Command Line Manual

For Unix-based systems

José Fernando Mendes da Silva Costa

Contents

Introduction

The command line is a text interface for a computer. It's a program that takes in commands, which it passes on to the computer's operating system to run.

From the command line, you can navigate through files and folders on a computer, just as you would with Finder on Mac OS or Windows Explorer on Windows. The difference is that the command line is fully text-based.

The advantage of using the command line is its power. You can run programs, write scripts to automate common tasks, and combine simple commands to handle difficult tasks - making it an important programming tool.

For the rest of this document, I will be focusing on the command line used for unix-based systems, such as Linux and Mac OS X.

I would also like to say this document is a product of what I have learned by completing Codecademy's *Learn the Command Line* course and information I found on the internet.

Glossary

Argument

When a file, directory or program is passed into a command, it is called an argument.

Bash profile

~/.bash_profile is the name of file used to store environment settings. It is commonly called the "bash profile". When a session starts, it will load the contents of the bash profile before executing commands.

The `~` represents the user's home directory.

The `.` indicates a hidden file.

The name `~/.bash_profile` is important, since this is how the command line recognizes the bash profile.

Command

A *command* is a directive to the computer to perform a specific task.

In the terminal, first you see `$`. This is called a shell prompt. It appears when the terminal is ready to accept a *command*.

Environment

Each time we launch the terminal application, it creates a new session. The session immediately loads settings and preferences that make up the command line *environment*.

We can configure the environment to support the commands and programs we create. This enables us to customize greetings and command aliases, and create variables to share across commands and programs.

Environment variables

Environment variables are variables that can be used across commands and programs and hold information about the environment.

Due note a `$` is always used when returning a variable's value. For example, to output the value of the `USER` variable using the `echo` command, it should look like this `echo $USER`.

Filesystem

A filesystem organizes a computer's files and directories into a tree structure:

The first directory in the filesystem is the root directory. It is the parent of all other directories and files in the filesystem.

Each parent directory can contain more child directories and files.

Each directory can contain more files and child directories. The parent-child relationship continues as long as directories and files are nested.

Options

Options modify the behavior of commands. `-a`, `-l`, and `-t` are some examples of options that can be used along with the `ls` command.

Redirection

Through *redirection* you can direct the input and output of a command to and from other files and programs, and chain commands together in a pipeline.

Redirection reroutes *standard input*, *standard output*, and *standard error* to or from a different location.

Standard error

Standard error, abbreviated as `stderr`, is an error message outputted by a failed process.

Standard input

Standard input, abbreviated as `stdin`, is information inputted into the terminal through the keyboard or input device.

Standard output

Standard output, abbreviated as `stdout`, is the information outputted after a process is run.

Wildcards

In addition to using filenames as arguments, we can use special characters like `*` to select groups of files. These special characters are called *wildcards*.

An example for the `*` wildcard, using `cp m*.txt scifi/` would copy all files in the working directory starting with ‘m’ and ending in ’.txt.’ to the scifi/ directory.

Commands

As explained before, in the terminal, first you see `$`. This is called a shell prompt and it appears when the terminal is ready to accept a *command*.

A *command* is a directive to the computer to perform a specific task.

Now let’s explore some of the available commands in the command line.

ls

When you type `ls`, the command line looks at the folder you are in, and then "lists" the files and folders inside it.

When using the command line, we refer to folders as directories. Files and directories on your computer are organized into a *filesystem*.

`ls` can also be used with *options*. Options modify the behavior of commands. There are three common options used with `ls`:

- `-a`: lists all contents, including hidden files and directories;
- `-l`: lists all contents of a directory in long format. The columns returned by using this option have the following order and meaning:
    - Access rights: these are actions that are permitted on a file or directory;
    - Number of hard links: this number counts the number of child directories and files. This number includes the parent directory link (`..`) and current directory link (`.`);
    - The username of the file's owner;
    - The name of the group that owns the file;
    - The size of the file in bytes;
    - The date & time that the file was last modified;
    - The name of the file or directory.
- `-t`: order files and directories by the time they were last modified.

Besides using each of the above options separately, it is also possible to simply use them all at the same. This would result in `-alt`, which would have the qualities of `-a`, `-l`, and `-t`, except combined: lists all contents, including hidden files and directories, in long format, ordered by the date and time they were last modified.

pwd

`pwd` stands for "print working directory". It outputs the name of the directory you are currently in, called the *working directory*.

Together with `ls`, the `pwd` command is useful to show where you are in the filesystem.

cd

`cd` stands for "change directory". Just as you would click on a folder in Windows Explorer or Finder, `cd` switches you into the directory you specify. In other words, `cd` changes the working directory.

The `cd` command takes a directory name as an argument, and switches into that directory.

To navigate directly to a directory, use `cd` with the directory's path as an argument.

To move up one directory, use `cd ..` .

mkdir

The `mkdir` command stands for "make directory".

It takes in a directory name as an argument, and then creates a new directory in the current working directory.

touch

The `touch` command creates a new file inside the working directory. It takes in a filename as an argument, and then creates an empty file in the current working directory.

cp

The `cp` command copies files or directories.

To copy a file into a directory, use `cp` with the source file as the first argument and the destination directory as the second argument.

To copy multiple files into a directory, use `cp` with a list of source files as the first arguments, and the destination directory as the last argument.

mv

The `mv` command moves files. It's similar to `cp` in its usage: to move a file into a directory, use `mv` with the source file as the first argument and the destination directory as the second argument.

To move multiple files into a directory, use `mv` with a list of source files as the first arguments, and the destination directory as the last argument.

To rename a file, use `mv` with the old file as the first argument and the new file as the second argument.

rm

The `rm` command deletes files and directories.

`-r` is an option that modifies the behavior of the `rm` command. The `-r` stands for "recursive," and it's used to delete a directory and all of its child directories.

Be careful when you use `rm`! It deletes files and directories permanently. There isn't an undelete command, so once you delete a file or directory with `rm`, it's gone.

echo

The `echo` command accepts a string as *standard input* and *echoes* it back to the terminal as *standard output*.

>

The `>` command redirects the *standard output* to a file.

Note that `>` overwrites all original content in the receiving file.

cat

The `cat` command outputs the contents of a file to the terminal.

>>

`>>` takes the *standard output* of the command on the left and appends (adds) it to the file on the right.

<

`<` takes the *standard input* from the file on the right and inputs it into the program on the left.

|

`|` is a "pipe". The `|` takes the standard output of the command on the left, and pipes it as standard input to the command on the right. You can think of this as "command to command" redirection.

Multiple `|`s can be chained together.

wc

The `wc` command outputs the number of lines, words, and characters in the file used as argument.

wc can use the following options:

- `-l` to output only the number of lines in the file;
- `-w` to output only the number of words in the file;
- `-c` to output the number of bytes in the file;
- `-m` to output the number of characters in the file;
- `-L` to output the length (number of characters) of the longest line in the file

sort

`sort` takes the *standard input* and orders it alphabetically for the *standard output*.

uniq

`uniq` stands for "unique" and filters out adjacent, duplicate lines in a file.

A more effective way to call `uniq` is to call `sort` to alphabetize a file, and "pipe" the *standard output* to `uniq`. By piping a sorted file to `uniq`, all duplicate lines are alphabetized (and thereby made adjacent) and filtered out.

grep

`grep` stands for "global regular expression print". It searches files for lines that match a pattern and returns the results. Due note `grep` is also case sensitive.

Using `grep` with the `-i` option enables the command to be case insensitive.

The `-R` option searches all files in a directory and outputs filenames and lines containing matched results. `-R` stands for "recursive".

The `-Rl` option searches all files in a directory and outputs only filenames with matched results. `-R` stands for "recursive" and `l` stands for "files with matches".

sed

`sed` stands for "stream editor". It accepts *standard input* and modifies it based on an *expression*, before displaying it as output data. It is similar to "find and replace".

The expression can look like this: ‘*`s/search/replace`*’. The ‘*`s`*’ in the expression stands for "substitution". It is always used when using `sed` for substitution. ‘*`search`*’ is the string to be searched for in the standard input. ‘*`replace`*’ is the replacement string, that is, the text to add in place of ‘*`search`*’. Due note this command only replaces the first instance of ‘*`search`*’ in each line.

If you add a *`/g`* at the end of *expression*, `sed` searches for ‘*`search`*’ globally, which means all instances of ‘*`search`*’ are replaced with ‘*`replace`*’.

clear

Clears the terminal window, moving the command prompt to the top of the screen.

source

The command `source` activates the changes in a file for the current session. Instead of closing the terminal and needing to start a new session, `source` makes the changes available right away in the current session.

alias

The `alias` command allows you to create keyboard shortcuts, or aliases, for commonly used commands.

For example, `alias pd="pwd"` creates the alias *pd* for the `pwd` command, which is then saved in the bash profile. Each time you enter *pd*, the output will be the same as the `pwd` command. This means each time we open up the terminal, we can use the *pd* alias.

history

This command outputs all the commands that were entered during the current session.

export

The export command makes a variable available to all child sessions initiated from the session you are in. This is a way to make the variable persist across programs.

Put other way, `export VARIABLE="Value"` sets and exports an environment variable.

env

The env command stands for "environment", and returns a list of the environment variables for the current user.

Wildcards

*

The `*` wildcard selects all files in the working directory.

For example, using `cp m*.txt scifi/` would copy all files in the working directory starting with 'm' and ending in '.txt.' to the scifi/ directory. Between 'm' and '.txt' can be any number of characters, if there's a file a 'ma.txt' file and a 'maaaaabbbbbb.txt' file, both will be copied.

?

The `?` wildcard selects all matches a single alphabet in a specific position.

For example, using `cp b?ll.txt scifi/` would copy all files in the working directory that start with 'b', end in 'll.txt' and in between have 1 single character to the scifi/ directory. So, in this case, we'd copy files such as 'ball.txt', 'bell.txt' and 'bill.txt'. It doesn't matter what character is in that position, as long as the name has the same length as the name you are searching for.

Command line text editor

nano

`nano` is a command line text editor. It works just like a desktop text editor like TextEdit or Notepad, except that it is accessible from the command line and only accepts keyboard input.

The command `nano argument.txt` opens a new text file named *argument.txt* in the `nano` text editor. Then you can input text into the terminal.

After opening the text editor there are some keyboard shortcuts you can use. Some of them are *Ctrl + O* to save the file (Output), *Ctrl + X* to exit nano (eXit), and *Ctrl + G* opens a help menu.

Environment variables

Environment variables are variables that can be used across commands and programs and hold information about the *environment*.

USER

`USER` is usually set to the name of the computer's owner.

PS1

`PS1` is a variable that defines the makeup and style of the command prompt. The default for the command prompt is `$`.

HOME

The `HOME` variable is an environment variable that displays the path of the home directory.

PATH

`PATH` is an environment variable that stores a list of directories separated by a colon (`,`).

Each directory contains scripts for the command line to execute. The `PATH` variable simply lists which directories contain scripts.

DOS commands

This last section focuses in providing the Windows DOS versions of the UNIX commands presented in this document.

Description	Unix version	DOS version
List files	`ls`	`dir`
Output your location in the filesystem	`pwd`	`chdir`
Change directories to a specified path	`cd`	`cd`
Create a directory	`mkdir`	`mkdir`
Create a file	`touch`	`echo`
Copy a file	`cp`	`copy`
Move a file	`mv`	`move`
Delete a file	`rm`	`del`
"Echo" a *standard input* as *standard output* to the terminal	`echo`	`echo`
Redirect a standard output from a source to a file	`>`	`>`
Output the contents of a file	`cat`	`Type`
Append the content of a command to a file	`>>`	`>>`
Takes the *standard input* from the file on the right and inputs it into the program on the left	`<`	`<`
Takes the standard output of the command on the left, and redirects it as standard input to the command on the right	`|`	`|`
Outputs the number of lines, words, and characters in a file	`wc`	

Sort the *standard input* alphabetically	`sort`	`sort`
Filters out adjacent, duplicate lines in a file	`uniq`	
Finds a string of text in a file	`grep`	`find`
Searches a file for a specified string and replaces it for another specified string	`sed`	
Clears the screen	`clear`	`clear`
Activates the changes made in a file in the current session, without needing to restart the terminal to activate them	`source`	
Allows the creation of keyboard shortcuts or aliases for commands	`alias`	
Outputs all the commands that were entered during the current session	`history`	
Makes a variable available to all child sessions initiated from the current session	`export`	
Returns a list of the *environment variables* for the current user	`env`	`set`