Shell Programming

**Using Multiple Commands**

$ date ; who

#!/bin/bash

# This script displays the date and who’s logged on

date

who

$ echo $PATH

$ echo Let’s see if this’ll work

$ echo "This is a test to see if you’re paying attention"

$ echo ’John says "scripting is easy".’

$ cat test1

#!/bin/bash

# This script displays the date and who’s logged on

echo The time and date are:

date

echo "Let’s see who’s logged into the system:"

who

$

echo -n "The time and date are: "

**Using Variables**

**Environment variables**

$ set

$ cat test2

#!/bin/bash

# display user information from the system.

echo "User info for userid: $USER"

echo UID: $UID

echo HOME: $HOME

$

$ echo "The cost of the item is $15"

The cost of the item is 5

$ echo "The cost of the item is \$15"

The cost of the item is $15

**User variables**

var1=10

var2=-57

var3=testing

var4="still more testing"

$ cat test3

#!/bin/bash

# testing variables

days=10

guest="Katie"

echo "$guest checked in $days days ago"

days=5

guest="Jessica"

echo "$guest checked in $days days ago"

$

What is the output of

$ cat test4

#!/bin/bash

# assigning a variable value to another variable

value1=10

value2=value1

echo The resulting value is $value2

$

**The backtick**

$ cat test5

#!/bin/bash

# using the backtick character

testing=`date`

echo "The date and time are: " $testing

$

#!/bin/bash

# copy the /usr/bin directory listing to a log file

today=`date +%y%m%d`

ls /usr/bin -al *>* log.$today

**Redirecting Input and Output**

**Output redirection**

command *>* outputfile

$ date *>* test6

$ who *>* test6

$ date *>>* test6

**Input redirection**

command < inputfile

$ wc < test6

The wc command provides a count of text in the data. By default it produces three values:

■ The number of lines in the text

■ The number of words in the text

■ The number of bytes in the text

**Pipes**

$ rpm -qa > rpm.list ….$sudo apt install rpm

$cat /etc/passwd | sort

$cat /etc/passwd | sort | more

$cat /etc/passwd | sort > newfile

**Performing Math**

**The expr command**

$ expr 1 + 5 -, / What about \*

#!/bin/bash

# An example of using the expr command

var1=10

var2=20

var3=`expr $var2 / $var1`

echo The result is $var3

**Using brackets**

$[ operation ]

$ var1=$[1 + 5]

$ echo $var1

6

$ var2 = $[$var1 \* 2]

$ echo $var2

12

$

$ cat test7

#!/bin/bash

var2=50

var3=45

var4=$[$var1 \* ($var2 - $var3)]

echo The final result is $var4

$var1=100

The bash shell mathematical operators only support integer arithmetic.

**A floating-point solution**

built-in bash calculator (called bc).

$bc

The floating-point arithmetic is controlled by a built-in variable called scale.

What is the default value of scale variable?

$ bc -q

var1=10

var1 \* 4

40

var2 = var1 / 5

print var2

2

quit

$

**Using bc in scripts**

variable=`echo "options; expression" | bc`

#!/bin/bash

var1=`echo " scale=4; 3.44 / 5" | bc`

echo The answer is $var1

var1=100

var2=45

var3=`echo "scale=4; $var1 / $var2" | bc`

echo The answer for this is $var3

var1=20

var2=3.14159

var3=`echo "scale=4; $var1 \* $var1" | bc`

var4=`echo "scale=4; $var3 \* $var2" | bc`

echo The final result is $var4

var1=10.46

var2=43.67

var3=33.2

var4=71

var5=`bc *<<* EOF

scale = 4

a1 = ( $var1 \* $var2)

b1 = ($var3 \* $var4)

a1 + b1

EOF

`

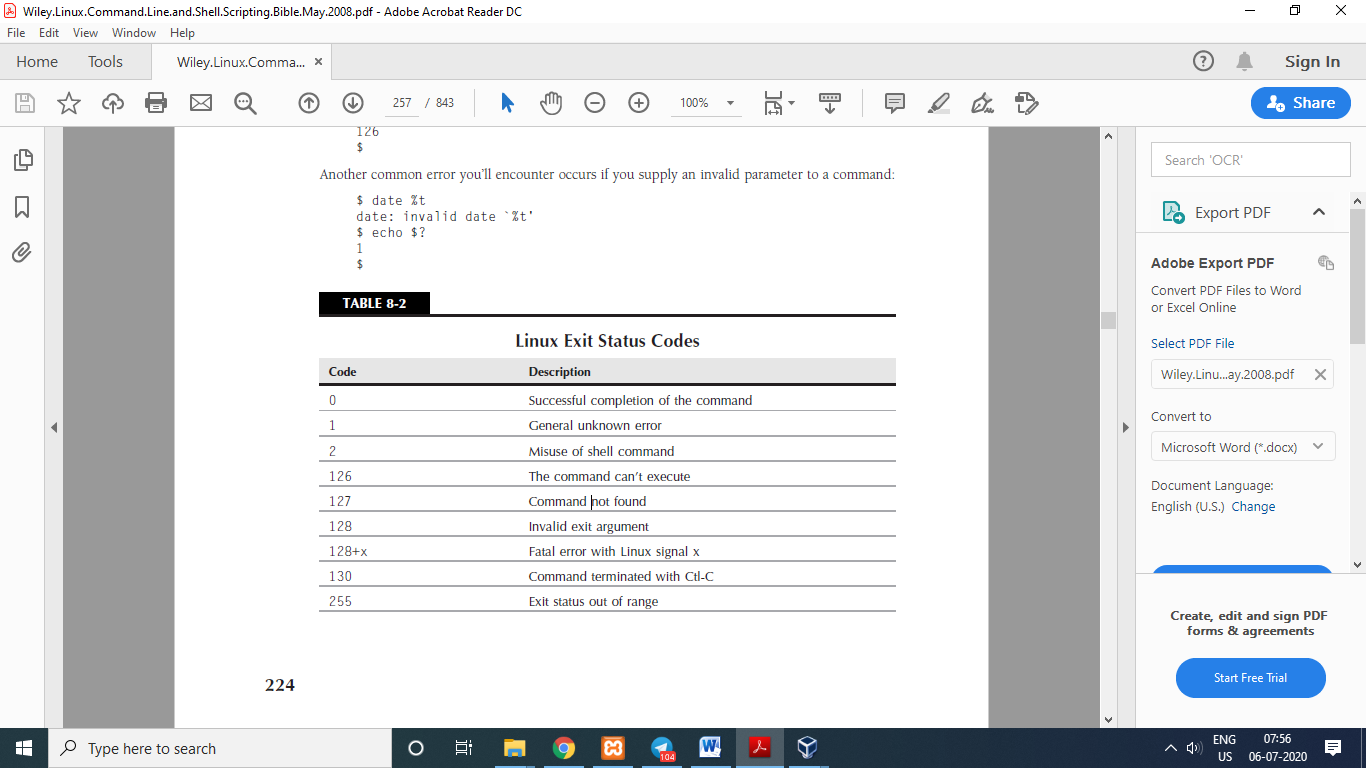
echo The final answer for this mess is $var5

**Exiting the Script**

**Checking the exit status**

Linux provides the $? special variable that holds the exit status value from the last command

that executed.



By default, your shell script will exit with the exit status of the last command in your script:

# testing the exit status

var1=10

var2=30

var3=$[ $var1 + var2 ]

echo The answer is $var3

exit 5

#!/bin/bash

# testing the exit status

var1=10

var2=30

var3=$[ $var1 \* var2 ]

echo The value is $var3

exit $var3

**Working with the if-then Statement**

if command

then

commands

fi

$ cat test1

#!/bin/bash

# testing the if statement

if date

then

echo "it worked"

fi

$ cat test2

#!/bin/bash

# testing a bad command

if asdfg

then

echo "it didn’t work"

fi

echo "we’re outside of the if statement"

$ cat test3

#!/bin/bash

# testing multiple commands in the then section

testuser=girish

if grep $testuser /etc/passwd

then

echo The bash files for user $testuser are:

ls -a /home/$testuser/.b\*

fi

if command

then

commands

else

commands

fi

$ cat test4

#!/bin/bash

# testing the else section

testuser=badtest

if grep $testuser /etc/passwd

then

echo The files for user $testuser are:

ls -a /home/$testuser/.b\*

else

echo "The user name $testuser doesn’t exist on this system"

fi

**Nesting ifs**

if command1

then

commands

elif command2

then

more commands

fi

if command1

then

command set 1

elif command2

then

command set 2

elif command3

then

command set 3

elif command4

then

command set 4

fi

**The test Command**

The format of the test command is pretty simple:

test condition

if test condition

then

commands

fi

if [ condition ]

then

commands

fi

There are three classes of conditions the test command can evaluate:

■ Numeric comparisons

■ String comparisons

■ File comparisons

**Numeric comparisons**

$ cat test5

#!/bin/bash

# using numeric test comparisons

val1=10

val2=11

if [ $val1 -gt 5 ]

then

echo "The test value $val1 is greater than 5"

fi

if [ $val1 -eq $val2 ]

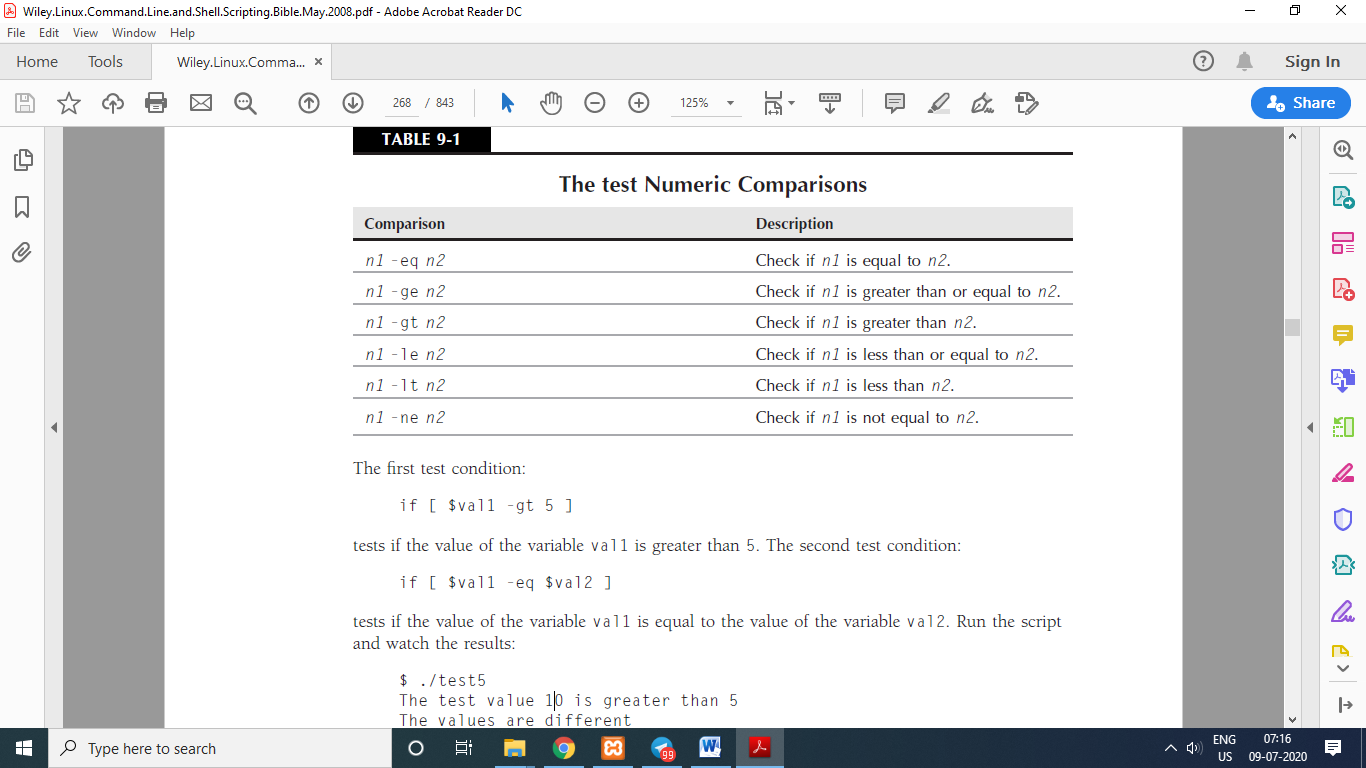
then

echo "The values are equal"

else

echo "The values are different"

fi



$ cat test6

#!/bin/bash

# testing floating point numbers

val1=` echo "scale=4; 10 / 3 " | bc`

echo "The test value is $val1"

if [ $val1 -gt 3 ]

then

echo "The result is larger than 3"

fi

**String comparisons**

# testing string equality

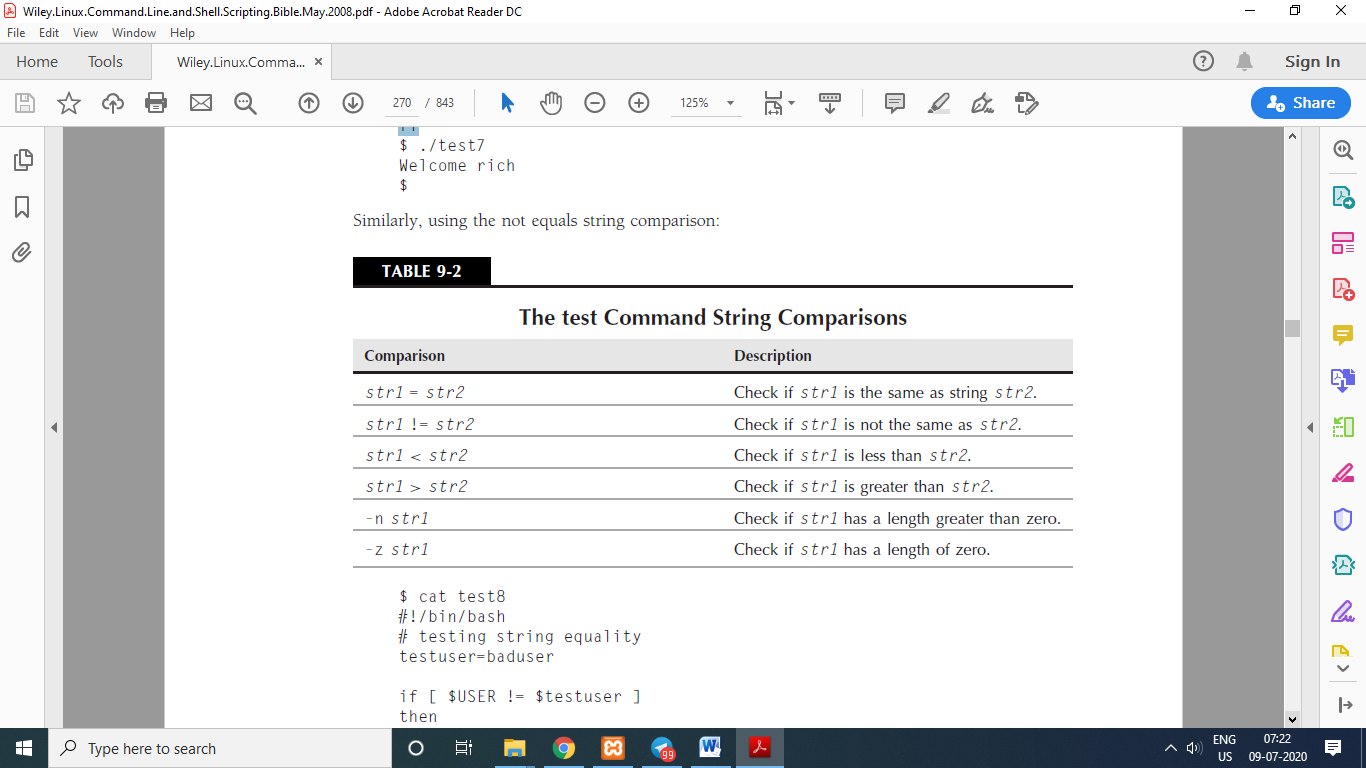
testuser=girish

if [ $USER = $testuser ]

then

echo "Welcome $testuser"

fi



$ cat test8

#!/bin/bash

# testing string equality

testuser=baduser

if [ $USER != $testuser ]

then

echo "This isn’t $testuser"

else

echo "Welcome $testuser"

fi

$ cat badtest

#!/bin/bash

# mis-using string comparisons

val1=baseball

val2=hockey

if [ $val1 > $val2 ]

then

echo "$val1 is greater than $val2"

else

echo "$val1 is less than $val2"

fi

**String size**

$ cat test10

#!/bin/bash

# testing string length

val1=testing

val2=’’

if [ -n $val1 ]

then

echo "The string ’$val1’ is not empty"

else

echo "The string ’$val1’ is empty"

fi

if [ -z $val2 ]

then

echo "The string ’$val2’ is empty"

else

echo "The string ’$val2’ is not empty"

fi

if [ -z $val3 ]

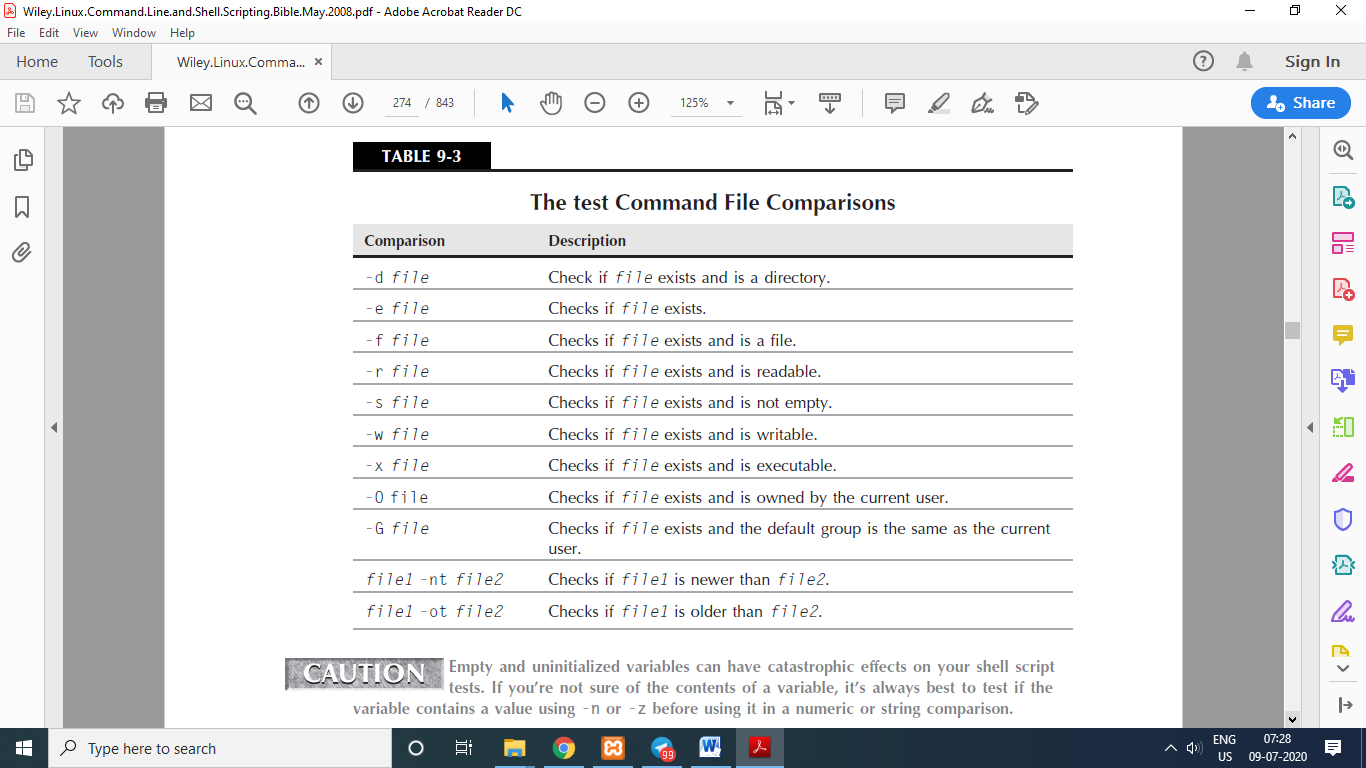
then

echo "The string ’$val3’ is empty"

else

echo "The string ’$val3’ is not empty"

fi



**File comparisons**

$ cat test11

#!/bin/bash

# look before you leap

if [ -d $HOME ]

then

echo "Your HOME directory exists"

cd $HOME

ls -a

else

echo "There’s a problem with your HOME directory"

fi

**Checking if an object exists**

$ cat test12

#!/bin/bash

# checking if a directory exists

if [ -e $HOME ]

then

echo "OK on the directory, now let’s check the file"

# checking if a file exists

if [ -e $HOME/testing ]

then

# the file exists, append data to it

echo "Appending date to existing file"

date >> $HOME/testing

else

# the file doesn’t exist, create a new file

echo "Creating new file"

date > $HOME/testing

fi

else

echo "Sorry, you don’t have a HOME directory"

fi

**Checking for a file**

$ cat test13

#!/bin/bash

# check if a file

if [ -e $HOME ]

then

echo "The object exists, is it a file?"

if [ -f $HOME ]

then

echo "Yes, it’s a file!"

else

echo "No, it’s not a file!"

if [ -f $HOME/.bash history ]

then

echo "But this is a file!"

fi

fi

else

echo "Sorry, the object doesn’t exist"

fi

**Test for reading a file**

$ cat test14

#!/bin/bash

# testing if you can read a file

pwfile=/etc/shadow

# first, test if the file exists, and is a file

if [ -f $pwfile ]

then

# now test if you can read it

if [ -r $pwfile ]

then

tail $pwfile

else

echo "Sorry, I’m unable to read the $pwfile file"

fi

else

echo "Sorry, the file $file doesn’t exist"

fi

**Checking for empty files**

$ cat test15

#!/bin/bash

# testing if a file is empty

file=t15test

touch $file

if [ -s $file ]

then

echo "The $file file exists and has data in it"

else

echo "The $file exists and is empty"

fi

date > $file

if [ -s $file ]

then

echo "The $file file has data in it"

else

echo "The $file is still empty"

fi

**Checking if you can write to a file**

$ cat test16

#!/bin/bash

# checking if a file is writeable

logfile=$HOME/t16test

touch $logfile

chmod u-w $logfile

now=`date +%Y%m%d-%H%M`

if [ -w $logfile ]

then

echo "The program ran at: $now" > $logfile

echo "The fist attempt succeeded"

else

echo "The first attempt failed"

fi

chmod u+w $logfile

if [ -w $logfile ]

then

echo "The program ran at: $now" > $logfile

echo "The second attempt succeeded"

else

echo "The second attempt failed"

fi

**Checking if you can run a file**

$ cat test17

#!/bin/bash

# testing file execution

if [ -x test16 ]

then

echo "You can run the script:"

./test16

else

echo "Sorry, you are unable to execute the script"

fi

**Checking ownership**

$ cat test18

#!/bin/bash

# check file ownsership

if [ -O /etc/passwd ]

then

echo "You’re the owner of the /etc/passwd file"

else

echo "Sorry, you’re not the owner of the /etc/passwd file"

fi

The -G comparison checks the default group of a file, and it succeeds if it matches the group of the default group for the user.

$ cat test19

#!/bin/bash

# check file group test

if [ -G $HOME/testing ]

then

echo "You’re in the same group as the file"

else

echo "The file is not owned by your group"

fi

**Compound Condition Testing**

■ [ condition1 ] && [ condition2 ]

■ [ condition1 ] || [ condition2 ]

#!/bin/bash

# testing compound comparisons

if [ -d $HOME ] && [ -w $HOME/testing ]

then

echo "The file exists and you can write to it"

else

echo "I can’t write to the file"

fi

**Advanced if-then Features**

■ Double parentheses for mathematical expressions

■ Double square brackets for advanced string handling functions

**Using double parentheses**

(( *expression* ))

#!/bin/bash

# using double parenthesis

val1=10

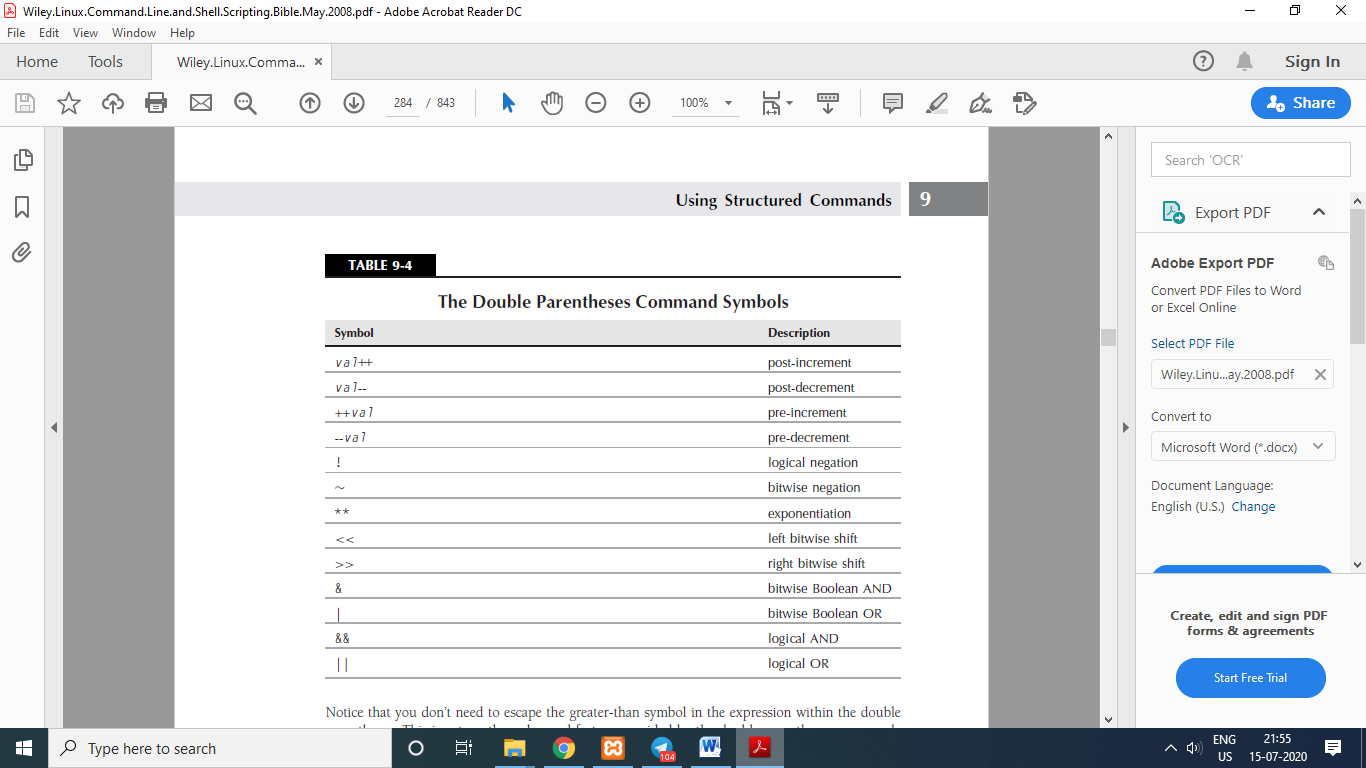
if (( $val1 \*\* 2 *>* 90 ))

then

(( val2 = $val1 \*\* 2 ))

echo "The square of $val1 is $val2"

fi



**Using double brackets**

[[ expression ]]

#!/bin/bash

# using pattern matching

if [[ $USER == g\* ]]

then

echo "Hello $USER"

else

echo "Sorry, I don’t know you"

fi

**The case Command**

#!/bin/bash

# looking for a possible value

if [ $USER = "girish" ]

then

echo "Welcome $USER"

echo "Please enjoy your visit"

elif [ $USER = baby ]

then

echo "Welcome $USER"

echo "Please enjoy your visit"

elif [ $USER = testing ]

then

echo "Special testing account"

elif [ $USER = james ]

then

echo "Don’t forget to logout when you’re done"

else

echo "Sorry, you’re not allowed here"

fi

case *variable* in

*pattern1* | *pattern2*) *commands1*;;

*pattern3*) *commands2*;;

\*) *default commands*;;

esac

#!/bin/bash

# using the case command

case $USER in

girish | baby)

echo "Welcome, $USER"

echo "Please enjoy your visit";;

testing)

echo "Special testing account";;

james)

echo "Don’t forget to log off when you’re done";;

\*)

echo "Sorry, you’re not allowed here";;

esac

**The for Command**

for var in *list*

do

*commands*

done

**Reading values in a list**

#!/bin/bash

# basic for command

for test in Alabama Alaska Arizona Arkansas California Colorado

do

echo The next state is $test “…” ‘….’

done

**Reading complex values in a list**

#!/bin/bash

# another example of how not to use the for command

for test in I don’t know if this’ll work

do

echo "word:$test"

done

There are two ways to solve this problem:

■ Use the escape character (the backslash) to escape the single quotation mark.

■ Use double quotation marks to define the values that use single quotation marks.

for test in I don\’t know if "this’ll" work

do

echo "word:$test"

done

Yet another problem you may run into is multi-word values.

for test in Nevada New Hampshire New Mexico New York North Carolina

do

echo "Now going to $test"

done

for test in Nevada "New Hampshire" "New Mexico" "New York"

do

echo "Now going to $test"

done

**Reading a list from a variable**

list="Alabama Alaska Arizona Arkansas Colorado"

list=$list" Connecticut"

for state in $list

do

echo "Have you ever visited $state?"

done

**Reading values from a command**

#!/bin/bash

# reading values from a file

file="states"

for state in `cat $file`

do

echo "Visit beautiful $state"

done

By default, the bash shell considers the following characters as field separators:

■ A space

■ A tab

■ A newline

IFS=$’\n’

IFS.OLD=$IFS

IFS=$’\n’

‹use the new IFS value in code›

IFS=$IFS.OLD

IFS=$’\n’:;"

**Reading a directory using wildcards**

for file in /home/rich/test/\*

do

if [ -d "$file" ]

then

echo "$file is a directory"

elif [ -f "$file" ]

then

echo "$file is a file"

fi

done

**The C-Style for Command**

for (i = 0; i ‹ 10; i++)

{

printf("The next number is %d\n", i);

}

for (( variable assignment ; condition ; iteration process ))

for (( a = 1; a ‹ 10; a++ ))

for (( i=1; i ‹= 10; i++ ))

do

echo "The next number is $i"

done