Bash, Scripting & Coding

- Bash
  - BASH ULTIMATE MAN PAGE
  - New Page
- Programming Constructs

Bash

Bash scripting

BASH ULTIMATE MAN PAGE

==========

```
  #!/bin/bash
```

Beginning a bash script

==========

```
echo 'enter domain name to test'
read domain
```

Set user input as a variable

==========

GREP

```
grep -i pattern filename
```

ignore character case

```
grep -v pattern filename
```

inverted match - essentially removes whatever pattern is specified.

```
grep -H pattern filenames
```

Print the file name for each match - useful for grepping patterns in multiple files. To achieve the opposite (no files names) pass -h.

```
grep -n pattern filename
```

Prefix each line of output with the 1-based line number within its input file.

Character Classes and Bracket Expressions

A bracket expression is a list of characters enclosed by [and]. It matches any single character in that list. If the first character of the list is the caret ^ then it matches any character not in the list; it is unspecified whether it matches an encoding error. For example, the regular expression [0123456789] matches any single digit.

Basic vs Extended Regular Expressions
In basic regular expressions the meta-characters ?, +, {, |, (, and) lose their special meaning; instead use the backslashed versions \?, \+, \{, \|, \(, and \).

Repetition
A regular expression may be followed by one of several repetition operators:
? The preceding item is optional and matched at most once.
* The preceding item will be matched zero or more times.
+ The preceding item will be matched one or more times.
{n} The preceding item is matched exactly n times.
{n,} The preceding item is matched n or more times.
{,m} The preceding item is matched at most m times. This is a GNU extension.
{n,m} The preceding item is matched at least n times, but not more than m times.

==========

ECHO

```
echo -e
```

allow interpretation of backslash-escaped character:

\a alert (bell)
\b backspace
\c suppress further output
\e escape character
\E escape character
\f form feed
\n new line
\r carriage return
\t horizontal tab
\v vertical tab
\\ backslash
\0nnn the character whose ASCII code is NNN (octal). NNN can be
0 to 3 octal digits
\xHH the eight-bit character whose value is HH (hexadecimal). HH
can be one or two hex digits
\uHHHH the Unicode character whose value is the hexadecimal value HHHH.
HHHH can be one to four hex digits.
\UHHHHHHHH the Unicode character whose value is the hexadecimal value
HHHHHHHH. HHHHHHHH can be one to eight hex digits.

```
echo "${x%Y}"
```

Remove a character (Y) from a string

==========

SORT

```
sort -k1
```

Sort a file or output by a specific column, ie sort -k1 would sort by the first column

---

```
sort -n
```

Numeric sort (default sort order is **ascending).**

```
sort -r
```

Reverse the output of the sort.

==========

AWK

```
awk '{print $1,$2}'
```

Print the first and second column

SEQ

(sequence)

The below shows what the seq command can be used for, creating sequences of numbers.

```
seq 10
1
2
3
4
5
6
7
8
9
10
```

seq [*OPTION*]... *LAST*
seq [*OPTION*]... *FIRST LAST*

seq [*OPTION*]... *FIRST INCREMENT LAST*

For Loops

For loops are a fundamental programming construct used to execute a block of code repeatedly. They are particularly useful when you need to process a sequence of items, like elements in a list or characters in a string.

Example:

```
#!/bin/bash
for filename in `seq 10` ##sets the variable filename as the output of seq 10
do
        touch video$filename.mp4 ##create file video$filename.mp4
done
```

While Loops

While loops are another common programming construct for repeated execution of code blocks. They differ from for loops in how they determine how many times to run.

Example:

```
#!/bin/bash


filenum=10 ##variable filenum has a value of 10##
while [ $filenum -gt 0 ] ##while variable filenumber is greater than 0
do
        touch vide$filenum.mp4 ##create a file video$filenum.mp4
        filenum=$(($filenum -1)) ##change variable filenum to it's current value -1
done
```

In this example, a while loop is being used to create 10 files, all numbered, counting down from 10 until the value of 0 is reaching - meaning that 10 files are created.

Bash

New Page



Programming Constructs

FOR LOOPS

For loops are a fundamental programming construct used to execute a block of code repeatedly. They are particularly useful when you need to process a sequence of items, like elements in a list or characters in a string.

Example:

```
#!/bin/bash

for filename in `seq 10` ##sets the variable filename as the output of seq 10
do
touch video$filename.mp4 ##create file video$filename.mp4

done
```

WHILE LOOPS

While loops are another common programming construct for repeated execution of code blocks. They differ from for loops in how they determine how many times to run.

Example:

```
#!/bin/bash

filenum=10 ##variable filenum has a value of 10##

while [ $filenum -gt 0 ] ##while variable filenumber is greater than 0
do
touch vide$filenum.mp4 ##create a file video$filenum.mp4
```

```
filenum=$(($filenum -1)) ##change variable filenum to it's current value -1


done
```

In this example, a while loop is being used to create 10 files, all numbered, counting down from 10 until the value of 0 is reaching - meaning that 10 files are created.

IF & ELSE Statement

An if statement is a fundamental building block in programming that allows you to control the flow of your code based on certain conditions. It's like a decision-making tool for your program.

if statements are opened with the if command, and closed with the fi command. Anything past the fi command will run regardless of the outcome of the if statement.

example:

```
#!/bin/bash


echo -e "Username:   "
read username ##prompt user input for variable username
if
        id -u $username > -1 ##if UID of the username (variable) is greater than -1
then
        echo "That username already exists"
else ##Providing that the criteria defined by the IF statement isn't met, the else statement
can proceed
        echo -e "Username   "
        read fullname
        echo -e "Password   "
        read password
        sudo useradd -c "$fullname" $username =p $password
fi
done
```
