* <filename.sh>

To create a file: touch script1.sh
nano script1.sh
vi script1.sh
vim script1.sh
emacs script1.sh

script1.sh

#!/bin/bash

#!/bin/sh

/sbin/bash

! → bang

CS213: Software Systems Laboratory

Autumn 2023-24

#!/bin/bash

cat /etc/group | sort

* shebang #!
* hashband
* sharp-exclamation
* pound-bang
* sha-bang

Koteswararao Kondepu

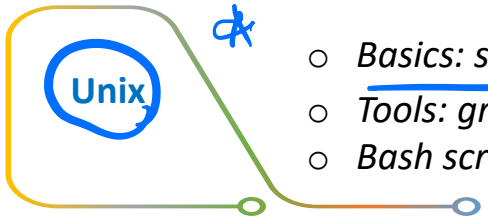
k.kondepu@iitdh.ac.in

→ script1.sh ✓

chmod +x script1.sh

./script1.sh ✓

List of Topics [C213]



- *Basics: shell, file system, permissions, process hierarchy, process monitoring, ssh, rsync*
- *Tools: grep, find, head, tail, tar, cut, sort, sed, awk*
- *Bash scripting: I/O redirection, pipes, makefile, libraries and linking*

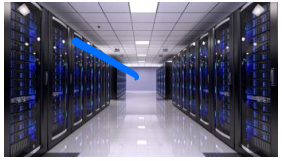
1

From Wikipedia, the free encyclopedia

Unix (/ˈjuːnɪks/; trademarked as **UNIX**) is a family of multitasking, multi-user computer operating systems that derive from the original AT&T Unix, whose development started in 1969[1] at the Bell Labs research center by Ken Thompson, Dennis Ritchie, and others.[4]

Unix Manual, first edition (bell-labs.com)

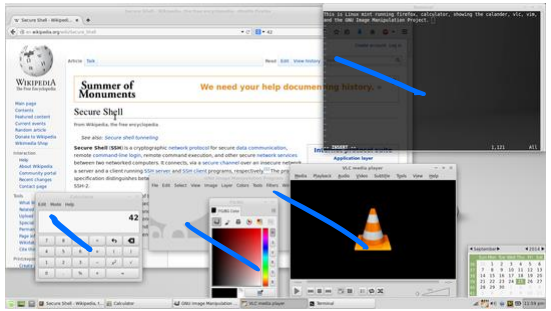
2



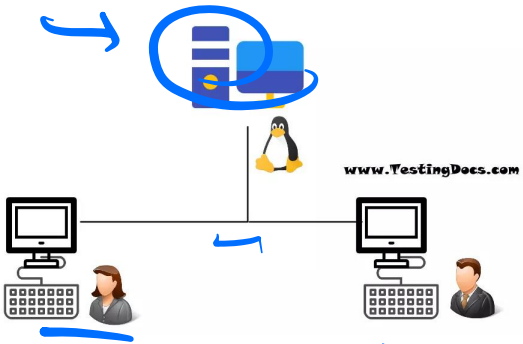
Unix distinguishes itself from its predecessors as the first portable operating system: almost the entire operating system is written in the C programming language, which allows Unix to operate on numerous platforms.[6]

Unix Features

1 Multitasking



2 Multiuser



3 Portable

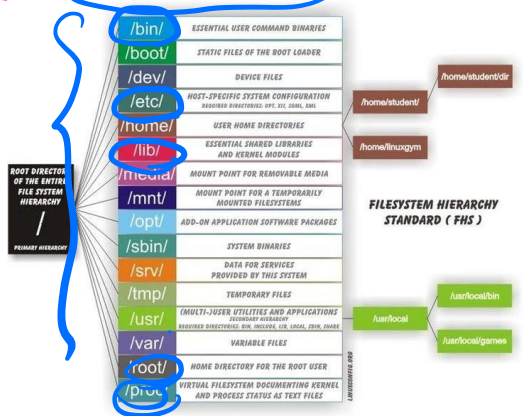
4 Security

5 Utilities and commands *****

- manintro.ps, manintro.pdf, Title page and Introduction
- man11.ps, man11.pdf, Commands, part 1
- man12.ps, man12.pdf, Commands, part 2
- man13.ps, man13.pdf, Commands, part 3
- man14.ps, man14.pdf, Commands, part 4
- man21.ps, man21.pdf, System calls, part 1
- man22.ps, man22.pdf, System calls, part 2
- man31.ps, man31.pdf, Library routines
- man41.ps, man41.pdf, Special files
- man51.ps, man51.pdf, File formats
- man61.ps, man61.pdf, User-maintained software
- man71.ps, man71.pdf, Miscellaneous

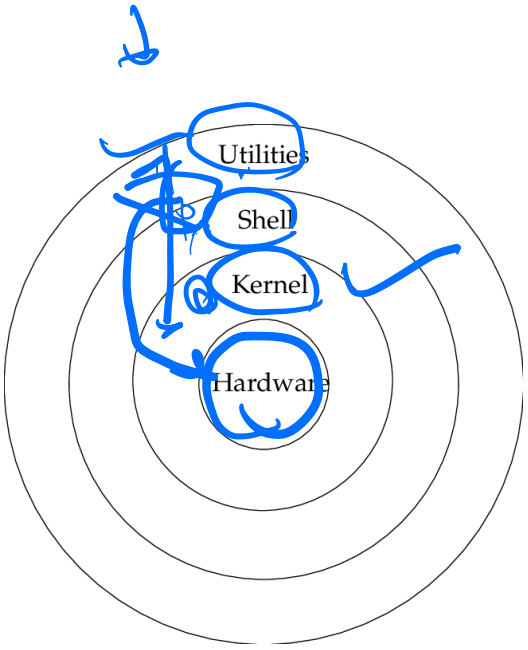
Unix Manual, first edition (bell-labs.com)

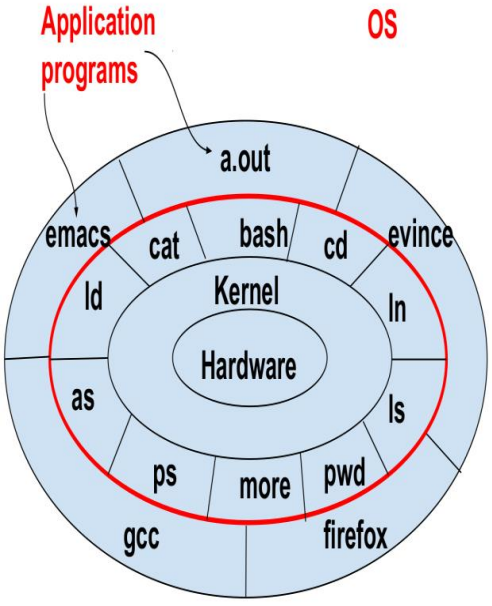
6 File and Directory *****



Hierarchy

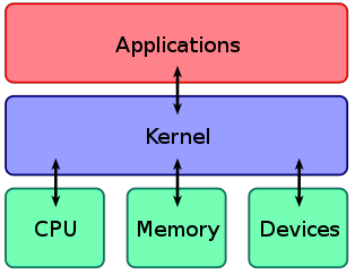
1. Kernel
2. Shell
3. Utilities and File Systems



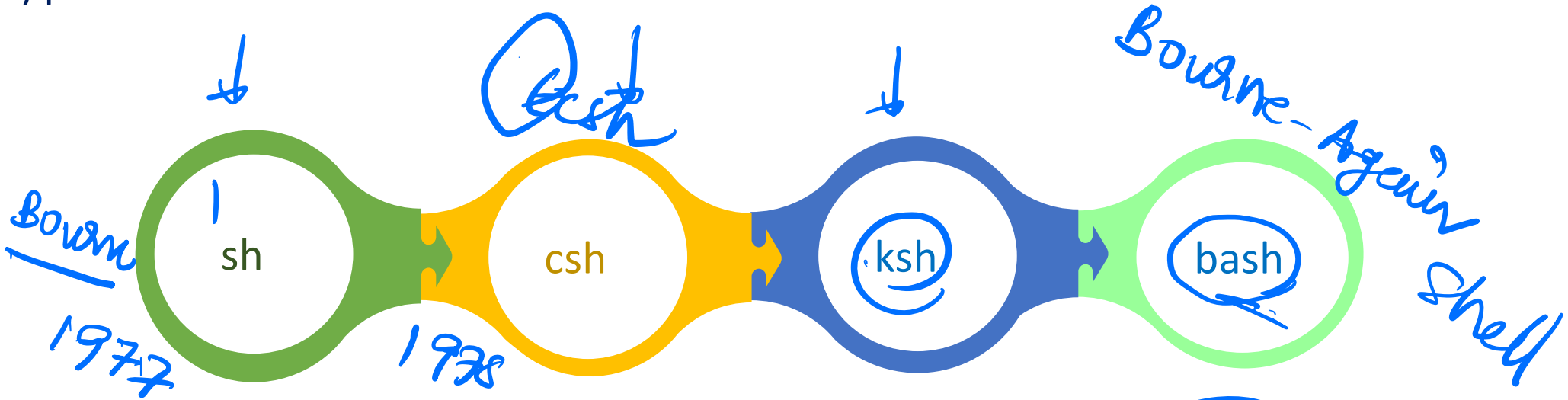


https://minnie.tuhs.org/cgi-bin/utree.plierarchy

https://www.bell-labs.com/usr/dmr/www/hist.html



Types of Shells

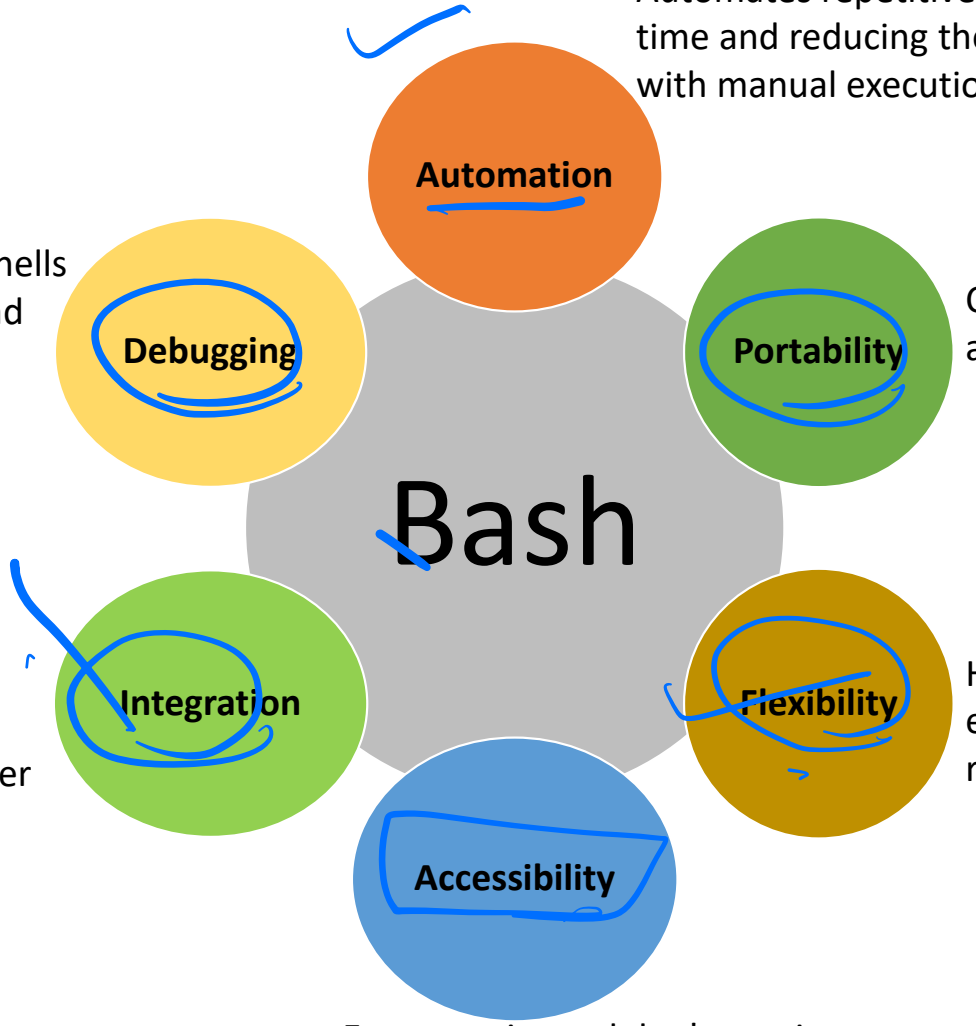


[8 Types of Linux Shells (phoenixnap.com)]

[The birth of the Bash shell | Opensource.com]

[Comparison of command shells - Wikipedia]

Bash Advantages



Automates repetitive tasks and processes, saving time and reducing the risk of errors that can occur with manual execution.

Easy to debug, and most shells have built-in debugging and error-reporting

Can be run on various platforms and operating systems

Highly customizable and can be easily modified to suit specific requirements

Can be integrated with other tools and applications

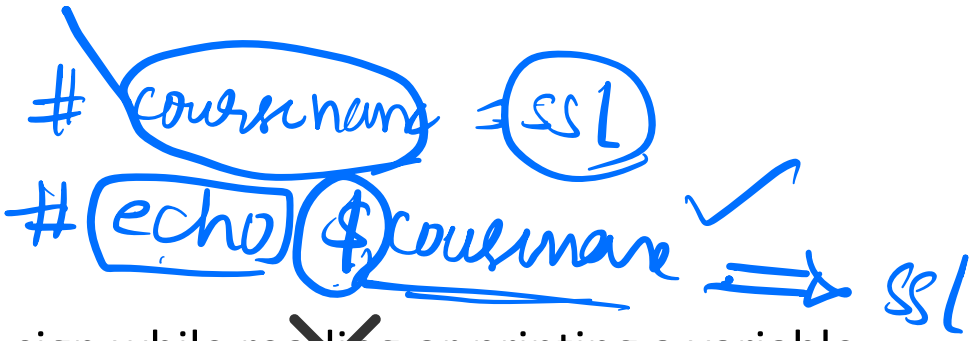
Easy to write and don't require any special tools or software.

Bash Installation in Windows 11

```
PS C:\Users\Utente> wsl --install
Installing: Virtual Machine Platform
Virtual Machine Platform has been installed.
Installing: Windows Subsystem for Linux
Windows Subsystem for Linux has been installed.
Installing: Ubuntu
Ubuntu has been installed.
The requested operation is successful. Changes will not be effective until the system is rebooted.
PS C:\Users\Utente>
```

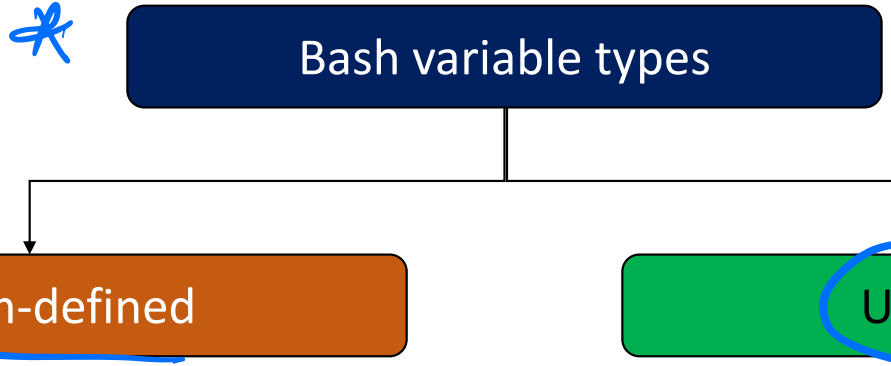
Bash Variables

- Variables → Specifies the memory location through characters, numeric, and alphanumeric.
  - Syntax: Variablename = value
- Rules Set for Defining Bash Variables:
  - Prefix the variable name with dollar ($) sign while reading or printing a variable.
  - Leave off the dollar sign ($) while setting a variable with any value.
  - A variable name is ***case-sensitive***
  - Bash variables are **untyped**



data type

Bash Variables (1)



- Pre-defined variables as they are created and maintained by the LINUX operating system itself.

- BASH, BASH_VERSION, HOME, LOGNAME, OSTYPE, PWD, USERNAME

- Try *declare, typeset, set, env, and printenv* commands to know about more variables

- Variables that are created and maintained by the user.

- Example: Coursename = cs213

Command	Description
env	Show environment variables
echo $NAME	Output value of $NAME variable
export NAME = value	Set $NAME to value
$PATH	Executable search path
$HOME	Home directory
$SHELL	Current shell

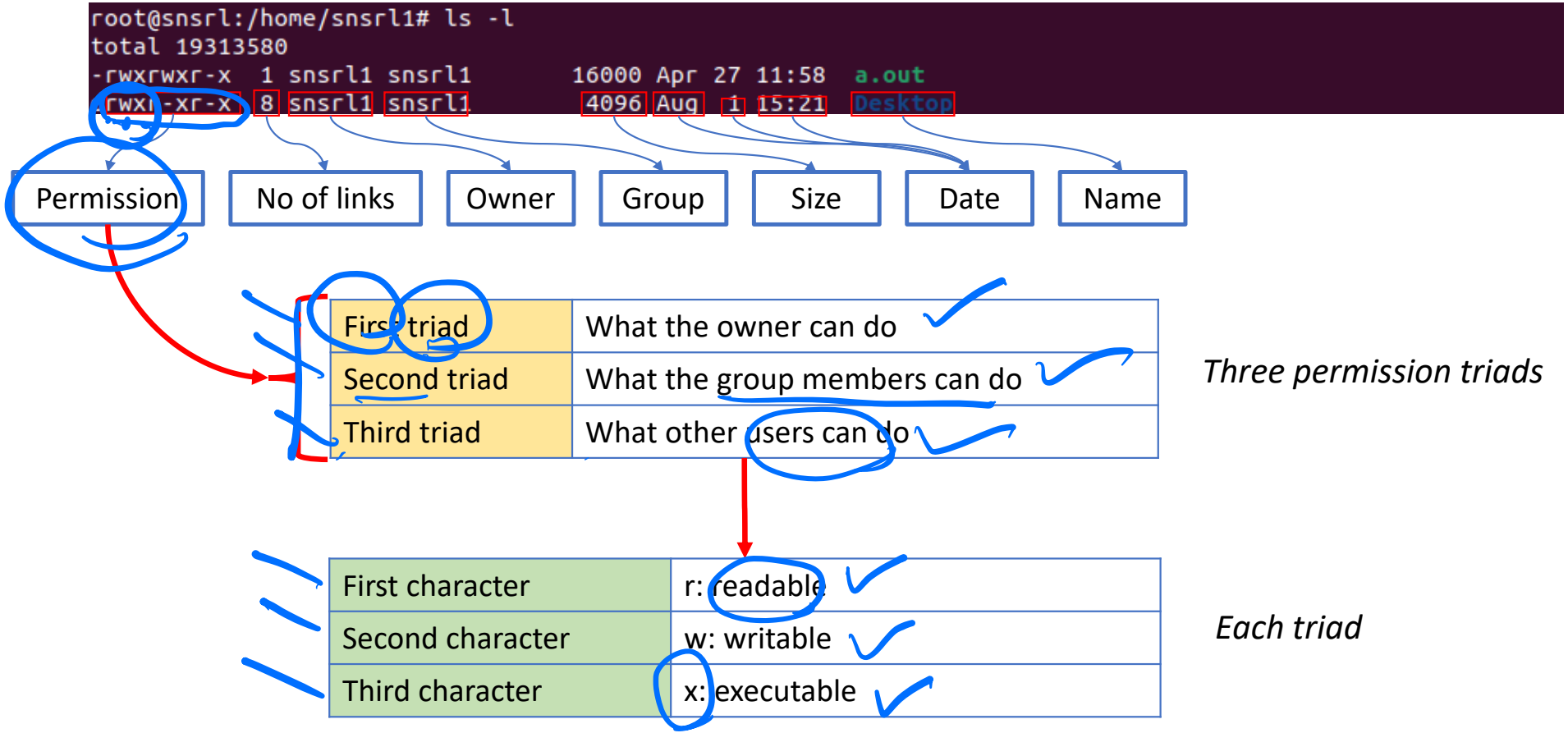
Bash File permission numbers

```
root@snsrl:/home/snsrl1# ls -l
total 19313580
-rwxrwxr-x  1 snsrl1 snsrl1       16000 Apr 27 11:58  a.out
drwxr-xr-x  8 snsrl1 snsrl1        4096 Aug  1 15:21  Desktop
```

Permission | No of links | Owner | Group | Size | Date | Name

- Owner and group
  - Each user is assigned a primary group
  - The system administrator can introduce the new group
  - The group can be changed (e.g. chgrp)

Bash File permission numbers (1)



Bash File permission numbers (2)

```
root@snsrl:/home/snsrl1# ls -l
total 19313580
-rwxrwxr-x  1 snsrl1 snsrl1       16000 Apr 27 11:58  a.out
drwxr-xr-x  8 snsrl1 snsrl1        4096 Aug  1 15:21  Desktop
```

Operation	*Octal value*
r: readable	4
w: writable	2
x: executable	1

Command	*Description*
chmod 755 cs213	Change mode of cs213 (file) to 775
chmod –R 600 cs213f	Recursively chmod cs213f (folder) to 600
chown root /cs213f	Change cs213f (folder) owner to root

Example:

Owner: rwx = 4+2+1 = 7
Group: rwx = 4+2+1 = 7
Others: r-x = 4+0+1 = 5

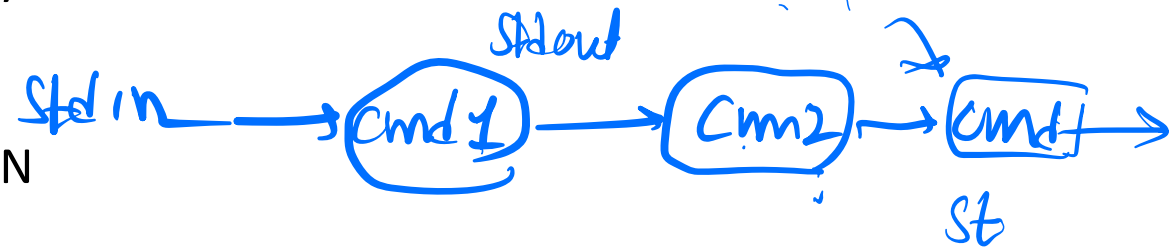
777

776

Bash Pipe (“|”)

- Used to combine two or more commands
- The output of one command acts as input to another command
- It can also be visualized as a temporary connection between two or more commands/ programs/ processes.

Syntax: cmd1 | cmd2 | ….. | cmdN



Command	***Description***
cmd1 | cmd2	Stdout of cmd1 to cmd2

*cmd - command

Utilities and commands: global regular expression print [grep]

```
grep [OPTION...] PATTERNS [FILE...]
```

grep syntax

- grep searches for PATTERNS in each FILE

- Finds each line that matched the provided PATTERN

```
snsrl1@snsrl:~$ grep --help
Usage: grep [OPTION]... PATTERNS [FILE]...
Search for PATTERNS in each FILE.
Example: grep -i 'hello world' menu.h main.c
PATTERNS can contain multiple patterns separated by newlines.

Pattern selection and interpretation:
  -E, --extended-regexp     PATTERNS are extended regular expressions
  -F, --fixed-strings       PATTERNS are strings
  -G, --basic-regexp        PATTERNS are basic regular expressions
  -P, --perl-regexp         PATTERNS are Perl regular expressions
  -e, --regexp=PATTERNS     use PATTERNS for matching
  -f, --file=FILE           take PATTERNS from FILE
  -i, --ignore-case         ignore case distinctions in patterns and data
      --no-ignore-case      do not ignore case distinctions (default)
  -w, --word-regexp         match only whole words
  -x, --line-regexp         match only whole lines
  -z, --null-data           a data line ends in 0 byte, not newline
```

Utilities and commands : grep (1)

grep [OPTION...] PATTERNS [FILE...]

grep syntax

Find a Matching String with grep

grep "unix" cs213.txt

```
snsrl1@snsrl:~$ grep "unix" cs213.txt
unix operating system. We will be using Andrew Linux and we
will see how we can use the power of unix to manipulate the
Andrew File System (AFS) and use unix tools and shell
be on the unix features that are more directly related to
focus on unix shell scripting, so we can develop powerful
scripts for managing tasks such as unix system calls, file
```

How to ignore case distinctions

grep -i "linux" cs213.txt

```
snsrl1@snsrl:~$ grep -i "UNIX" cs213.txt
unix operating system. We will be using Andrew Linux and we
will see how we can use the power of unix to manipulate the
Andrew File System (AFS) and use unix tools and shell
be on the unix features that are more directly related to
focus on unix shell scripting, so we can develop powerful
scripts for managing tasks such as unix system calls, file
```

```
snsrl1@snsrl:~$ grep --help
Usage: grep [OPTION]... PATTERNS [FILE]...
Search for PATTERNS in each FILE.
Example: grep -i 'hello world' menu.h main.c
PATTERNS can contain multiple patterns separated by newlines.

Pattern selection and interpretation:
  -E, --extended-regexp     PATTERNS are extended regular expressions
  -F, --fixed-strings       PATTERNS are strings
  -G, --basic-regexp        PATTERNS are basic regular expressions
  -P, --perl-regexp         PATTERNS are Perl regular expressions
  -e, --regexp=PATTERNS     use PATTERNS for matching
  -f, --file=FILE           take PATTERNS from FILE
  -i, --ignore-case         ignore case distinctions in patterns and data
      --no-ignore-case      do not ignore case distinctions (default)
  -w, --word-regexp         match only whole words
  -x, --line-regexp         match only whole lines
  -z, --null-data           a data line ends in 0 byte, not newline
```

cs213.txt

In this course, we will begin with an introduction to the unix operating system. We will be using Linux and we will see how we can use the power of unix to manipulate the File System (FS) and use unix tools and shell scripting to accomplish interesting tasks. Our focus would be on the unix features that are more directly related to writing, debugging and maintaining C programs. We will also focus on unix shell scripting, so we can develop powerful scripts for managing tasks such as unix system calls, file manipulation etc. To find out which version of the operating system you are running type

Utilities and commands : grep (2)

```
grep [OPTION...] PATTERNS [FILE...]
```

Grep syntax

How to Select the Non-Matching Lines

```
grep –v “unix” cs213.txt
```

```
snsrl1@snsrl:~$ grep -v "unix" cs213.txt
In this course, we will begin with an introduction to the
scripting to accomplish interesting tasks. Our focus would
writing, debugging and maintaining C programs. We will also
manipulation etc. To find out which version of the
operating system you are running type
```

How to Find the Line Numbers Against Matching Input

```
grep -n “linux” cs213.txt
```

```
snsrl1@snsrl:~$ grep -n "unix" cs213.txt
2 unix operating system. We will be using Andrew Linux and we
3 will see how we can use the power of unix to manipulate the
4 Andrew File System (AFS) and use unix tools and shell
6 be on the unix features that are more directly related to
8 focus on unix shell scripting, so we can develop powerful
9 scripts for managing tasks such as unix system calls, file
```

```
snsrl1@snsrl:~$ grep --help
Usage: grep [OPTION]... PATTERNS [FILE]...
Search for PATTERNS in each FILE.
Example: grep -i 'hello world' menu.h main.c
PATTERNS can contain multiple patterns separated by newlines.

Pattern selection and interpretation:
  -E, --extended-regexp     PATTERNS are extended regular expressions
  -F, --fixed-strings       PATTERNS are strings
  -G, --basic-regexp        PATTERNS are basic regular expressions
  -P, --perl-regexp         PATTERNS are Perl regular expressions
  -e, --regexp=PATTERNS     use PATTERNS for matching
  -f, --file=FILE           take PATTERNS from FILE
  -i, --ignore-case         ignore case distinctions in patterns and data
      --no-ignore-case      do not ignore case distinctions (default)
  -w, --word-regexp         match only whole words
  -x, --line-regexp         match only whole lines
  -z, --null-data           a data line ends in 0 byte, not newline
```

cs213.txt

In this course, we will begin with an introduction to the unix operating system. We will be using Linux and we will see how we can use the power of unix to manipulate the File System (FS) and use unix tools and shell scripting to accomplish interesting tasks. Our focus would be on the unix features that are more directly related to writing, debugging and maintaining C programs. We will also focus on unix shell scripting, so we can develop powerful scripts for managing tasks such as unix system calls, file manipulation etc. To find out which version of the operating system you are running type

Utilities and commands : find

find syntax

find [WHERE TO START SEARCHING FROM] [EXPRESSION DETERMINES WHAT TO FIND] [-OPTIONS] [WHAT TO FIND]

Find files starting with the given name in a directory

find ./Desktop –name “.txt”

```
snsrl1@snsrl:~$ find ./Desktop/ -name "*.txt"
./Desktop/5G-NR-SIM_COMMANDs.txt
./Desktop/cs213.txt
```

Find all empty folders and files in the given directory

find ./Desktop –empty

```
snsrl1@snsrl:~$ find ./Desktop/ -empty
./Desktop/database/oai_db2.sql
./Desktop/healthscripts/mysql-healthcheck2.sh
./Desktop/nssf_slice_config.yaml
```

Try ***locate*** command also

```
snsrl1@snsrl:~$ find --help
Usage: find [-H] [-L] [-P] [-Olevel] [-D debugopts] [path...] [expression]

default path is the current directory; default expression is -print
expression may consist of: operators, options, tests, and actions:
operators (decreasing precedence; -and is implicit where no others are given):
      ( EXPR )   ! EXPR   -not EXPR   EXPR1 -a EXPR2   EXPR1 -and EXPR2
      EXPR1 -o EXPR2   EXPR1 -or EXPR2   EXPR1 , EXPR2
positional options (always true): -daystart -follow -regextype

normal options (always true, specified before other expressions):
      -depth --help -maxdepth LEVELS -mindepth LEVELS -mount -noleaf
      --version -xdev -ignore_readdir_race -noignore_readdir_race
tests (N can be +N or -N or N): -amin N -anewer FILE -atime N -cmin N
      -cnewer FILE -ctime N -empty -false -fstype TYPE -gid N -group NAME
      -ilname PATTERN -iname PATTERN -inum N -iwholename PATTERN -iregex PATTERN
      -links N -lname PATTERN -mmin N -mtime N -name PATTERN -newer FILE
      -nouser -nogroup -path PATTERN -perm [-/]MODE -regex PATTERN
      -readable -writable -executable
      -wholename PATTERN -size N[bcwkMG] -true -type [bcdpflsD] -uid N
      -used N -user NAME -xtype [bcdpfls]       -context CONTEXT

actions: -delete -print0 -printf FORMAT -fprintf FILE FORMAT -print
      -fprint0 FILE -fprint FILE -ls -fls FILE -prune -quit
      -exec COMMAND ; -exec COMMAND {} + -ok COMMAND ;
      -execdir COMMAND ; -execdir COMMAND {} + -okdir COMMAND ;

Valid arguments for -D:
exec, opt, rates, search, stat, time, tree, all, help
Use '-D help' for a description of the options, or see find(1)

Please see also the documentation at https://www.gnu.org/software/findutils/.
You can report (and track progress on fixing) bugs in the "find"
program via the GNU findutils bug-reporting page at
https://savannah.gnu.org/bugs/?group=findutils or, if
you have no web access, by sending email to <bug-findutils@gnu.org>.
```

```
snsrl1@snsrl:~$ locate --help
Usage: plocate [OPTION]... PATTERN...

  -b, --basename          search only the file name portion of path names
  -c, --count             print number of matches instead of the matches
  -d, --database DBPATH   search for files in DBPATH
                          (default is /var/lib/plocate/plocate.db)
  -i, --ignore-case       search case-insensitively
  -l, --limit LIMIT       stop after LIMIT matches
  -0, --null              delimit matches by NUL instead of newline
  -N, --literal           do not quote filenames, even if printing to a tty
  -r, --regexp            interpret patterns as basic regexps (slow)
      --regex             interpret patterns as extended regexps (slow)
  -w, --wholename         search the entire path name (default; see -b)
      --help              print this help
      --version           print version information
```

Utilities and commands : head

```
head [OPTION] FILE_NAME
```

head syntax

show the specified number of lines

```
head –n 2 example.txt
```

```
snsrl1@snsrl:~$ head -n 2 example.txt
Connecticut
Delaware
```

show the specified number of bytes

```
head –c 20 example.txt
```

```
snsrl1@snsrl:~$ head -c 20 example.txt
Connecticut
Delawaresnsrl1@snsrl:~$
```

```
snsrl1@snsrl:~$ head --help
Usage: head [OPTION]... [FILE]...
Print the first 10 lines of each FILE to standard output.
With more than one FILE, precede each with a header giving the file name.

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

Mandatory arguments to long options are mandatory for short options too.
  -c, --bytes=[-]NUM       print the first NUM bytes of each file;
                             with the leading '-', print all but the last
                             NUM bytes of each file
  -n, --lines=[-]NUM       print the first NUM lines instead of the first 10;
                             with the leading '-', print all but the last
                             NUM lines of each file
  -q, --quiet, --silent    never print headers giving file names
  -v, --verbose            always print headers giving file names
  -z, --zero-terminated    line delimiter is NUL, not newline
      --help     display this help and exit
      --version  output version information and exit

NUM may have a multiplier suffix:
b 512, kB 1000, K 1024, MB 1000*1000, M 1024*1024,
GB 1000*1000*1000, G 1024*1024*1024, and so on for T, P, E, Z, Y.
Binary prefixes can be used, too: KiB=K, MiB=M, and so on.

GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
Full documentation <https://www.gnu.org/software/coreutils/head>
or available locally via: info '(coreutils) head invocation'
```

```
snsrl1@snsrl:~$ head example.txt
Connecticut
Delaware
Georgia
Maryland
Massachusetts
New Hampshire
New Jersey
New York
North Carolina
Pennsylvania
```

Utilities and commands : tail

```
tail [OPTION] FILE_NAME
```

tail syntax

show the specified number of lines

```
tail –n 4 example.txt
```

```
snsrl1@snsrl:~$ tail -n 4 example.txt
Pennsylvania
Rhode Island
South Carolina
Virginia
```

show the specified number of bytes

```
tail –c 20 example.txt
```

```
snsrl1@snsrl:~$ tail -c 20 example.txt
h Carolina
Virginia
```

```
snsrl1@snsrl:~$ tail example.txt
Maryland
Massachusetts
New Hampshire
New Jersey
New York
North Carolina
Pennsylvania
Rhode Island
South Carolina
Virginia
```

```
snsrl1@snsrl:~$ tail --help
Usage: tail [OPTION]... [FILE]...
Print the last 10 lines of each FILE to standard output.
With more than one FILE, precede each with a header giving the file name.

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

Mandatory arguments to long options are mandatory for short options too.
  -c, --bytes=[+]NUM       output the last NUM bytes; or use -c +NUM to
                             output starting with byte NUM of each file
  -f, --follow[={name|descriptor}]
                           output appended data as the file grows;
                             an absent option argument means 'descriptor'
  -F                       same as --follow=name --retry
  -n, --lines=[+]NUM       output the last NUM lines, instead of the last 10;
                             or use -n +NUM to output starting with line NUM
      --max-unchanged-stats=N
                           with --follow=name, reopen a FILE which has not
                             changed size after N (default 5) iterations
                             to see if it has been unlinked or renamed
                             (this is the usual case of rotated log files);
                             with inotify, this option is rarely useful
      --pid=PID            with -f, terminate after process ID, PID dies
  -q, --quiet, --silent    never output headers giving file names
      --retry              keep trying to open a file if it is inaccessible
  -s, --sleep-interval=N   with -f, sleep for approximately N seconds
                             (default 1.0) between iterations;
                             with inotify and --pid=P, check process P at
                             least once every N seconds
  -v, --verbose            always output headers giving file names
  -z, --zero-terminated    line delimiter is NUL, not newline
      --help     display this help and exit
      --version  output version information and exit

NUM may have a multiplier suffix:
b 512, kB 1000, K 1024, MB 1000*1000, M 1024*1024,
GB 1000*1000*1000, G 1024*1024*1024, and so on for T, P, E, Z, Y.
Binary prefixes can be used, too: KiB=K, MiB=M, and so on.

With --follow (-f), tail defaults to following the file descriptor, which
means that even if a tail'ed file is renamed, tail will continue to track
its end.  This default behavior is not desirable when you really want to
track the actual name of the file, not the file descriptor (e.g., log
rotation).  Use --follow=name in that case.  That causes tail to track the
named file in a way that accommodates renaming, removal and creation.

GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
Full documentation <https://www.gnu.org/software/coreutils/tail>
or available locally via: info '(coreutils) tail invocation'
```

Utilities and commands : sort

```
sort [OPTION]... [FILE]...
```

sort syntax

show the contents in sorted order

```
sort example.txt
```

```
snsrl1@snsrl:~$ sort example.txt
Connecticut
Delaware
Georgia
Maryland
Massachusetts
New Hampshire
New Jersey
New York
North Carolina
Pennsylvania
Rhode Island
South Carolina
Virginia
```

cat /etc/group | sort

```
snsrl1@snsrl:~$ sort --help
Usage: sort [OPTION]... [FILE]...
  or:  sort [OPTION]... --files0-from=F
Write sorted concatenation of all FILE(s) to standard output.

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

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

  -b, --ignore-leading-blanks  ignore leading blanks
  -d, --dictionary-order      consider only blanks and alphanumeric characters
  -f, --ignore-case           fold lower case to upper case characters
  -g, --general-numeric-sort  compare according to general numerical value
  -i, --ignore-nonprinting    consider only printable characters
  -M, --month-sort            compare (unknown) < 'JAN' < ... < 'DEC'
  -h, --human-numeric-sort    compare human readable numbers (e.g., 2K 1G)
  -n, --numeric-sort          compare according to string numerical value
  -R, --random-sort           shuffle, but group identical keys.  See shuf(1)
      --random-source=FILE    get random bytes from FILE
  -r, --reverse               reverse the result of comparisons
      --sort=WORD             sort according to WORD:
                                general-numeric -g, human-numeric -h, month -M,
                                numeric -n, random -R, version -V
  -V, --version-sort          natural sort of (version) numbers within text

Other options:

      --batch-size=NMERGE   merge at most NMERGE inputs at once;
                            for more use temp files
  -c, --check, --check=diagnose-first  check for sorted input; do not sort
  -C, --check=quiet, --check=silent  like -c, but do not report first bad line
      --compress-program=PROG  compress temporaries with PROG;
                              decompress them with PROG -d
      --debug               annotate the part of the line used to sort,
                              and warn about questionable usage to stderr
      --files0-from=F       read input from the files specified by
                            NUL-terminated names in file F;
                            If F is - then read names from standard input
  -k, --key=KEYDEF          sort via a key; KEYDEF gives location and type
  -m, --merge               merge already sorted files; do not sort
  -o, --output=FILE         write result to FILE instead of standard output
  -s, --stable              stabilize sort by disabling last-resort comparison
  -S, --buffer-size=SIZE    use SIZE for main memory buffer
  -t, --field-separator=SEP  use SEP instead of non-blank to blank transition
  -T, --temporary-directory=DIR  use DIR for temporaries, not $TMPDIR or /tmp;
                              multiple options specify multiple directories
      --parallel=N          change the number of sorts run concurrently to N
  -u, --unique              with -c, check for strict ordering;
                              without -c, output only the first of an equal run
```

thank you!

email:
k.kondepu@iitdh.ac.in