options> <file>**

Common Options:

file1 file2 : Concatenate both the files and provides output.

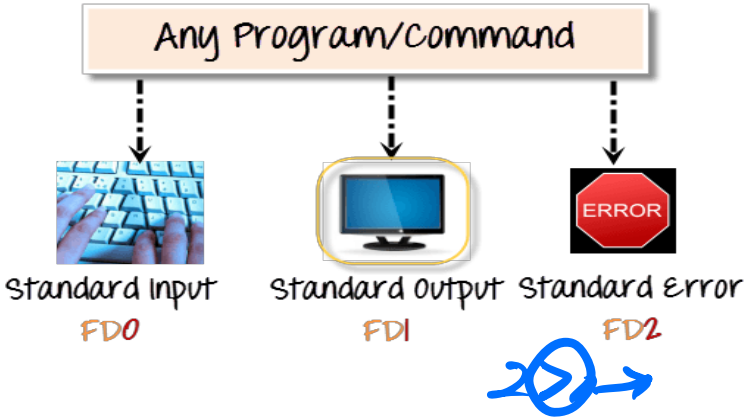
-e filename : To display $ character at the end of each line.

file1 > file2 : To copy content from file1 to file2.

```
#!/bin/bash file1
sudo cat >> cs213.txt
```

I/O Redirection

- *Redirection can be defined as changing the way from where commands read input to where commands sends output.*



File Descriptors on Linux

Name	Represents	Examples
File Descriptor 0 (fd[0])	Standard input	Keyboard, file, terminal
File Descriptor 1 (fd[1])	Standard output	Screen, database
File Descriptor 2 (fd[2])	Standard error	File, terminal

Symbol	Description
>	Directs the standard output of a command to a file. If the file exist, it is overwritten.
>>	Directs the output to a file, adding the output to the end of the existing file.
2>	Directs standard error to the file.
2>>	Directs standard error to a file, adding the output to the end of the existing file.
&>	Directs standard output and standard error to the file.
<	Directs the contents of a file to the command.
<<	Accepts text on the following lines as standard input.
<>	The specified file is used for both standard input and standard output.

File operations: touch

Create, change and modify timestamps of a file

```
NAME
       touch - change file timestamps

SYNOPSIS
       touch [OPTION]... FILE...

DESCRIPTION
       Update the access and modification times of each FILE to the current time

       A FILE argument that does not exist is created empty, unless -c or -h is supplied.

       A FILE argument string of - is handled specially and causes touch to change the times

       Mandatory arguments to long options are mandatory for short options too.

       -a     change only the access time

       -c, --no-create
              do not create any files

       -d, --date=STRING
              parse STRING and use it instead of current time

       -f     (ignored)
```

Usage: touch **<options> <file>**

Common Options:

-a : used to change access time.

-c : used to check whether a file is created or not.

-d : used to change modification date.

```
#!/bin/bash
touch filename.txt
```

File operations : mv

Rename a file or directory or move a file from one location to another location

```
NAME
       mv - move (rename) files

SYNOPSIS
       mv [OPTION]... [-T] SOURCE DEST
       mv [OPTION]... SOURCE... DIRECTORY
       mv [OPTION]... -t DIRECTORY SOURCE...

DESCRIPTION
       Rename SOURCE to DEST, or move SOURCE(s) to DIRECTORY.

       Mandatory arguments to long options are mandatory for short options too.

       --backup[=CONTROL]
              make a backup of each existing destination file

       -b     like --backup but does not accept an argument

       -f, --force
              do not prompt before overwriting

       -i, --interactive
              prompt before overwrite

       -n, --no-clobber
              do not overwrite an existing file

       If you specify more than one of -i, -f, -n, only the final one takes effect.

       --strip-trailing-slashes
              remove any trailing slashes from each SOURCE argument

       -S, --suffix=SUFFIX
              override the usual backup suffix
```

Usage: mv [options(s)] [source_file_name(s)] [Destination_file_name]

mv [source_file_name(s)] [Destination_path]

Common Options:

```
#!/bin/bash
mv file1.txt file2.txt
```

-i: user interactive.

-f: overrides this minor protection and overwrites the destination file forcefully and deletes the source file.

File operations : cp

Used to copy files or groups of files or directories

```
NAME
       cp - copy files and directories

SYNOPSIS
       cp [OPTION]... [-T] SOURCE DEST
       cp [OPTION]... SOURCE... DIRECTORY
       cp [OPTION]... -t DIRECTORY SOURCE...

DESCRIPTION
       Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.

       Mandatory arguments to long options are mandatory for short options too.

       -a, --archive
              same as -dR --preserve=all

       --attributes-only
              don't copy the file data, just the attributes

       --backup[=CONTROL]
              make a backup of each existing destination file

       -b     like --backup but does not accept an argument

       --copy-contents
              copy contents of special files when recursive

       -d     same as --no-dereference --preserve=links

       -f, --force
              if an existing destination file cannot be opened, remove it and try again (this option is ignored when the -n option is also used)

       -i, --interactive
              prompt before overwrite (overrides a previous -n option)
```

Common Options:

Usage: **cp [OPTION] Source Destination**

cp [OPTION] Source Directory

-i: Interactive copying.

-f: destination file is deleted first forcefully and then copying of content is done from source to destination file.

```
#!/bin/bash
cp file1.txt file2.txt
```

File operations : rm

Used to remove files or directories

```
NAME
       rm - remove files or directories

SYNOPSIS
       rm [OPTION]... [FILE]...

DESCRIPTION
       This manual page documents the GNU version of rm.  rm removes each specified file.  By default, it does not remove directories.

       If  the  -I or --interactive=once option is given, and there are more than three files or the -r, -R, or --recursive are given, then rm prompts the user for
       whether to proceed with the entire operation.  If the response is not affirmative, the entire command is aborted.

       Otherwise, if a file is unwritable, standard input is a terminal, and the -f or --force option is not given, or the -i  or  --interactive=always  option  is
       given, rm prompts the user for whether to remove the file.  If the response is not affirmative, the file is skipped.

OPTIONS
       Remove (unlink) the FILE(s).

       -f, --force
              ignore nonexistent files and arguments, never prompt

       -i     prompt before every removal

       -I     prompt  once  before removing more than three files, or when removing recursively; less intrusive than -i, while still giving protection against most
              mistakes

       --interactive[=WHEN]
              prompt according to WHEN: never, once (-I), or always (-i); without WHEN, prompt always

       --one-file-system
              when removing a hierarchy recursively, skip any directory that is on a file system different from that of the corresponding command line argument

       --no-preserve-root
              do not treat '/' specially

       --preserve-root
              do not remove '/' (default)

       -r, -R, --recursive
              remove directories and their contents recursively
```

Usage: rm **<options> <file>**

Common Options:

-f: force deletion.

-i: interactive deletion.

-r: recursive Deletion to remove directories.

```
#!/bin/bash
rm filename.txt
```

Directory operations: mkdir

Create or make new directories

```
NAME
       mkdir - make directories

SYNOPSIS
       mkdir [OPTION]... DIRECTORY...

DESCRIPTION
       Create the DIRECTORY(ies), if they do not already exist.

       Mandatory arguments to long options are mandatory for short options too.

       -m, --mode=MODE
              set file mode (as in chmod), not a=rwx - umask

       -p, --parents
              no error if existing, make parent directories as needed

       -v, --verbose
              print a message for each created directory

       -Z     set SELinux security context of each created directory to the default type

       --context[=CTX]
              like -Z, or if CTX is specified then set the SELinux or SMACK security context to CTX

       --help display this help and exit

       --version
              output version information and exit
```

mkdir SS1213

Usage: mkdir **<options> <directory>**

Common Options:

-p: enables the command to create parent directories as necessary.

-m: set the file modes, i.e., permissions, etc. for the created directories.

-v: It displays a message for every directory created.

#!/bin/bash
mkdir exercises

Directory operations : pwd

prints the path of the working directory

```
NAME
       pwd - print name of current/working directory

SYNOPSIS
       pwd [OPTION]...

DESCRIPTION
       Print the full filename of the current working directory.

       -L, --logical
              use PWD from environment, even if it contains symlinks

       -P, --physical
              avoid all symlinks

       --help display this help and exit

       --version
              output version information and exit

       If no option is specified, -P is assumed.

       NOTE:  your  shell  may  have  its  own version of pwd, which usually supersedes the version described here.  Please refer to your shell's documentation for
       details about the options it supports.
```

Usage: pwd

Common Options:

-L : prints the symbolic path.

-p: prints the actual path.

```
#!/bin/bash
pwd
```

Directory operations : cd

used to change the current directory of the terminal

```
cd: cd [-L|[-P [-e]] [-@]] [dir]
    Change the shell working directory.

    Change the current directory to DIR.  The default DIR is the value of the
    HOME shell variable.

    The variable CDPATH defines the search path for the directory containing
    DIR.  Alternative directory names in CDPATH are separated by a colon (:).
    A null directory name is the same as the current directory.  If DIR begins
    with a slash (/), then CDPATH is not used.

    If the directory is not found, and the shell option `cdable_vars' is set,
    the word is assumed to be  a variable name.  If that variable has a value,
    its value is used for DIR.

    Options:
      -L         force symbolic links to be followed: resolve symbolic
                 links in DIR after processing instances of `..'
      -P         use the physical directory structure without following
                 symbolic links: resolve symbolic links in DIR before
                 processing instances of `..'
      -e         if the -P option is supplied, and the current working
                 directory cannot be determined successfully, exit with
                 a non-zero status
      -@         on systems that support it, present a file with extended
                 attributes as a directory containing the file attributes

    The default is to follow symbolic links, as if `-L' were specified.
    `..' is processed by removing the immediately previous pathname component
    back to a slash or the beginning of DIR.

    Exit Status:
    Returns 0 if the directory is changed, and if $PWD is set successfully when
    -P is used; non-zero otherwise.
```

Usage: cd **<options> <file>**

cd /

cd ../../../

```
#!/bin/bash
cd Desktop
```

Read

Used to take the user's input from the terminal

```
read: read [-ers] [-a array] [-d delim] [-i text] [-n nchars] [-N nchars] [-p prompt] [-t timeout] [-u fd] [name ...]
    Read a line from the standard input and split it into fields.

    Reads a single line from the standard input, or from file descriptor FD
    if the -u option is supplied.  The line is split into fields as with word
    splitting, and the first word is assigned to the first NAME, the second
    word to the second NAME, and so on, with any leftover words assigned to
    the last NAME.  Only the characters found in $IFS are recognized as word
    delimiters.

    If no NAMEs are supplied, the line read is stored in the REPLY variable.

    Options:
      -a array  assign the words read to sequential indices of the array
                variable ARRAY, starting at zero
      -d delim  continue until the first character of DELIM is read, rather
                than newline
      -e        use Readline to obtain the line in an interactive shell
      -i text   use TEXT as the initial text for Readline
      -n nchars return after reading NCHARS characters rather than waiting
                for a newline, but honor a delimiter if fewer than
                NCHARS characters are read before the delimiter
      -N nchars return only after reading exactly NCHARS characters, unless
                EOF is encountered or read times out, ignoring any
                delimiter
      -p prompt output the string PROMPT without a trailing newline before
                attempting to read
      -r        do not allow backslashes to escape any characters
      -s        do not echo input coming from a terminal
```

Usage: read **[options] [var1, var2, var3…]**

Common Options:

-p : Outputs the prompt string before reading user input.

-a : Assigns the provided word sequence to a variable named <array>

.sh [SSL]

```
#!/bin/bash
#Take the input with the prompt message
read -p "Enter the book name: " book
#Print the input value
echo "Book name $book"
```

Exercise: List the active processes and redirect to a file

Usage: **ps** **<options>**
Common Options:

-e : Select all the processes
-A : Select all the processes
-p : Select by process ID
-u : Select by username or ID

ps -eup

ps -aux | grep "<→"

pkill -9 9696

```
core5g@core5g:~$ ps -efu
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
core5g   15350  0.0  0.0  22696  2184 pts/1    Ss   lug25   0:00 -bash LC_ADDRESS=it_IT.UTF-8 LC_NAME
core5g    3224  0.0  0.0  22828  3700 pts/0    Ss   lug22   0:00 -bash LC_ADDRESS=it_IT.UTF-8 LC_NAME
core5g    9696  0.0  0.0  39676  3572 pts/0    R+   19:02   0:00  \_ ps -efu LS_COLORS=rs=0:di=01;34:
```

```
#!/bin/bash
ps -ef > results.txt
```

Exercise:

Write a script that appends the contents of multiple files to a single file named "files.txt". The names of the files should be provided as command-line arguments!

```
#!/bin/bash
if [ $# -ne 0 ]; then
for fname in $@
do
if [ -f $fname ]; then
cat $fname >> files.txt
else
echo "$fname does not exist or it is not a regular file"
fi
done
else
echo "Please pass at least one file name"
fi
```

file1.txt >> files.txt
file2.txt >> files.txt
file3.txt >> files.txt

Decision Making: If

```
if [<CONDITION1>]
    then
      <STATEMENT>
fi
```

if_condition.sh

```
#!/bin/bash

a=50
b=10

if [ $a -gt $b ]
  then
      echo "a is greater than b"
fi
```

a=$1
b=$2

- **-eq**: equal
- **-ne:** not equal
- **-gt**: greater than
- **-lt**: less than
- **-ge:** greater than or equal
- **-le:** less than or equal

Output:

```
$ chmod +x if_condition.sh
$ ./if_condition.sh

a is greater than b
```

Decision Making : If .. else

```
if [ <CONDITION> ]
        then
            <STATEMENT1>
        else
            <STATEMENT2>
        fi
```

* [[ ]]

bash (( ))

csh

if_elsecondition.sh

```
#! /bin/bash
a=110
b=48
if [ $a -lt $b ]
then
    echo "a is less than b"
else
    echo "a is greater than b"
fi
```

( ) ? : :

if ( )

[ ]

(( ))

Output:

```
$ chmod  +x if_elsecondition.sh
$ ./if_elsecondition.sh

a is greater than b
```

Decision Making: If ..elif.. else

```
if [ condition ]
    then
      statement 1
 elif [ condition ]
    then
      statement 2
    else
   do this by default
      fi
```

if_elif_elsecondition.sh

```
#! /bin/bash

a=10
b=10
c=20
if [ $a == $b -a $b == $c -a $a == $c ]
then
echo EQUILATERAL
elif [ $a == $b -o $b == $c -o $a == $c ]
then
echo ISOSCELES
else
echo SCALENE
fi
```

Output:

```
$ chmod +x if_elif_elsecondition.sh
$ ./if_elif_elsecondition.sh

ISOSCELES
```

Exercise: Special variables

Write a script that accepts two integer numbers as command line arguments and displays their sum. If the number of arguments is not exactly 2, the script should output a message: "*Please provide exactly 2 arguments*." Additionally, if the provided arguments are invalid integer numbers, the script should display the message: "Please provide valid integer numbers."

```
#!/bin/bash

  if [ $# -ne 0 ]; then
    echo “Please provide exactly 2 arguments”
  exit 1
  fi
  x=$1
  y=$2
  sum=‘expr $x + $y’
  if [ $? -ne 0 ]; then
    echo “Please provide valid integer numbers”
  exit 2
  else
  echo “The sum: $sum"
  fi
```

8.8 + 9.8

int

Logical AND

Usage: command1 && command2

logical_and.sh

```
#!/bin/bash
n=38
if [ $((n % 2)) == 0 ] && [ $((n % 5)) == 0 ]
then
   echo "Number is  divisible by  both 2 and 5"
else
   echo "Number is not divisible by both of them"
fi
```

Output:

```
$ chmod +x logical_and.sh
$ ./logical_and.sh

The number is not divisible by both of them
```

Logical OR

Usage: command1 || command2

logical_or.sh

```
#!/bin/bash
num=38
if [[ $num%2 == 0 ] || [ $num%5 == 0 ]]
then
  echo "Divisible by either 2 or 5"
else
  echo "Not divisible by either of them"
fi
```

Output:

```
$ chmod +x logical_or.sh
$ ./logical_or.sh

Divisible by either 2 or 5.
```

Logical NOT

Usage: !command1

logical_not.sh

```
#!/bin/bash

a=76
b=78

if [[ ! ${a} == ${b} ]]
then
   echo "both are not the same"
else
   echo "both are  the same"
fi
```

Output:

```
$ chmod +x logical_not.sh
$ ./logical_not.sh

both are not the same
```

CASE STATEMENT

case [ ] :

```
case EXPRESSION in
    Pattern_Case_1)
            STATEMENTS
        ;;
    Pattern_Case_1)
            STATEMENTS
        ;;
    Pattern_Case_N)
            STATEMENTS
        ;;
        *)
            STATEMENTS
        ;;
    esac
```

default :

*)

Output:

```
$ chmod +x case.sh
$ ./case.sh
Your DEPARTMENT is Computer Science
```

case.sh

```
#!/bin/bash
DEPARTMENT="Computer Science"
echo -n "Your DEPARTMENT is "
case $DEPARTMENT in

 "Computer Science")
        echo -n "Computer Science"
  ;;
 "Electrical and Electronics Engineering" | "Electrical
Engineering")
        echo -n "Electrical and Electronics Engineering or
Electrical Engineering"
   ;;

 "Information Technology" ")
        echo -n "Information Technology"
   ;;

  *)
        echo -n "Invalid"
   ;;
esac
```

While

```
while [ CONDITION ]
  do
  <STATEMENT>
  done
```

$1
$2
$3

while.sh

```
#!/bin/sh
a=0
while [ $a -lt 10 ]
do
  echo $a
  a=`expr $a + 1`
done
```

Output:

```
$ chmod +x while.sh
$ ./while.sh
0
1
2
3
4
5
6
7
8
9
```

FOR

```
for <var> in <value1 value2 ... valuen
do
<command 1>
<command 2>
done
```

for.sh

```
#!/bin/bash
for i in 1 2 3 4 5
do
  echo "Welcome $i times"
done
```

Output:

```
$ chmod +x for.sh
$ ./for.sh
Welcome 1 times
Welcome 2 times
Welcome 3 times
Welcome 4 times
Welcome 5 times
```

FOR Loop with Break and Continue Statement

for_break.sh

```
#!/bin/bash
for a in 1 2 3 4 5 6 7 8 9 10
do
  if [ $a == 6 ]
   then
     break
  fi
    echo "Iteration no $a"
done
```

Output:

```
$ chmod +x for_break.sh
$ ./for_break.sh

Iteration no 1
Iteration no 2
Iteration no 3
Iteration no 4
Iteration no 5
```

for_continue.sh

```
#!/bin/bash
for a in {1..10}
do
   if [ $((a % 2)) -eq 0 ]
   then
      continue
   fi
   echo "Iteration no $a"
done
```

Output:

```
$ chmod +x for_continue.sh
$ ./for_continue.sh

Iteration no 1
Iteration no 3
Iteration no 5
Iteration no 7
Iteration no 9
```

thank you!

email:
k.kondepu@iitdh.ac.in