|  |  |  |
| --- | --- | --- |
| Command | Description | EXAMPLE |
| ssh | connect to a remote host | ssh user@name.cs.umass.edu |
| passwd | Change account password! |  |
| cd | Change directory! | If we start in /usr/local/src, then  cd /home/hussam  cd . /usr/local/src  cd .. /usr/local  ~ - current user’s home directory  . - the current directory  .. - the parent directory of the current directory |
| ls | List directory contents | ls <folder>  ls –ltr  -a : lists hidden ﬁles  -i : to display inode number of file  -d : to show directories not the contents of dir |
| pwd | Print working directory |  |
| touch [flags] <file> | The easiest way to create an empty ﬁle is touch   * With no ﬂags uses the current date/time | touch <filename> |
| mkdir [flags] <dir> | Makes a new directory with the speciﬁed names |  |
| rm [flags] <filename> | Removes the ﬁle called <filename>  **By default, rm cannot remove directories** | rm \* - removes every file in the current dir  rm \*.jpg - removes every .jpg file in the current dir  rm -i filename - prompt before deletion |
| rmdir [flags] <dir>  To delete a directory and all its subdirectories, we pass rm the ﬂag -r (for recursive) | * rmdir Removes a empty directory * Throws an error if the directory is not empty | **rm -r /home/user1/oldstuff** (recursive delete of directory and its sub directories) |
| cp [flgs] <file> <dest> | * Copies a ﬁle from one location to another * To copy multiple ﬁles you can use wildcards (such as \*) * To copy a complete directory use   **cp -r <src> <dest>** | cp \*.mp3 /Music/  - copies all .mp3 ﬁles from the **current directory** to /home/<username>/Music |
| mv [flgs] <src> <dest> | * move command automatically recuses for directories * Moves a ﬁle or directory from one place to another   Also used for **renaming**, just move from <oldname> to <newname> |  |
| man <command\_name> |  |  |
| File permissions , changing group permission amd changing ownership | | |
| -rwxrwxrwx  User’s Permissions Group’s Permissions Other’s permissions | R = Read, W = Write, X = Execute  Dir Permissions begin with a **d**  instead of a - |  |
| File Ownership :  how do you change the permissions of your own ﬁles?  chmod <mode> <file>  r – 2^2 = 4 ,  w – 2^1= 2 ,  x -2^0 = 1 | Changes ﬁle/directory permissions based on <mode>  The format of <mode> is a combination of 3 ﬁelds:  Who is affected - a combination of  **u, g, o, or a (all)**  Whether adding or removing permissions **+ or –**  Which permissions are being added/removed - **r, w, x.** | **chmod ug+rx myfile** adds read and execute permissions for user and group.  **chmod a-r myfile** remove read access for everyone  **chmod ugo-rwx** removes all permissions from myﬁle  **chmod -R o-w** ~/Documents/  **-R is the recursive** option  removes write privileges for other uses for every ﬁle and every directory in ~/Documents/  **chmod 755** : rwxr-xr-x  **chmod 600** : rw------  **chmod 777** : rwxrwxrwx |
| chgrp group <target> |  |  |
| chown user:group <target> |  |  |
| umask mode | Removes mode from the ﬁle’s permission |  |
| Reading Files : we only want to see what is in a ﬁle without opening it for editing. | | |
| cat <filename>  cat <file1> <file2> | Prints the contents of the ﬁle to the terminal window  Prints the ﬁrst ﬁle then the second which is what it is really for |  |
| more <filename> | allows you to scroll **through the ﬁle 1 page at a time** |  |
| less <filename> | Lets you scroll up and down **by pages or lines** |  |
| head -[num] <flnm>  tail -[numlines] <filnm> | Prints the ﬁrst/last numlines of the ﬁle  Default is 10 lines | **tail /var/log/Xorg.0.log**  Prints the last ten lines of the log ﬁle. |
| echo <text\_string>  Prints the input string to terminal  echo \* | Prints the input string to the standard output (the terminal)  The shell expanded **\*** **to all files in the current directory**. This is an example of path expansion, one type of shell expansion. | echo This is a string  echo ’This is a string’  echo "This is a string"  Lec1.pdf Lec1.dvi Lec1.tex Lec1.aux |
| ln [options] <target file> [link\_name]  ln -s <target\_file> [link\_name] | we can create links to ﬁles and directories. There are two types of links, **hard links and symbolic links.**  Creates a link to <target file> at [link\_name], defaulting to the current directory  The link points to the same ﬁle on the system i.e. the link is indistinguishable from the original ﬁle  Creates a symbolic link to the target ﬁle or directory  The link ﬁle contains a string that is the pathname of the original ﬁle or directory.  In other words the symbolic link points to the other ﬁle |  |
| alias name=command | **The alias allows you to rename or type something simple instead of typing a long command.**  You can set an alias for your current session at the command prompt.  To set an alias more permanently add it to your **.bashrc** or **.bash\_profile** file in your home directory | alias ls=‘ls --color=auto’  alias dc=cd  alias ll="ls -l" |
| file <filename> | Displays the ﬁle type. |  |
| cut [-b] [-c] [-d delim] [-f list] [-s] [file]  delim is a delimiter that separates fields  list consists of one of N, N-M, N  -d: especifies a delimiter (tab by default)  -f: especifies a range of fields separated by a delimiter  -s: supressses line if delimiter is not found  -b: extracts using range of bytes  -c: extracts using range of characters | cut extract sections from each line of the input | cut **-d : -f 1 -s** employee.txt  cut **-d : -f 3,4 -s** employee.txt  cut **-d : -f 2** employee.txt  cut **-d : -c 1** employee.txt |
| paste [options] [file1 ...]  -d : paste concatenate files side-by-side.  -s : concatenates serialy instead of side-by-side | paste concatenate files side-by-side. | paste -d : names.txt phones.txt  paste names.txt phones.txt |
| split [options] file1] [prefix]  -l: how many lines in each file  -b: how many bytes in each file  prefix: name prefix of each file produced | Splits a files into pieces, i.e., files named xaa, xab, ... |  |
| join [options] file1 file2  -1 field: join by the field-th field of file 1  -2 field: join by the field-th field of file 2  -a file number: displays unpaired lines of file file number | Join lines that contain the same keys between two different files. | **Age.txt**  Alice 12  Bob 30  Charlie 23  **Salaries.txt**  Bob 129,000  Charlie 75,000  Join –a1 age.txt salaries.txt  Bob 30 129,000  Charlie 23 75,000  Alice 12 |
| bc [options]  -l field: increase the precision to 20 decimal places (default 0) | Performs arithmetic and logical calculations | echo "1/3" | bc  0  echo "1/3" | bc -l 0.33333333333333333333  echo "1>3" | bc -l  0 |
| find [where to look] criteria [what to do]  -name : name of file or directory to look for  -maxdepth num : descend at most num levels of directories while searching  -mindepth num : descend at least num levels of directories while searching  -amin n : file last access was n minutes ago  -atime n : file last access was n days ago  -group name : file belongs to group name  -path pattern : file name matches shell pattern pattern  -perm mode : file permission bits are set to mode  -type d : type is directory  -type f : type is file  -print : print to console  -perm : permission on file  -empty : empty files  -user : Search files based on user  -group : find files or dir based on group  -size : find files based on size | * used to locate files or directories * search any set of directories for files that match a criteria * search by name, owner, group, type, permissions, last modification date, and other criteria * search is recursive (will search all subdirectories too)   normally all modifiers for find are evaluated in conjunction (i.e. AND). We can find files matching a pattern OR another by using the -o flag.  executes a command on found files by using the **-exec command ‘{}’ +** flag.  **executes a command on found files by using the -exec command ‘{}’ \; flag.**  The difference between **\;** and **+** is that with **\; a single grep command for each file** is executed whereas with **+ as many files as possible** are given as parameters to grep at once. | display pathnames of all files in current directory and subdirectories  find . –print  find .  search for a **file by name** find . -name my awesome file.txt  find **. -name** my awesome file.txt  Find all files accessed **at most** 10 minutes ago  find . -amin **-10**  Find all files accessed **at least 10 minutes** ago  find . **–atime +10**  Find all the files whose permissions are 777.  find . -type f -perm 0777 -print  To find all files that belongs to user Tecmint under /home directory.  find /home –user Techmint –type f  Display all the contents of files accessed in the last 10 minutes  **find . –atime -10 –exec cat {} +** |
| sed [options] [script] [file]  sed ‘s/<regex>/<text>’ [file]  This form replaces anything that matches <regex> with <text>.  sed goes line by line searching for the regular expression.  sed ’/regexp/d’  deletes all the line that contain the regex  sed ‘s/(regex)/\1/p’  sed -n : Supress automatic printng of pattern  sed -e : specifies the regular expression pattern  sed ‘s///p’  Sed substitution  In the substituion , using sed we can change the part and also put the required data using *& .*  Replace test with testHareesha  cat input.txt | sed ‘s/test/&Hareesha/g’  sed 's/^.\*$/<<<&>>>/g' path.txt | sed is a stream editor  Stream editor for filtering and transforming text  Here **d** inidcated that if match is found then delete whole line  **\1** holds the value that is specified in braces  **p** indicated if substitution was made print the new pattern space  **g** indicate to **replace the match with the substitution on all occurances**.If **g is not given** the substitution is made only on the first occurance.  **&** holds the matched part. So in the substitution part it can can be used  This is a sed command that slelects wole line and which is captured by **&. And it is relpaced <<<&>>> at the start and end of line.** | sed ’s/not guilty/guilty/**g**’ filename  Replaces **not guilty** with **guilty** everywhere in the file.  Without the **g**, it will only do one  substitution per line  Sed ‘/[Dd]avid/**d**’ filename > filename2  Delete all lines that contain david or David  Example:  **Input.txt**  This is a **test** file that is testing on sed substitution  **Output**  This is a **testHareesha** file that is testing on sed substitution |
| basename | Used to trim off the suffix and also the directory path. |  |
| crontab –e  -e flag, which lets you edit your cron file  crontab -l to display a listing of your cron jobs. | **crontab** command to schedule commands or scripts to run regularly at times you specify  When scheduling cron jobs, you need to specify full and absolute paths to the file  Given crontab –e command to open a file and enter the cron expression followed by the command that needs to be executed | When entering a cron job, you specify  **Minutes (0–59)**  **Hours (0–23)**  **Day of the month (1–31)**  **Month (1–12)**  **Day of the week (0–6, with Sunday as 0)**  If you replace the number with a **\***, cron will match all possible values, so, if a job is scheduled for  **Use a hyphen (-) to indicate a range** |

|  |  |  |
| --- | --- | --- |
| AWK : awk is meant for processing column-oriented text data, such as tables, presented to it on standard input. The variables $1, $2, and so forth are the contents of the first, second, etc. column of the current input line. | | |
|  |  |  |
|  |  |  |

|  |  |
| --- | --- |
| grep <string>[file]  grep -i - ignores case  grep -A 20 -B 10 - prints the 10 lines before and 20 lines after each match  grep -v - inverts the match or print non matching lines  grep -o - shows only the matched substring  grep -n - displays the line number  grep -w : Checking for full words, not for sub-strings grep -A <N> : Display N lines after match grep -B <N> : Display N lines before match grep –C <N> : Display N lines around match grep –i 🡪Ignore case and match  grep –E ----🡪 This is extended grep that is it doesn’t require escape sequence for meta characters  grep –v 🡪 Inverts the match  grep –l -🡪prints only names of the matching file names  grep –c --🡪count of the matching lines  grep -r -🡪 does the recursive search on all filess  grep –H 🡪 display filename of the matching string as well  grep -q : -q option prevents grep from writing anything to standard output. This is useful when grep is embedded within a shell script and you want to check if a pattern exists in one or more files but do not want to generate any output during processing. | The purpose of grep is to print the lines that match a particular pattern.  grep -v # bashscript  Prints all noncommented lines |

|  |  |  |
| --- | --- | --- |
| Regular Expressions | | |
| special characters:  $ < > & ? { } () | | The shell interprets them in a special way unless we escape (\$) or place them in quotes “$” | **[] \* ^ .**  No escapng is needed unless they are part of string that need to be parsed |
| \* - matches 0 or more occurrences of the expression    \? - matches 0 or 1 occurrences of the expression  \+ - matches 1 or more occurrences of the expression  The above are quantative  . matches any single character by itself , not that it takes the preceing character into consideration | A RegExp followed by one of these repetition operators defines how many times that pattern should be matched | Lec\* Lecture1.pdf Lec.avi  Lecture?.pdf Lecture1.pdf |
| ^ matches the beginning of a line  $ matches the end of a line |  |  |
| \{n\} - preceding item is repeated exactly n times  \{n,\} - preceding item is repeated at least n times  \{i,j\} matches between i and j occurrences of strings that match e | Matching A Range of Repetitions |  |
| \(expr\) : matches expr  c1\|c2 matches the expression c1 or the expression c2. | useful for grouping expressions together |  |
| \< matches the beginning of a word \> matches the end of a word |  | match any word that begins with **c** and ends with **d**  grep **‘\<c.\*d\>’**  want **5** letter words beginning with **c** and ending with **d**  **grep ‘\<c…d\>’** |
| [...] | Matches any character inside the square brackets  Use a dash to indicate a range of characters  Can put commas between characters/ranges | [SL]ec\* Lecture Section  Day[1-4].pdf  Day1.pdf Day2.pdf |
| [^...] | matches any character not inside the square brackets | [^A-P]ec\* Section.pdf  [^A-Za-z]\*  9Days.avi  .bash\_profile |
| {... , ...} | matches any phrase inside the comma-separated brackets |  |
|  | With extended regular expressions you do not need to escape special characters such as ?, +, () and {}.  To use extended regular expressions with grep use the **variant egrep or grep -E**.  Extended regular expressions tend to be cleaner and easier to read: | **grep** ’\(woo\+t\)\{2,3\}’  becomes  **egrep** ’(woo+t){2,3}’ |
| [:alnum:] - alphanumeric characters  [:alpha:] - alphabetic characters [:digit:] - digits  [:punct:] - punctuation characters  [:lower:] - lowercase letters [:upper:] - uppercase letters [:space:] - whitespace characters | POSIX character classes | ls | grep **[[:digit:]]** |
| ^ when this is inside the brackets | negate ranges of characters: | **[^abc] -** matches any character that is not a b or **c**  **[^a-z]** - matches any non lowercase letter |

|  |  |  |
| --- | --- | --- |
| Compression & Archiving | | |
| gzip | Compress files using Lempel-Ziv coding  Does not bundle files, the compressed files will replace the original files | **gzip** <file-to-compress>  **gunzip** <compressed\_file> |
| bzip2 | Compress files using Burrows-Wheeler block sorting text compression algorithm and Huffman coding.  **More efficient than gzip on most files, but a bit slower.**  Like **gzip**, **this is** **only a compression tool, and thus compressed files will replace the original files.** | **bzip2** <file-to-compress>  **bunzip2** <compressed-files> |
| Tar   1. c – create a archive file. 2. x – extract a archive file. 3. v – show the progress of archive file. 4. f – filename of archive file. 5. t – viewing content of archive file. 6. j – filter archive through bzip2. 7. z – filter archive through gzip. 8. r – append or update files or directories to existing archive file. 9. W – Verify a archive file. 10. wildcards – Specify patters in unix tar command. | **tar** *bundles multiple files together into a single file* (but **does not compress them or replace them**)  Its like creating a zip file.  **tar -zxvf tecmintbackup.tar.gz one.xml**  To untar a sngle file one.xml from tar file  To **add files or directories to existing tar archived file** we use the option r (append) | **tar -cf archive.tar foo bar**  **Create** archive.tar from files foo and bar  **tar -xf archive.tar**  **Extract** all files from archive.tar  **-C** option is used to **extract files to specific dir**.If not specified gets extracted to pwd.  **tar -xvf public\_html-14-09-12.tar -C /home/public\_html/videos**/  **tar –tvf ex.tar** |
| tar –czf <archive name> <file names>  -z : compress using gzip  -j : compress using bzip2  tar -czf archive.tar.gz foo bar  tar -czf cs2042.tgz cs2042/\* | To compress a tarball we can pipe the outcome of tar to a tool like gzip or bzip2.  However, tar has flags to automatically do this  Works with directories also .. | **archive.tar.gz** or **archive.tgz**: gzipped tarballs archive.tar.**bz2** or archive.**tbz**: bzip2 tarballs  Creates a compressed file containing the directory and contents of cs2042 directory |
| tar –tvf  -t : list the contents of tar file without extracting.Its like opening tar file to see the contents |  |  |
| #! /bin/bash  tar -czf ~/backups/cs2042.backup.tar.gz \  ~/Documents/cs2042/ |  | To backup all the files of Document directories |

|  |  |
| --- | --- |
| Vi Editor | |
| Modes :  Insert mode – To add or modify contents of file  Command mode --- To issue some commands (default mode) |  |
| 1. To go to insert mode, insert before character 2. Insert after character   x- To delete a character, need to be in command mode.  dd- To delete entire line, need to be in command mode.  Ecs To go to command mode  Shift + ZZ together to come out of the file completely  :q is for quit  :wq is for save and quit  :w save the file  :set number To enable linenumber on the file |  |
| view filename | OPENS FILE IN READ ONLY MODE |

|  |  |  |
| --- | --- | --- |
| Informative | | |
| df [options] [filename] | reports the amount of available [disk](http://www.computerhope.com/jargon/d/disk.htm) space being used by [file systems](http://www.computerhope.com/jargon/f/filesyst.htm) |  |
| du [options] [filename] | du estimates and displays the [disk](http://www.computerhope.com/jargon/d/disk.htm) space used by [files](http://www.computerhope.com/jargon/f/file.htm). |  |
| ifconfig | To find the configuration of the system |  |
| env | Lists all the environment varailes |  |
| ps -aux | List all the processes |  |
| top | Prints the processe that has top CPU usage Process and system info |  |
| nice –n <proprity> | All to control the pripority of the processes.  **0** - Default  **-20** - high priority  **19** is the highest priority |  |
| renice | To reset the priority of the process |  |
| df -h | To find the empty space on the disk space |  |

**File Descriptors :**

File descriptor **2** represents **standard error**.

other special file descriptors include **0** for **standard input** and **1** for **standard output**).

|  |
| --- |
| Syntax to read file line by line on a Bash Unix & Linux shell:   1. The syntax is as follows for bash, ksh, zsh, and all other shells – 2. while read -r line; do COMMAND; done < input.file 3. The -r option passed to read command prevents backslash escapes from being interpreted. 4. Add IFS= option before read command to prevent leading/trailing whitespace from being trimmed - 5. while IFS= read -r line; do COMMAND\_on $line; done < input.file |

|  |
| --- |
| IFS The Internal Field Separator that is used for word splitting after expansion and to split lines into words with the read builtin command. The default value is "<space><tab><newline>"  **you must read to more than one variable at a time** for the IFS=<something> read ... construct to have a visible effect  **IFS** is used to separate line into number of variables based on the assigned IFS separater.  This comes in handy when we have to read a line and parse into based on a delimiter |

|  |
| --- |
| ***Arrays In UNIX***  arr=(one two three)  to call the elements  $ echo ${arr[0]}  one  $ echo ${arr[2]}  three |

|  |
| --- |
| ***HOW TO CONNECT TO MYSQL FROM SHELLSCRIT***  #!/bin/bash  echo "show all tables"  mysql -uroot -p'password' dbname<<EOFMYSQL  show tabbles;  EOFMYSQL  echo "Count of all records"  mysql -uroot -p'password' dbname<<EOFMYSQL  select count(\*) from tbname;  EOFMYSQL  #!/usr/bin/bash  #Script to run automated sql queries  #Declaring mysql DB connection  MASTER\_DB\_USER='username'  MASTER\_DB\_PASSWD='password'  MASTER\_DB\_PORT=3160  MASTER\_DB\_HOST='mysql.hostname'  MASTER\_DB\_NAME='mysqlDbName'  #Prepare sql query  SQL\_Query='select \* from tablename limit 10'  #mysql command to connect to database  MYSQL -u$MASTER\_DB\_USER -p$MASTER\_DB\_PASSWD -P$MASTER\_DB\_PORT -h$MASTER\_DB\_HOST -D$MASTER\_DB\_NAME **<<EOF**  $SQL\_Query  **EOF**  echo "End of script" |

|  |  |  |
| --- | --- | --- |
|  | ftp – FILE TRANSFER PROTOCOL |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |