UNIX - QUICK GUIDE

GETTING STARTED:

What is Unix ?

The UNIX operating system is a set of programs that act as a link between the computer and the user.

- Unix was originally developed in 1969 by a group of AT&T employees at Bell Labs, including Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna.
- There are various Unix variants available in the market. Solaris Unix, AIX, HP Unix and BSD are few examples. Linux is also a flavor of Unix which is freely available.
- Several people can use a UNIX computer at the same time; hence UNIX is called a multiuser system.
- A user can also run multiple programs at the same time; hence UNIX is called multitasking.

Login Unix:

You can log in to the system using **login** command as follows:

```
login : amrood
amrood's password:
Last login: Sun Jun 14 09:32:32 2009 from 62.61.164.73
$
```

Logging Out:

When you finish your session, you need to log out of the system to ensure that nobody else accesses your files while masquerading as you.

To log out:

1. Just type **logout** command at command prompt, and the system will clean up everything and break the connection

FILE MANAGEMENT:

In UNIX there are three basic types of files:

1. **Ordinary Files:** An ordinary file is a file on the system that contains data, text, or program instructions. In this tutorial, you look at working with ordinary files.
2. **Directories:** Directories store both special and ordinary files. For users familiar with Windows or Mac OS, UNIX directories are equivalent to folders.
3. **Special Files:** Some special files provide access to hardware such as hard drives, CD-ROM drives, modems, and Ethernet adapters. Other special files are similar to aliases or shortcuts and enable you to access a single file using different names.

Filename Substitution:

Command	Description
ls -[l]	List Files in Current Directory
ls -[l]a	List Hidden Files
~	Home Directory

~user	Home Directory of Another User
?	Wild Card, matches single character
*	Wild Card, matches multiple characters

Filename Manipulation:

Command	Description
cat filename	Display File Contents
cp source destination	Copy source file into destination
mv oldname newname	Move (Rename) a oldname to newname.
rm filename	Remove (Delete) filename
chmod nnn filename	Changing Permissions
touch filename	Changing Modification Time
ln [-s] oldname newname	Creates softlink on oldname
ls -F	Display information about file type.

DIRECTORY MANAGEMENT:

Command	Description
mkdir dirname	Create a new directory dirname
rmdir dirname	Delete an existing directory provided it is empty.
cd dirname	Change Directory to dirname
cd -	Change to last working directory.
cd ~	Change to home directory
pwd	Display current working directory.

ENVIRONMENT SETUP:

When you type any command on command prompt, the shell has to locate the command before it can be executed. The **PATH** variable specifies the locations in which the shell should look for commands.

PS1 and PS2 Variables:

The characters that the shell displays as your command prompt are stored in the variable PS1.

When you issue a command that is incomplete, the shell will display a secondary prompt and wait for you to complete the command and hit Enter again. The default secondary prompt is > (the greater than sign), but can be changed by re-defining the PS2 shell variable:

Escape Characters:

Escape Sequence	Description
\t	Current time, expressed as HH:MM:SS.
\d	Current date, expressed as Weekday Month Date
\n	Newline.
\s	Current shell environment.
\W	Working directory.
\w	Full path of the working directory.
\u	Current user.s username.
\h	Hostname of the current machine.
\#	Command number of the current command. Increases with each new command entered.
\$	If the effective UID is 0 (that is, if you are logged in as root), end the prompt with the # character; otherwise, use the $.

Environment Variables:

Following is the partial list of important environment variables. These variables would be set and accessed as mentioned above:

Variable	Description
DISPLAY	Contains the identifier for the display that X11 programs should use by default.
HOME	Indicates the home directory of the current user: the default argument for the cd built-in command.
IFS	Indicates the Internal Field Separator that is used by the parser for word splitting after expansion.
LANG	LANG expands to the default system locale; LC_ALL can be used to override this. For example, if its value is pt_BR, then the language is set to (Brazilian) Portuguese and the locale to Brazil.
LD_LIBRARY_PATH	On many Unix systems with a dynamic linker, contains a colon-separated list of directories that the dynamic linker should search for shared objects when building a process image after exec, before searching in any other directories.
PATH	Indicates search path for commands. It is a colon-separated list of directories in which the shell looks for commands.
PWD	Indicates the current working directory as set by the cd command.
RANDOM	Generates a random integer between 0 and 32,767 each time it is referenced.
SHLVL	Increments by one each time an instance of bash is started. This variable is useful for determining whether the built-in exit command ends the current session.
TERM	Refers to the display type

TZ	Refers to Time zone. It can take values like GMT, AST, etc.
UID	Expands to the numeric user ID of the current user, initialized at shell startup.

FILTERS & PIPES:

Command	Description
wc [-l]	Word/Line Count
tail [-n]	Displays last n lines from a file
sort [-n]	Sort lines
pr -t	Multicolumn Output
grep "pattern" filename	Searching for a pattern with grep
pg or more	Paginate a file content display.

SPECIAL VARIABLES

Variable	Description
$0	The filename of the current script.
$n	These variables correspond to the arguments with which a script was invoked. Here n is a positive decimal number corresponding to the position of an argument (the first argument is $1, the second argument is $2, and so on).
$#	The number of arguments supplied to a script.
$*	All the arguments are double quoted. If a script receives two arguments, $* is equivalent to $1 $2.
$@	All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2.
$?	The exit status of the last command executed.
$$	The process number of the current shell. For shell scripts, this is the process ID under which they are executing.
$!	The process number of the last background command.

SHELL BASIC OPERATORS

Arithmetic Operators:

Assume variable a holds 10 and variable b holds 20 then:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	`expr $a + $b` will give 30

-	Subtraction - Subtracts right hand operand from left hand operand	`expr $a - $b` will give -10
*	Multiplication - Multiplies values on either side of the operator	`expr $a * $b` will give 200
/	Division - Divides left hand operand by right hand operand	`expr $b / $a` will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	`expr $b % $a` will give 0
=	Assignment - Assign right operand in left operand	a=$b would assign value of b into a
==	Equality - Compares two numbers, if both are same then returns true.	[$a == $b] would return false.
!=	Not Equality - Compares two numbers, if both are different then returns true.	[$a != $b] would return true.

Relational Operators:

Assume variable a holds 10 and variable b holds 20 then:

Operator	Description	Example
-eq	Checks if the value of two operands are equal or not, if yes then condition becomes true.	[$a -eq $b] is not true.
-ne	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	[$a -ne $b] is true.
-gt	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	[$a -gt $b] is not true.
-lt	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	[$a -lt $b] is true.
-ge	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	[$a -ge $b] is not true.
-le	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	[$a -le $b] is true.

Boolean Operators:

Assume variable a holds 10 and variable b holds 20 then:

Operator	Description	Example
!	This is logical negation. This inverts a true condition into false and vice versa.	[! false] is true.
-o	This is logical OR. If one of the operands is true then	[$a -lt 20 -o $b -gt 100] is true.

	condition would be true.	
-a	This is logical AND. If both the operands are true then condition would be true otherwise it would be false.	[$a -lt 20 -a $b -gt 100] is false.

String Operators:

Assume variable a holds "abc" and variable b holds "efg" then:

Operator	Description	Example
=	Checks if the value of two operands are equal or not, if yes then condition becomes true.	[$a = $b] is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	[$a != $b] is true.
-z	Checks if the given string operand size is zero. If it is zero length then it returns true.	[-z $a] is not true.
-n	Checks if the given string operand size is non-zero. If it is non-zero length then it returns true.	[-z $a] is not false.
str	Check if str is not the empty string. If it is empty then it returns false.	[$a] is not false.

File Test Operators:

Assume a variable **file** holds an existing file name "test" whose size is 100 bytes and has read, write and execute permission on:

Operator	Description	Example
-b file	Checks if file is a block special file if yes then condition becomes true.	[-b $file] is false.
-c file	Checks if file is a character special file if yes then condition becomes true.	[-c $file] is false.
-d file	Check if file is a directory if yes then condition becomes true.	[-d $file] is not true.
-f file	Check if file is an ordinary file as opposed to a directory or special file if yes then condition becomes true.	[-f $file] is true.
-g file	Checks if file has its set group ID (SGID) bit set if yes then condition becomes true.	[-g $file] is false.
-k file	Checks if file has its sticky bit set if yes then condition becomes true.	[-k $file] is false.
-p file	Checks if file is a named pipe if yes then condition becomes true.	[-p $file] is false.
-t file	Checks if file descriptor is open and associated with a terminal if yes then condition becomes true.	[-t $file] is false.

-u file	Checks if file has its set user id (SUID) bit set if yes then condition becomes true.	[-u $file] is false.
-r file	Checks if file is readable if yes then condition becomes true.	[-r $file] is true.
-w file	Check if file is writable if yes then condition becomes true.	[-w $file] is true.
-x file	Check if file is execute if yes then condition becomes true.	[-x $file] is true.
-s file	Check if file has size greater than 0 if yes then condition becomes true.	[-s $file] is true.
-e file	Check if file exists. Is true even if file is a directory but exists.	[-e $file] is true.

SHELL DECISION MAKING

The if...fi statement:

```
if [ expression ]
then
   Statement(s) to be executed if expression is true
fi
```

The if...else...fi statement:

```
if [ expression ]
then
   Statement(s) to be executed if expression is true
else
   Statement(s) to be executed if expression is not true
fi
```

The if...elif...fi statement:

```
if [ expression 1 ]
then
   Statement(s) to be executed if expression 1 is true
elif [ expression 2 ]
then
   Statement(s) to be executed if expression 2 is true
elif [ expression 3 ]
then
   Statement(s) to be executed if expression 3 is true
else
   Statement(s) to be executed if no expression is true
fi
```

The case...esac Statement:

```
case word in
  pattern1)
     Statement(s) to be executed if pattern1 matches
     ;;
  pattern2)
     Statement(s) to be executed if pattern2 matches
     ;;
  pattern3)
     Statement(s) to be executed if pattern3 matches
     ;;
esac
```

SHELL LOOP TYPES:

The while Loop:

```
while command
do
   Statement(s) to be executed if command is true
done
```

The for Loop:

```
for var in word1 word2 ... wordN
do
   Statement(s) to be executed for every word.
done
```

The until Loop:

```
until command
do
   Statement(s) to be executed until command is true
done
```

The select Loop:

```
select var in word1 word2 ... wordN
do
   Statement(s) to be executed for every word.
done
```

SHELL LOOP CONTROL:

The break statement:

```
break [n]
```

The continue statement:

```
continue [n]
```

SHELL SUBSTITUTIONS:

The shell performs substitution when it encounters an expression that contains one or more special characters.

Command Substitution:

The command substitution is performed when a command is given as:

```
`command`
```

Variable Substitution:

Here is the following table for all the possible substitutions:

Form	Description
${var}	Substitue the value of *var*.
${var:-word}	If *var* is null or unset, *word* is substituted for **var**. The value of *var* does not

	change.
${var:=word}	If *var* is null or unset, *var* is set to the value of **word**.
${var:?message}	If *var* is null or unset, *message* is printed to standard error. This checks that variables are set correctly.
${var:+word}	If *var* is set, *word* is substituted for var. The value of *var* does not change.

REDIRECTION COMMANDS:

Following is the complete list of commands which you can use for redirection:

Command	Description
pgm > file	Output of pgm is redirected to file
pgm < file	Program pgm reads its input from file.
pgm >> file	Output of pgm is appended to file.
n > file	Output from stream with descriptor n redirected to file.
n >> file	Output from stream with descriptor n appended to file.
n >& m	Merge output from stream n with stream m.
n <& m	Merge input from stream n with stream m.
<< tag	Standard input comes from here through next tag at start of line.
|	Takes output from one program, or process, and sends it to another.

SHELL MAN PAGES HELP:

This quick guide lists commands, including a syntax and brief description. For more detail, use:

```
$man command
```