Linux and Command Prompt

Basics and How-To

What's a Command Prompt?

1. Also called Terminal or Bash and is used to execute commands.
2. It is typically a black screen with white font and has a very simple User Interface. You type in commands, and it interprets these commands and runs operations in the computer

SSH and How to do it?

1. SSH or "Secure Shell" is essentially a magic way to safely access unsafe networks. This is commonly used to remotely login to a computer and execute commands.
2. To use SSH to remote login:
    a. On Windows: Use an SSH client like PuTTY.
    b. On Mac or Linux: In terminal, type "*ssh {user}@{host}*". User refers to the account you want to access and host is the domain or IP address of the computer you are trying to access.
3. For a more in-depth understanding of SSH, we recommend this article.

The Swiss Army Knife: Netcat!

1. Netcat is a tool that can help you read or write data over the internet and is called "The Swiss Army Knife of Information Security" by its fans.
2. It earned its nickname because one can use netcat to perform a lot of different tasks including file transfer, chatting, port scanning and can even serve as both a client and a server.
3. The basic syntax for netcat commands is "*nc [options] [destination] [port]*". Here,
    a. *Options* is an optional argument or "flag" that you can use to change the behavior of netcat. For example "nc -h" prints helpful information about nc.
    b. *Destination*, is the IP address of the computer you are trying to contact.
    c. *Port* is the endpoint and helps identify the type of communication happening
4. There are many different uses of netcat and in general, you can get all the information by typing in "*nc -h*" into your terminal window! We also

recommend this article to understand how to use netcat. This other article contains a great list of all the *option flags* (look at the "Netcat Command Flags" box).

Fun With Terminal

1. Terminal has a bunch of useful commands that span a wide range of functionalities. You can navigate between folders (aka directories) using *cd* ("change directory"). You can test your net connection(s). You can create, delete or even edit files from within Terminal! Here is a great list of everyday (beginner) commands and even some more nuanced ones to help you do some really cool things.
2. **Warning!** Be careful with the *rm* ("remove") command. If you execute "*rm -rf* " into your terminal, you can delete all your files! So be careful in the terminal. You have a lot of power (and a lot of responsibility) so make sure you use the correct command and don't be afraid to use your favorite search engine (Google, Bing, DuckDuckGo, etc.) to search for something if you are unsure!
3. **Command Line Text Editor?** One particular section that might confuse some of you from the article mentioned above is "Intermediate Command 3: nano, vi...". If you know what these are, awesome; you can begin your journey doing everything in terminal. However, if you think nano is an old iPod, then you should reference the below section on text editors. For now, it is a text editor like Notepad that you can access from your terminal.
4. **Root Privileges!** Another important command that usually confuses people is "*sudo"* or *"SuperUserDO*". Essentially, this command allows you to have root privileges in your terminal. This is very risky as the root user has absolute power over the system and you can essentially do anything and possibly cause a lot of harm. But you can also do a lot of cool things so if you're using sudo, make sure you know what you are doing and that you're careful.
5. We explain some more important commands at the end of this document.

Who needs a Text Editor when you got Terminal?

1. There are many terminal text editors out there: *Nano, Vim, Emacs* and many more. They all have similar functionality but each programmer has their own

reasons why one is clearly superior to the other. So, what is your choice of editor?

2. Executing in *nano <filename.txt>* or *vim <filename.txt>* in your terminal. If a file with the same name exists, you will open it and if it doesn't, it will create a new file.
3. Here is a basic guide to using *GNU Nano* in the terminal!

Numbers, Numbers, Numbers!

1. You are most probably familiar with the Base-10 Decimal system! However, computer understand 1's and 0's or Binary (aka Base-2)! There are also other systems for example, Base-8 or Octal and Base-16 or Hexadecimal. In general, a Base-N system has digits from 0 to N-1. For example, Base-2 consists of 0 and 1 as digits.

Let's do an in depth analysis of some of the most important number systems.

Binary (Base -2)

1. *Digits*: 0, 1
2. *To convert Binary to Decimal*:
    a. Multiply each digit with 2^(position of that number)
    b. Add the above for all digits
    c. For example: $(10011)_2 = (1*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0)_{10} = (19)_{10}$
3. *To convert Decimal to Binary*:
    a. Divide the number by 2
    b. Keep track of the remainder
    c. Divide the quotient by 2 and keep repeating till you get a quotient of 0 while keeping track of remainders
    d. Read off the remainders in reverse order and that is your number in Binary
    e. For example: $(6)_{10} = 2(3) + 0$, $(3)_{10} = 2(1) + 1$, $(1)_{10} = 2(0) + 1$. Therefore, $(6)_{10} = (110)_2$
4. *Addition:*
    a. In binary, there are 4 possibilities when adding 1 bit to another.
        i. 0 + 0 = 0 with a carry of 0
        ii. 0 + 1 = 1 with a carry of 0
        iii. 1 + 0 = 1 with a carry of 0
        iv. 1 + 1 = 0 with a carry of 1
    b. Addition is commutative and works exactly the same way as in decimal.

5. *Two's Complement Representation*: This is a form of representing Binary numbers which makes it very easy to store both positive and negative numbers and to find the binary representation of the negation of a given number.
    a. The leftmost bit (most significant bit) is the "sign bit" which means that it tells you whether the number is positive or negative. '1' means that the number is negative and '0' means it is positive.
    b. The rest of the digits represent the absolute value of the number in the normal Base 2 way.
    c. This means that given n-bits to represent a number, we can only use n-1 bits to represent the number. This leads to a largest positive number of $2^{(n-1)} - 1$ and smallest negative number as $-2^{(n-1)}$. Can you prove this to yourself? Try a few examples.
    d. Now how do you negate a number?
        i. Take the original two's complement representation of the number and flip all the bits (1 becomes 0 and vice versa)
        ii. Add 1 to this new number and this leads to the negative of the original number.
    e. To convert a two's complement number to decimal, you first check the leftmost bit to see if it is positive or negative. If it is positive, you convert to decimal as normal. If it is negative, negate the number, convert to decimal as normal, then add the negative sign back in front of it.
    f. $(4)_{10} = (0100)_2$ in Two's Complement form. Now let's apply our little algorithm! Flipping bits yields $(1011)_2$ and when you add 1, you get $(1100)_2$. Convert it to decimal and you will see that 1100 is the two's complement form of -4.
    g. Arithmetic works pretty much the same as normal binary numbers.

Little Endian and Big Endian!

These are ways to store numbers or data in memory addresses. Let's use a 16-bit word as example[1], $(0xFEED)_{16}$ in this case. Let's also assume we are storing this word starting at address 0x4000. We store these words in terms of bytes and not bits, so we need some conversion. Recall that 16 bits is 2 bytes (since 1 byte is 8 bits). The word is stored in pairs to make up the required 1 byte per memory location so our two parts will be 'FE' and 'ED'.

[1] In this case, 'word' means any number in Base-16 or Hexadecimal. So, a 16-bit word is a hexadecimal number with 16 bits or 4 values.

1. *Big Endian*: This refers to big end first, which means that we store the most significant byte at the smallest memory location and the rest follow normally. Therefore, to store our word in this case, memory location 0x4000 will have the byte *FE* and memory location 0x4001 will store the byte *ED*.
2. *Little Endian*: This refers to little end first and opposed to Big Endian, we store the least significant bit at the smallest memory address and the rest follow normally with the most significant bit at the last memory address. So, 0x4000 will have the byte *ED* and 0x4001 will have *FE*.

Big Endian is commonly used in Networking application, while Little Endian is most commonly used in processors.