# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Intermediate Bash Scripting**

*Summer internship*

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Pratush kumar pusti

### pratushkumarpusti@gmail.com

Beginning May 26, 2025

## Table of contents

May 26, 2025

1. Creating Remote Repository

May 27, 2025

1. bash Basics
   1. What is a shell
   2. History of UNIX Shells
   3. Getting bash
   4. Interactive Shell Use
   5. Files
   6. Input and Output

May 28, 2025

1. Background Jobs
2. Special character and quoting
3. Help
4. Command-Line Editing
5. Enabling Command-Line Editing (leaving emacs)
6. The History List
7. vi Editing Mode
8. The fc Command
9. History Expansion

May 29, 2025

1. readline
2. Keyboard Habits
3. Customizing Your Environment
4. The .bash\_profile, .bash\_logout, and .bashrc Files
5. Aliases
6. Options
7. Shell Variables

May 30, 2025

1. Customization and Subprocesses
2. Customization Hints

1. Basic Shell Programming
   1. Shell Scripts and Functions
   2. Shell Variables
   3. String Operators

May 31, 2025

1. Command Substitution
2. Advanced Examples: pushd and popd
3. Flow Control
4. if/else
5. for
6. case

Jun 1, 2025

1. select
2. while and until
3. Recorded video

Jun 2, 2025

1. Compiled and uploaded video

Jun 3, 2025

1. Edited the videos from ch 1-4
2. Completed Ch 5

Jun 4, 2025

1. Made slides of Ch 5
2. Command-Line Options
3. Typed Variables

Jun 5, 2025

1. Installed conda & TOGA on my mac
2. Made changes to .bashrc and .zshrc to add conda to my PATH
3. Integer Variables and Arithmetic

Jun 6, 2025

1. Arrays
2. I/O Redirectors

Jun 7, 2025

1. Made slides of Ch 6

Jun 8, 2025

1. Made video of Ch 5

Jun 9, 2025

1. String I/O

May 26, 2025

1. Created a remote repo on github and commited Summer\_internship.docx .
2. Added .gitignore and added README.md files.

May 27, 2025

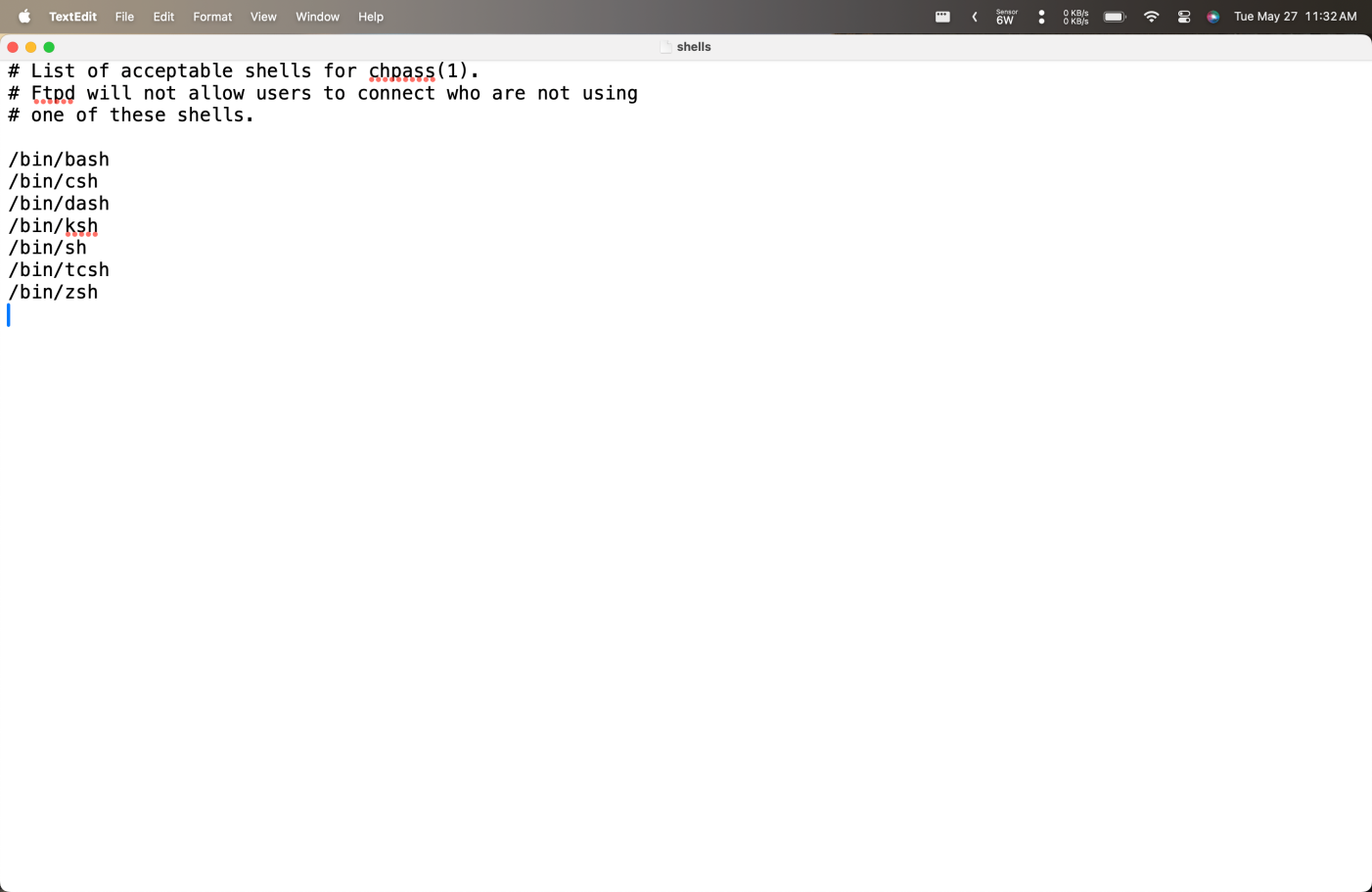
1. bash Basics :

What is a shell?

* The shell's job, is to translate the user's command lines into operating system instructions.
* The shell itself is not UNIX—just the user interface to it.
* UNIX is one of the first operating systems to make the user interface independent of the operating system.

History of UNIX shells

* The first major shell was the Bourne shell (named after its inventor, Steven Bourne). The Bourne shell is known on the system as *sh.*
* The first widely used alternative shell was the C shell, or csh. This was written by Bill Joy at the University of California at Berkeley as part of the Berkeley Software Distribution (BSD).
* The C shell gets its name from the resemblance of its commands to statements in the C Programming Language.It supports a number of operating system features like job control and aliases.
* Korn shell is a commercial product that incorporates the best features of the Bourne and C shells, plus many features of its own. It is similar to bash is most respects.
* Bash or Bourne Again shell was created for use in the GNU project. The GNU project was started by Richard Stallman of the Free Software Foundation (FSF) for the purpose of creating a UNIX-compatible operating system and replacing all of the commercial UNIX utilities with freely distributable ones.
* bash, intended to be the standard shell for the GNU system, was officially "born" on Sunday, January 10, 1988.
* Brian Fox wrote the original versions of bash and readline and continued to improve the shell up until 1993. Early in 1989 he was joined by Chet Ramey, who was responsible for numerous bug fixes and the inclusion of many useful features. Chet Ramey is now the official maintainer of bash and continues to make further enhancements.
* Some of the features of bash that have lead to it becoming quite popular include command-line editing modes and job control. With command-line editing, it's much easier to go back and fix mistakes. Job control gives you the ability to stop, start, and pause any number of commands at the same time.

*Image listing the shells available on the system (/etc/shells file)*

Getting Bash

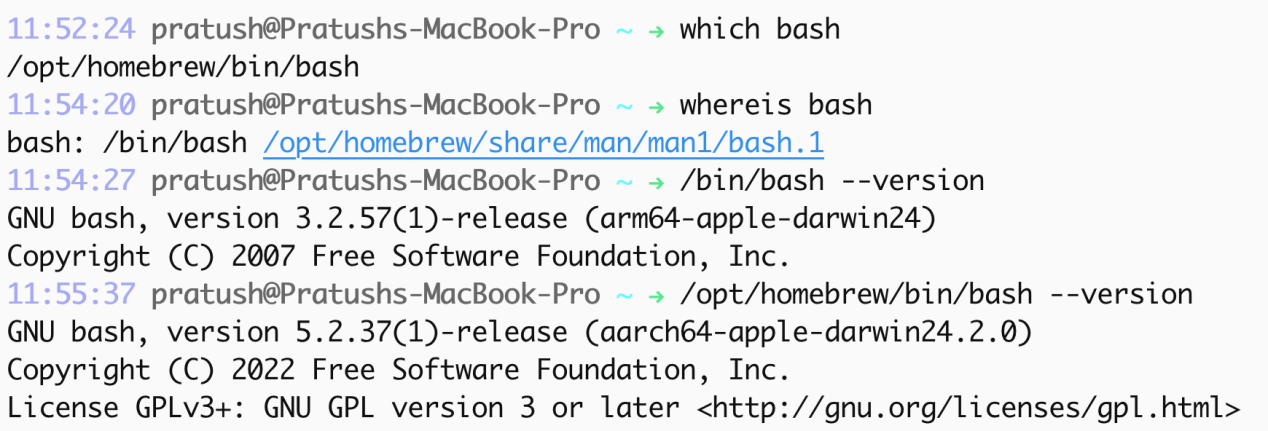
* Log in to your system and type echo $SHELL at the prompt. You will see a response containing sh, csh, ksh, or bash. Just type bash to change the shell to bash.
* To find out where bash is on your system, i.e., in which directory it's installed. Just type whereis bash, if that doesn't work, try whence bash or which bash.
* To install bash as your login shell, type chsh bash-name {location of bash from whereis bash cmd}



*output to echo $SHELL cmd*



*version of bash installed*



*Results to which bash, whereis bash and the versions of preinstalled bash and bash installed using brew package manager.*

Interactive Shell Use

* When you use the shell interactively, you engage in a login session that begins when you log in and ends when you type exit or logout or press CTRL-D.
* Shell command lines consist of one or more words, which are separated on a command line by blanks or TABs. The first word on the line is the *command*. The rest (if any) are *arguments*.
* An option is a special type of argument that gives the command specific information on what it is supposed to do.

Files

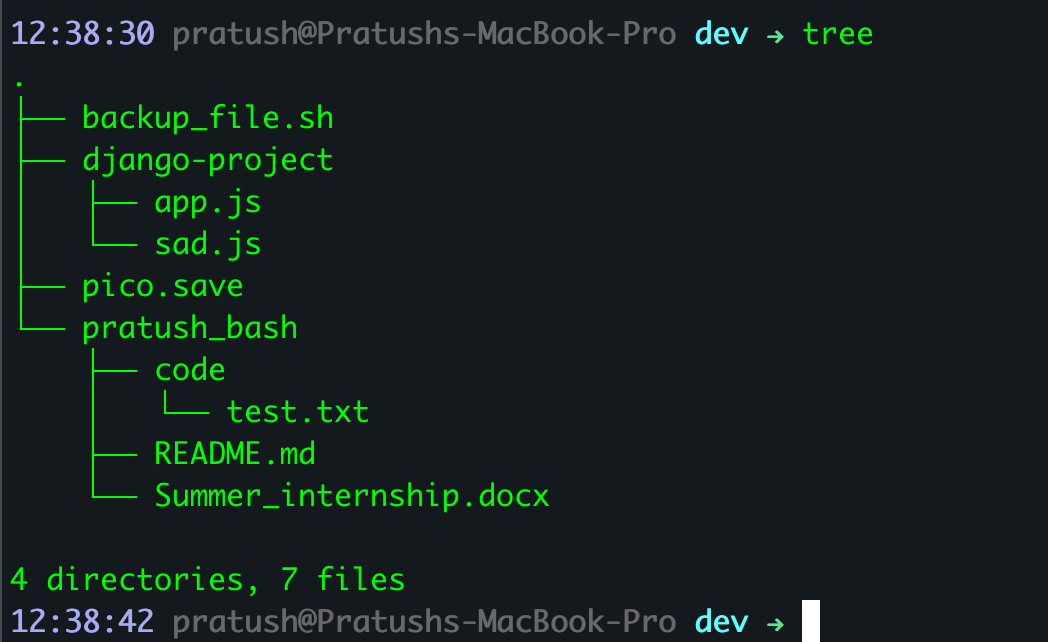
* A file can contain any kind of information, and indeed there are different types of files. Three types are by far the most important:
* Regular files: Also called text files; these contain readable

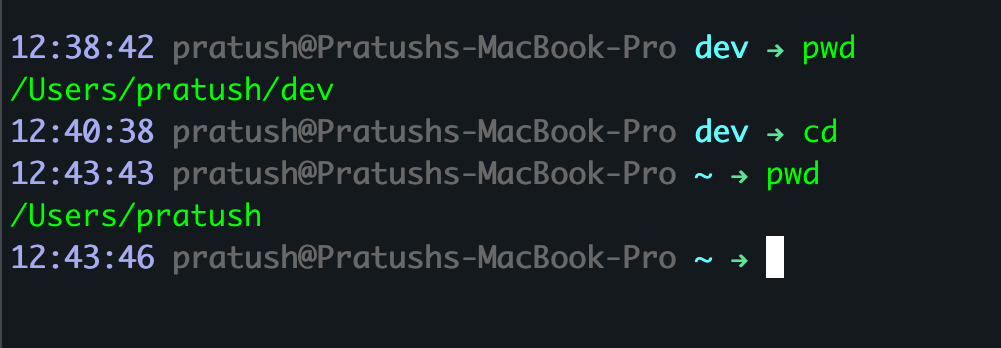
characters.

* Executable files: Also called programs; these are invoked as commands. Some can't be read by humans. The shell scripts are just special text files. The shell itself is a (non-human-readable) executable file called bash.
* Directories: These are like folders that contain other files—possibly other directories (called subdirectories).

Directories

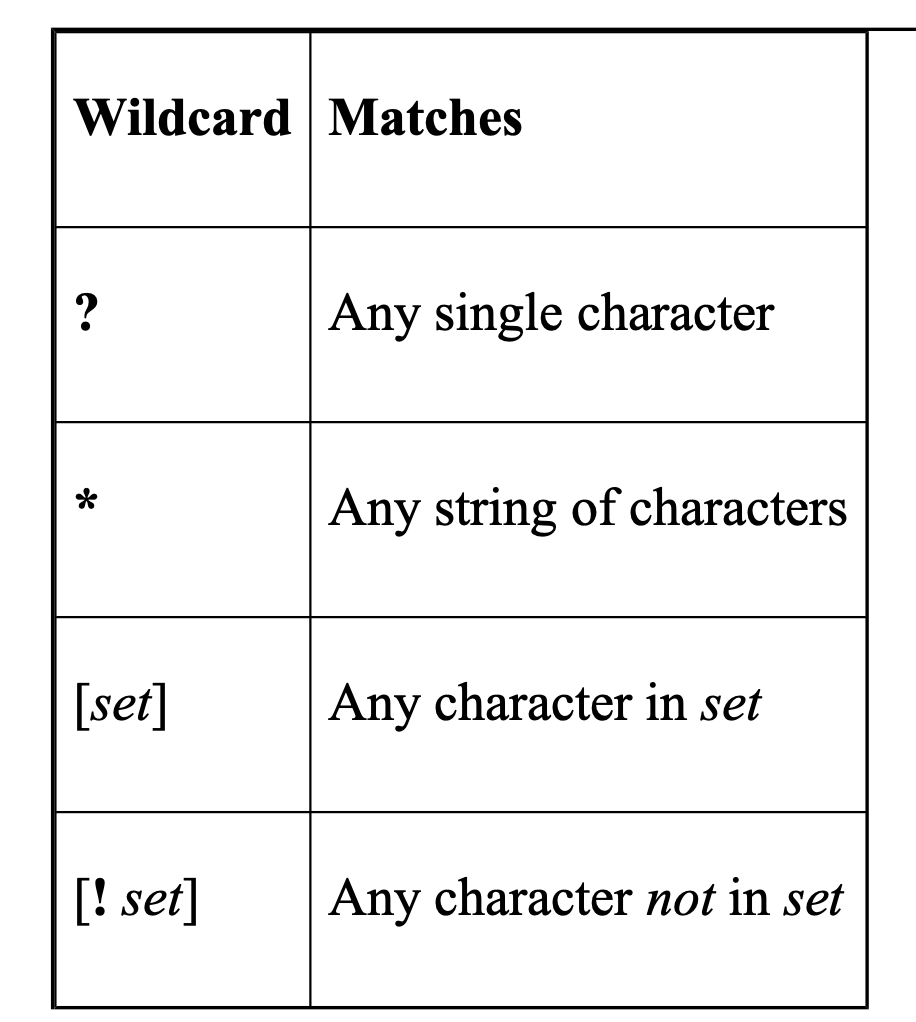
* The fact that directories can contain other directories leads to a hierarchical structure, more popularly known as a tree.
* The top of the tree is a directory called root that has no name on the system.[6] All files can be named by expressing their location on the system relative to root; such names are built by listing all of the directory names (in order from root), separated by slashes (/), followed by the file's name. This way of naming files is called a full (or absolute) pathname.
* A better way is to write the path relative to the working directory. If you give a pathname with no leading slash, then the location of the file is worked out relative to the working directory. Such pathnames are called relative pathnames.
* When you log in to the system, your working directory is initially set to a special directory called your home (or login) directory.
* Tilde by itself refers to your own home directory. cd~ takes the user to his home directory.
* The cd command is used to change the working directory. The command pwd tells the shell to print the working directory. Simply typing cd takes you to your home directory. cd - takes you to the previous working directory.
* Every directory contains two special directories i.e. “.” & “..” which is the directory itself and its parent directory respectively.

*The tree command’s output*

*Output of pwd and cd commands*

Filenames, Wildcards, and Pathname Expansion

* ls command lists information about files in the working directory, it lists all filenames except special hidden files, whose names begin with a dot. To list these files -a argument must be used. The -l (long) option, tells ls to list the file's owner, size, time of last modification, and other such information.
* Filenames are so important in UNIX that the shell provides a built-in way to specify the pattern of a set of filenames without having to know all of the names themselves by using special characters known as wildcards.

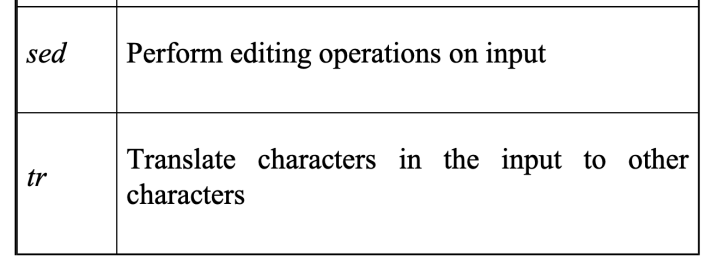
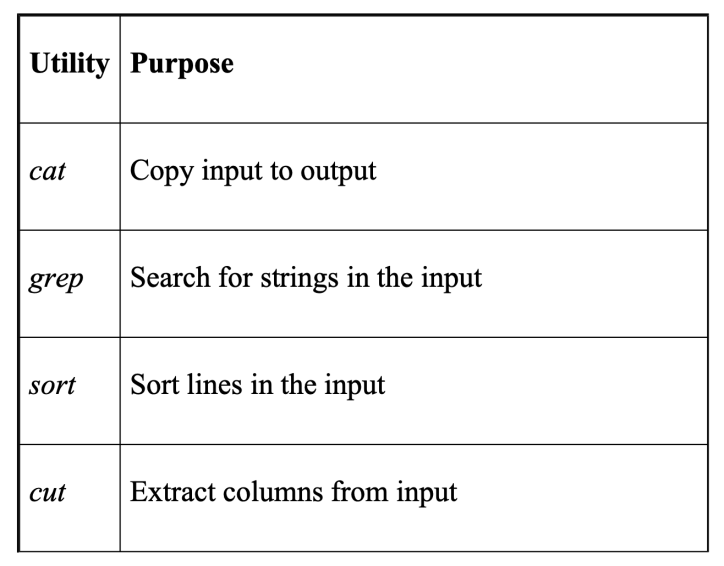


*A list of wildcards*

* The ? wildcard matches any single character. The “\*” matches zero or more characters.
* [abc] matches any one character from the set.
* [a-z] matches any character in the inclusive range.
* [!abc] matches any character not in the set. An exclamation point after the left bracket lets you "negate" a set.
* To match ! or - , preceed it with a backslash.
* The process of matching expressions containing wildcards to filenames is called wildcard expansion or globbing.
* If you type ls g\*, then (because there is no match) ls will be given the literal string g\* and will complain with the error message, g\*: No such file or directory.
* The wildcard expansion can also be used within pathnames (pathname expansion). For example, ls /usr\*/[be]\*.

Input and output

* By convention, each UNIX program has a single way of accepting input called standard input, a single way of producing output called standard output, and a single way of producing error messages called standard error output.
* You can also hook programs together in a pipeline, in which the standard output of one program feeds directly into the standard input of another.



*A list of popular UNIX data filtering utilities*

I/O redirection

* The shell lets you redirect standard input so that it comes from a file. The notation command < filename does this. Outputs can be redirected similarly using > instead.
* It is also possible to redirect the output of a command into the standard input of another command instead of a file. The construct that does this is called the pipe, notated as |. An example for this is “sort < cheshire | lp”.
* The cut command extracts the first field (-f1), where fields are separated by colons (-d:), from the input.

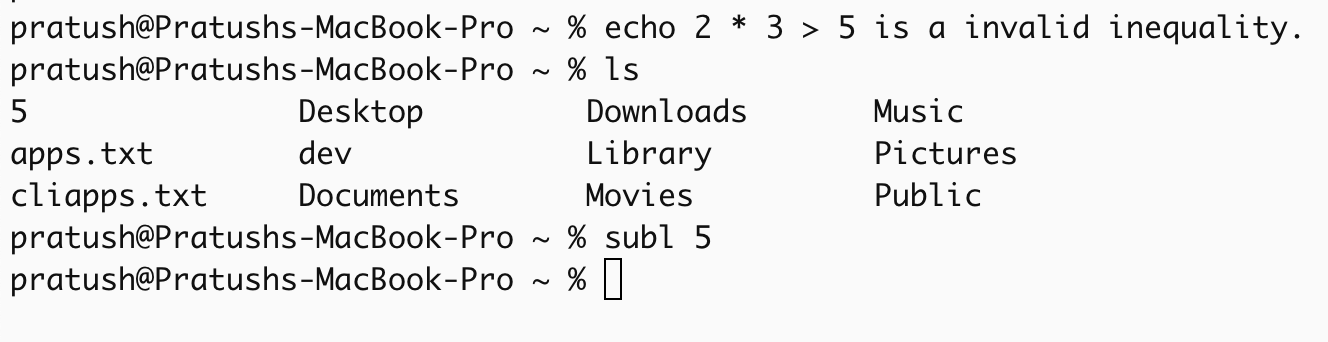
May 28, 2025

Background Jobs

* The shell also lets you run more than one command at a time during a single login session.
* But if you want to run a command that does not require user input and you want to do other things while the command is running, put an ampersand (&) after the command.
* Background jobs should not interact with the terminal's input/output directly.
* Background jobs don't control the terminal and if a background job writes to the terminal while another job is running in the foreground, the outputs will intermix and become unreadable.
* Usually redirecting Input/Output for background jobs to files is better.
* The command “nice” lets you lower the priority of any job.

Special Characters and Quoting

* Using special characters literally, i.e., without their special meanings is called quoting.
* echo is a way of making the result of that processing available on the standard output.



 *Result of echo 2 \* 3 > 5 is a invalid inequality.*



*Result of the same command but with quotations.*

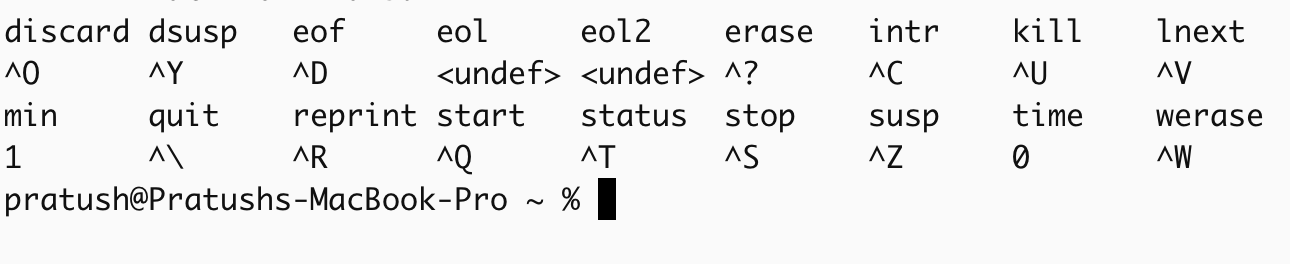
* Another way to change the meaning of a character is to precede it with a backslash (\). This is called backslash-escaping the character.
* To use a literal backslash, just surround it with quotes ('\') or, backslash-escape it (\\).
* You can also use a backslash to include double quotes within a quoted string.

Continuing Lines

* To continue a line just quote the RETURN key. After all, RETURN is really just another character. This can be achieved by by ending a line with a backslash, or by not closing a quote mark.

Control Keys

* CTRL key and hitting another key—are another type of special character.
* RETURN is actually the same as CTRL-M.
* DEL and CTRL-? are the same character.
* The stty command loads your settings. Type stty all to see your control keysettings.

 *Output of stty all command*

Help

* The help command gives information on commands in bash.
* If you type help by itself, you'll get a list of the built-in shell commands along with their options.
* help with a shell command name it will give you a detailed description of the command.
* Running help with a partial name, will return details on all commands matching the partial name.