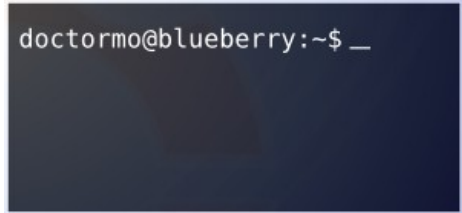


# Command Line Basics – Overview Sheet

By Martin Owens

## What is the Command Line

- Looks like this --->
- Uses the keyboard only .
- Runs all available programs .
- Access to the low levels of the computer.



```
doctormo@blueberry:~$ _
```

## Why is the Command Line Important

- Powerful set of tools
- Standard for administrators
- Quicker co-ordination
- Can be automated with scripting

## Programs

A program is a file you can run in order to achieve results. Running a program is also known as executing it and so long as you have permission to execute a file the computer will attempt to when asked. **Example Programs:**

- /bin/hostname
- /bin/ls
- /usr/bin/wget

A program has optional inputs which are specified on the command line as arguments. These change the behaviour of the program.

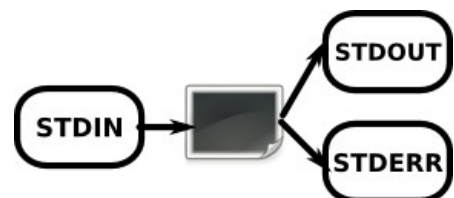
```
program -o --option file.txt
```

## Commands

A Command is a user entered program with options and is typed into a Command Line. **Example Commands:**

- hostname -d -v
- ls -l ~/
- wget online\_file.txt

A Command has one input and two outputs known as standard in out and error. This allows processing data to flow into and out of your commands, as well as errors.



## Processes

A Process is a command while it is running. It can be signalled to exit from outside.

## Shell Process

Running commands requires a shell, this program interprets what you type into something the computer understands. We will use Bash since it is the most common and easy to use, but be aware that there are other shells with different ways of writing commands.

When you run a command, it runs under the shell process as a child, if the shell dies or quits then the child process will also quit.

Commands will also inherit the same user and group as the shell it's running in. Thus it inherits the same authority and security as your user.



```
graph TD; Parent[Bash : doctormo] --> Child[My Command : doctormo];
```

**My  
Command :  
doctormo**

## Command Interpretation

The Bash command line parses the input after you've hit enter and passes a filtered version of the results. The two most common input interpretations are file expansion and variable replacement.

### File Expansion

When you need to specify multiple files with a pattern instead of manually specifying all files. For instance if you specify \*.txt then all files ending in ".txt" will be expanded to fill in arguments into the command. It is *very* important to remember that it is Bash that expands the files and not the executing program, for example:

**We type in:** cp -v \*.txt image1.png ~/

**Bash Executes:** cp -v file1.txt file2.txt file3.txt image1.png ~/

### Variable replacement

When you specify a word with a \$ prefix the named variable is replaced with the value of the variable in the environment. The environment can be listed and set with the `env` command. Examples: \$USER is always set to the current logged on user and \$PWD is always set to the current working directory. These variables can be used in more advanced commands such as this:

env	- List all current environment variables
env MYVAR="banana"	- Set a new environment variable
mv image1.png \$MYVAR.png	- Move image1.png to banana.png

## Practical Command Line

To run a program, you type in the program name followed by any options and then press enter to execute it:

```
dmo@berry:~$ ls <enter>
dmo@berry:~$ date
Mon Jul 13 17:13:30 EDT 2009
```

To specify options you add combinations of single dash and double dash arguments as well as any required inputs separated by spaces, if an argument requires spaces then precede it with a backslash '\' or encase the entire phrase in quotes:

dmo@berry:~\$ mv -v file1 file2	- Move one file to another file (renames file) and be verbose
dmo@berry:~\$ cp -R dir1 dir2	- Copy directory and all contents recursively to directory 2
dmo@berry:~\$ apropos "search term"	- Searches the manual index for functionality

## Finding a Program

To find a program you may not already know about you can use a program called apropos, you will find more programs that just a single entry for most basic searches, so look through the resulting list for the one you want:

```
dmo@berry:~$ apropos download
wget (1) - Network Downloader
hp-firmware (1) - Firmware Download
dmo@berry:~$ apropos "copy file"
cp (1) - copy files and directories
```

## Learning how to use a Program

Once you've found a program you want to learn how to use the program, use the man (manual) documentation, also use --help option which can also document the program:

dmo@berry:~\$ cp --help	- Programs internal help
dmo@berry:~\$ man wget	- System manual page for this program

## Moving Data from one Program to Another

Data printed out from a process can be directed into a following command using a pipe. It can also be directed into a file:

```
dmo@berry:~$ ls | sort > list.txt          - List directory contents, sort them and save to list.txt
dmo@berry:~$ cat file2.txt >> a.txt      - Append the contents of file2.txt to a .txt
dmo@berry:~$ date >> dates.txt          - Append the current date/time to a file.
```

## Running as another User

You can change the user that a command will run under by using sudo (super user do) which by default will run the command as root, but can also run programs as other users:

```
dmo@berry:~$ sudo ls /etc                - List the contents of a directory as if I were root
dmo@berry:~$ sudo apt-get install        - Install a program on the debian computer.
```

## Scripting

Scripting is very easy, simply create a file where each line is a separate command that you would type on the command line. Specifying the shell to use at the top in a bang line:

```
dmo@berry:~$ echo "#!/bin/bash
date >> dates.txt" > store_date.sh      - Store our script contents in a file.
dmo@berry:~$ chmod 755 store_date.sh   - Give ourselves permission to execute this script as a program
dmo@berry:~$ ./store_date.sh           - Execute script in current directory.
```