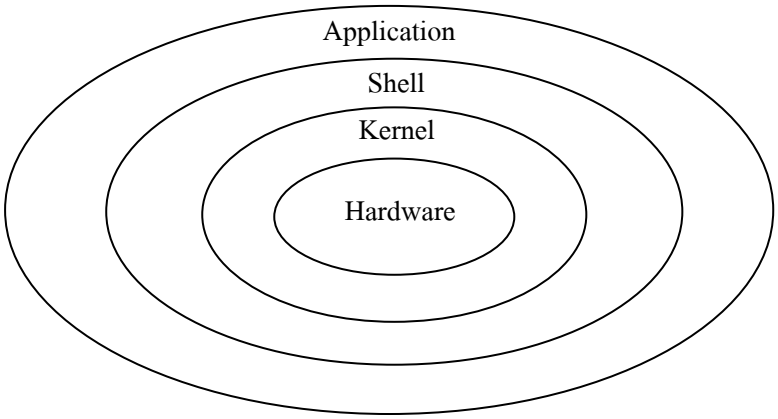
การใช้ Shell Script เบื้องต้น

โครงสร้างพื้นฐานการทำงานของระบบ UNIX มีอยู่ 4 ส่วนด้วยกัน คือ Hardware, Kernel, Shell และ Application ดังรูป



Shell คือ โปรแกรมหนึ่งบนระบบ UNIX ที่ทำหน้าที่เป็น interface ระหว่าง user กับ UNIX (Kernel) user สามารถสั่งงาน UNIX ได้โดยผ่านทาง Shell เท่านั้น โปรแกรม Shell ยังมีคุณสมบัติของ Shell Programming Language ทำให้ user สามารถนำคำสั่งต่างๆของ Shell มาเขียนเป็นโปรแกรมเก็บเป็นไฟล์ไว้ได้ เรียกว่า Shell Script

Shell ที่นิยมใช้ในปัจจุบัน

- Bourne shell (/bin/sh) เป็น shell ในยุคแรกๆ ที่มีใช้กันอย่างแพร่หลาย มีการกำหนดโครงสร้างภาษาคล้ายๆ กับภาษาอัลกอ (Algo) สามารถเขียน shell script ได้ และยังเป็น starndard shell ที่มีใน UNIX ทุกตัว และยังสามารถย้าย shell script ไปยัง UNIX ระบบอื่นโดยไม่ต้องแก้ไขอะไรได้อีกด้วย จะมี default prompt เป็นเครื่องหมาย “ $ ”
- C shell (/bin/csh) เป็น shell ที่พัฒนาขึ้นมาหลังจาก Bourne shell มีรูปแบบคำสั่งและไวยากรณ์เหมือนกับภาษา C มี function การทำงานหลายที่ดีและอย่างสะดวก อีกทั้งยังสามารถควบคุมการไหลของข้อมูลได้ดีกว่า Bourne shell และยังมีความสามารถในการเรียกใช้คำสั่งที่ใช้ไปแล้ว จะมี default prompt เป็นเครื่องหมาย “ % ”
- Korn shell (/bin/ksh) เป็น shell ที่พัฒนามาจาดต้นแบบของ Bourne shell และ C shell สามารถทำงานใน function ของ Bourne shell ได้ทุกอย่าง การเขียน shell script ทำได้ง่ายและรัดกุมขึ้น สามารถนำคำสั่งที่ใช้ไปแล้วกลับมา execute ไปใหม่ได้ ถือได้ว่า Korn shell เป็นการรวมเอาข้อดีของ Bourne shell และ C shell มาไว้ด้วยกัน แต่ไม่ได้มีในUNIX ทุกตัว จะมี default prompt เป็นเครื่องหมาย “ $ ”
- Bourne again shell (/bin/bash หรือ /usr/local/bin/bash) เป็นการเอา Bourne shell นำกลับมาพัฒนาใหม่ สามารถทำงานแบบ line editing ได้ และยังได้เพิ่มประสิทธิภาพในการทำงานอีกหลายอย่าง bash shell นี้ไม่ใช่ standard UNIX shell แต่เป็น default shell ของ linux ในปัจจุบัน จะมี default prompt เป็นเครื่องหมาย “ $ ”

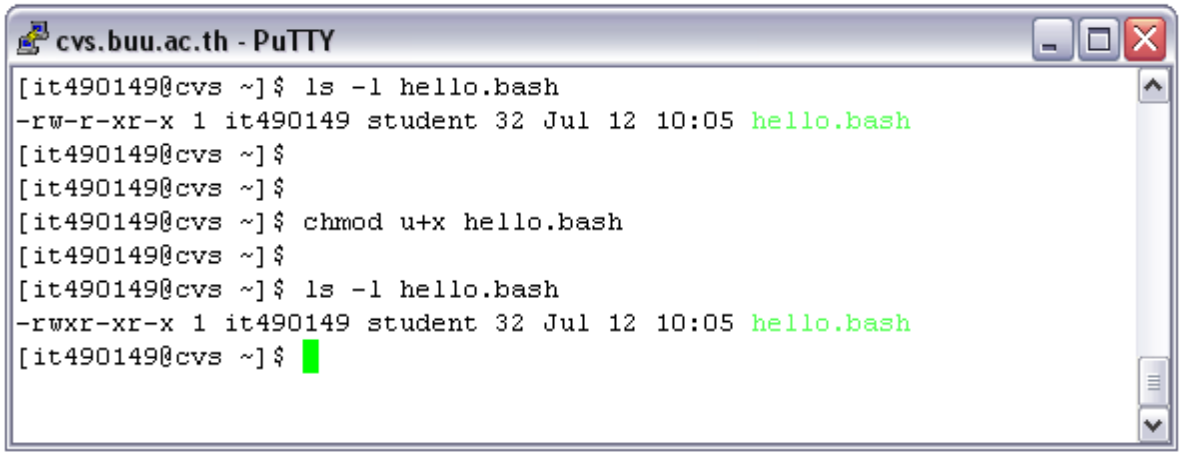
Hello World

Shell script เป็นไฟล์เท็กซ์ธรรมดาสามารถสร้างด้วย editor ชนิดไหนก็ได้ เช่น vi, nano, gedit, emacs เป็นต้น

	การเข้าใช้งาน vi (ใน prompt)
Syntax	**$ vi filename**
Example	**การเขียน shell script ใน vi (ชื่อไฟล์ hello.bash)**

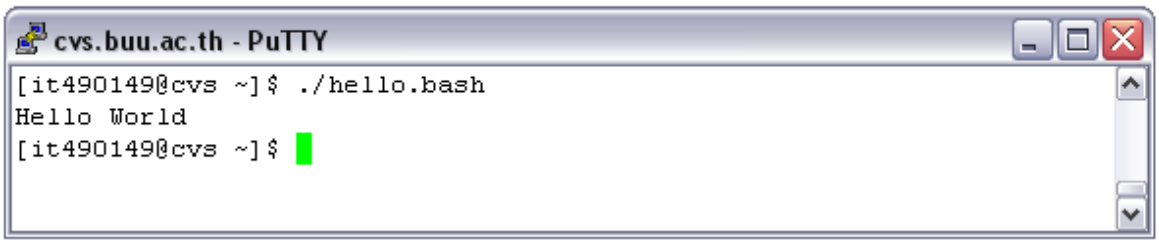
การที่จะรันสคริปต์ได้ต้องทำให้ไฟล์ที่สร้างขึ้นมาสามารถกระทำการได้ ถ้าดูไฟล์ด้วยคำสั่ง " ls –l hello.bash " (สมมติให้ไฟล์ชื่อว่า hello.bash) จะเห็นว่าสิทธิ์การใช้ไฟล์จะเป็น –rw-r-xr-x ซึ่งหมายความว่า ไฟล์นั้นยังไม่สามารถกระทำการได้ การที่จะทำให้ไฟล์นั้นสามารถกระทำการได้ ต้องใช้คำสั่ง chmod กำหนดสิทธิให้ไฟล์นั้นเสียก่อน

	การเรียกดูสิทธิในการเข้าถึงไฟล์ และ การกำหนดสิทธิให้กับไฟล์ (ใน prompt)
Syntax	**$ ls –l filename และ $ chmod [u / g / o] [+ / -] [r / w / x] filename**
Example	**การใช้คำสั่ง ls -l และ chmod**

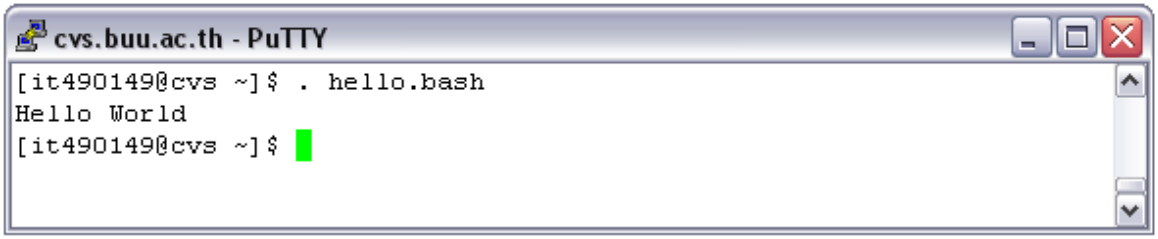


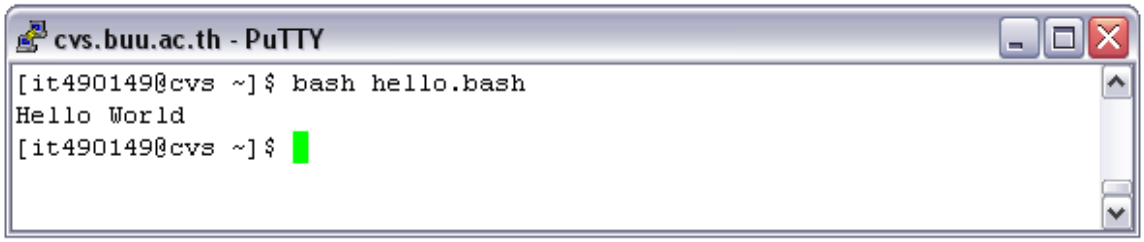
ถ้าดูไฟล์ด้วยคำสั่ง “ ls –l hello.bash “ จะเห็นว่าสิทธิ์การใช้ไฟล์เปลี่ยนเป็น -rwxr-xr-x. ซึ่งหมายความว่า ไฟล์นั้นสามารถกระทำการได้แล้ว (x = ก็คือการexecutable นั่นเอง).

	การรัน shell script (ใน prompt)
Syntax	**$. / filename**
Example	**การรัน File hello.bash**



	การรัน shell script แบบอื่น (ใน prompt)
yntax	**$. hello.bash หรือ $ bash hello.bash**
Example	**การรัน File hello.bash**





ตัวแปร (variable)

เชลล์สคริปต์เหมือนกับโปรแกรมทั่วๆไปลที่มีตัวแปรไว้เก็บค่าต่างๆ สำหรับใช้งาน ตัวแปรที่ใช้ในเชลล์นั้นไม่จำเป็นต้องประกาศชนิดตัวแปรเหมือนกับภาษาซี สามารถตั้งค่าแล้วนำไปใช้ได้ทันที รูปแบบการตั้งค่าตัวแปร

	**รูปแบบการตั้งตัวแปร (ใน vi)**
Syntax	**variable = value**
Example	**การสร้างตัวแปร**

ข้อควรระวังคือในการใช้ตัวแปร คือในช่วงก่อนหน้าและหลังเครื่องหมาย ”=” ห้ามมีช่องว่าง เพราะเชลล์ถือว่าช่องว่าคือตัวแบ่ง argument

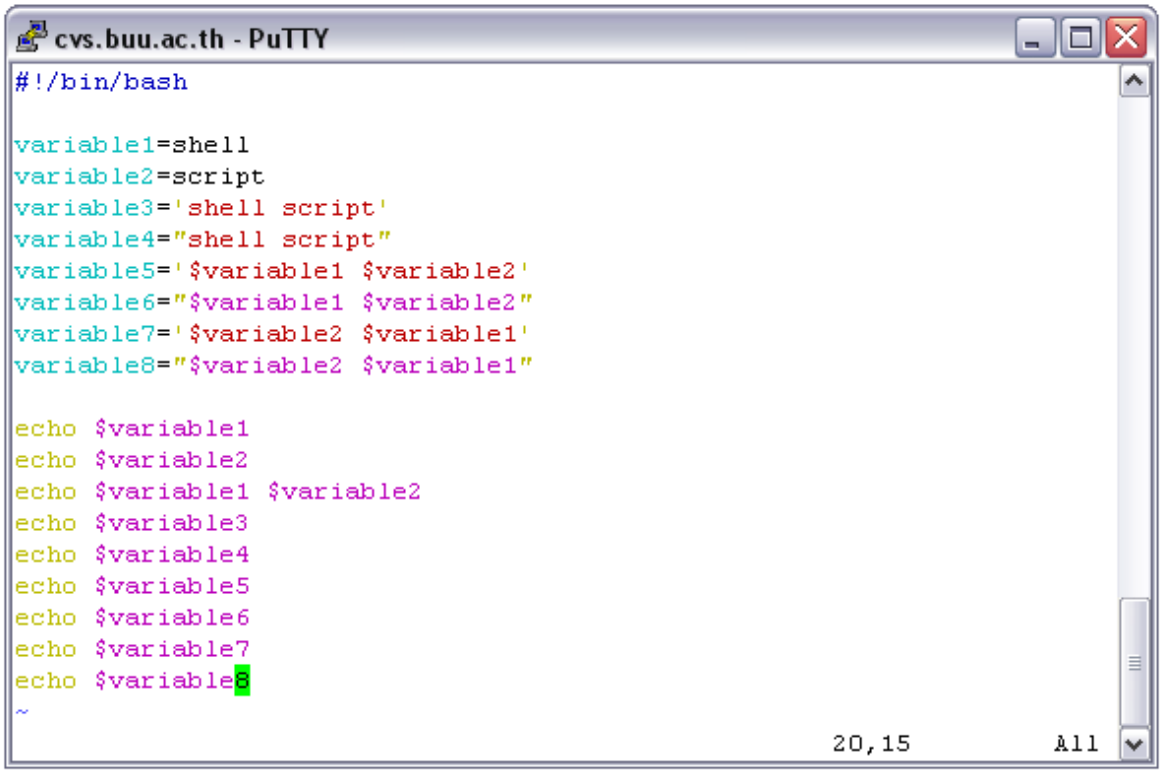
	**การเรียกใช้ตัวแปร (ใน vi)**
Syntax	**$ variable**
Example	**การเรียกใช้ตัวแปร**

เวลาที่จะต้องการแสดงค่าให้ใช้เครื่อง”$”นำหน้าตัวแปร

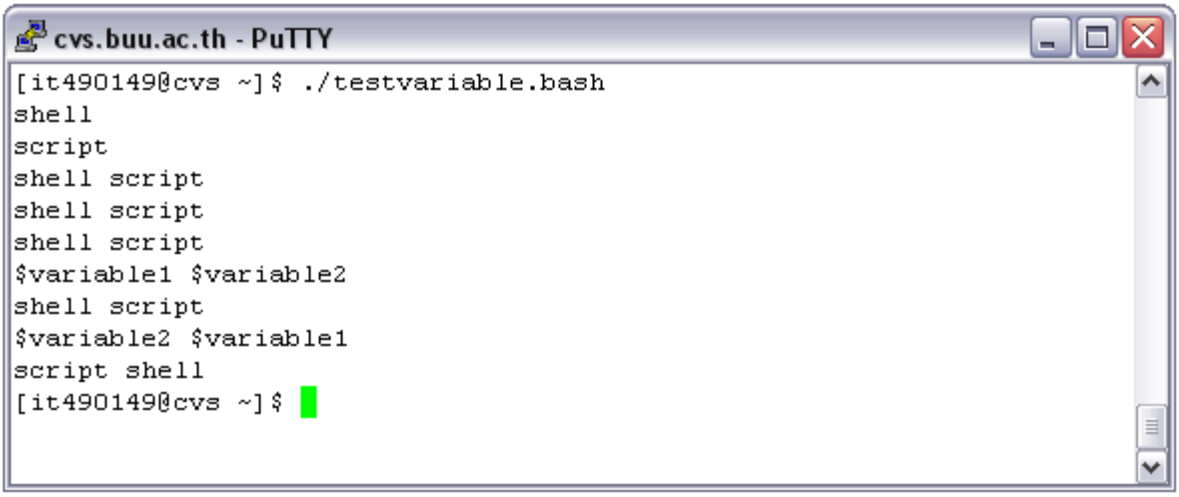
เครื่องหมาย Quote (‘) และ Double quote (“) ช่องว่างในเชลล์นั้นถือว่าเป็นตัวแบ่งอาร์กิวเมนต์ ถ้าต้องการตั้งค่าตัวแปรที่มีช่องว่าง ให้ใช้เครื่องหมาย Quote (‘) หรือ Double quote (“) คล่อมคำที่ต้องการ

	การตั้งค่าตัวแปรที่มีช่องว่าง (ใน vi)
Syntax	**variable=’value’ or variable=”value”**
Example	**การตั้งค่าตัวแปร**

	การตั้งค่าตัวแปรพร้อมการเรียกใช้
Example	**File testvariable.bash**



	ผลลัพธ์ที่ได้



ความแตกต่างระหว่าง Quote (‘) และ Double quote (“) คือสิ่งที่พิมพ์ใน เครื่องหมาย Quote (‘) จะมีค่าตามสิ่งที่พิมพ์ ส่วนในเครื่องหมาย Double quote (“) จะเป็นการอ้างอิงนำผลลัพธ์ที่ตัวแปรเก็บไว้หรือค่าที่ตัวแปรเก็บไว้มาใช้

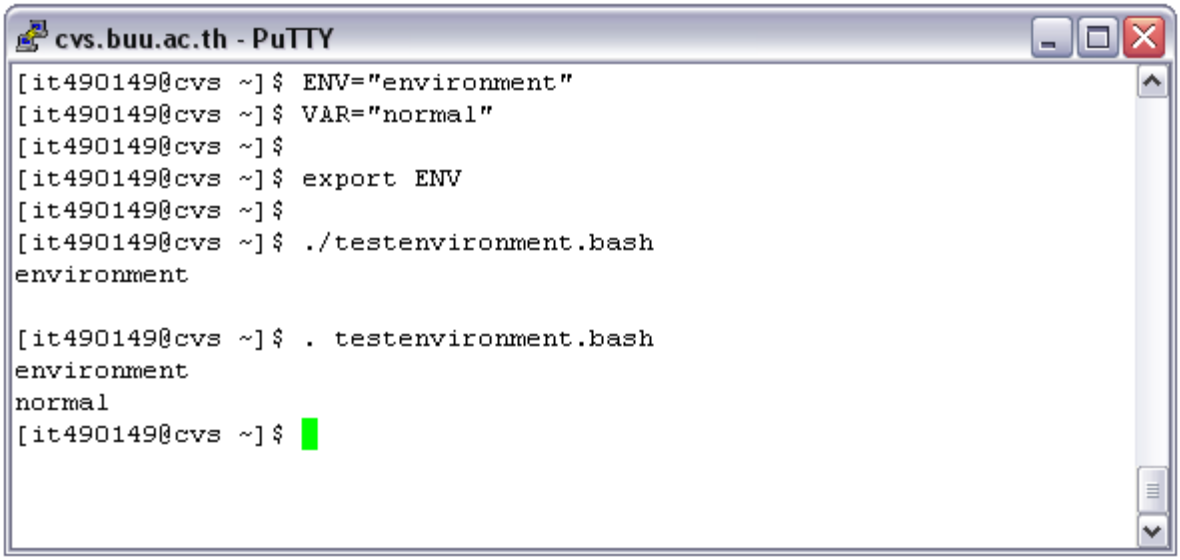
ตัวแปรสภาพแวดล้อม (environment variable)

ตัวแปรที่ใช้ในเชลล์จะมีสองชนิดด้วยกัน คือ 1.ตัวแปรธรรมดา และ 2.ตัวแปรสภาพแวดล้อม (environment variable) ตัวแปรสภาพแวดล้อมนั้นคล้ายกับตัวแปรธรรมดาแต่แตกต่างตรงที่ เมื่อโปรแกรมรันในเชลล์นั้น จะสืบทอดตัวแปรสภาพแวดล้อมและค่าที่อยู่ในตัวแปรสภาพแวดล้อมไปด้วย

เราสามารถสร้างตัวแปรธรรมดาให้เป็นตัวแปรสภาพแวดล้อมได้ โดยการใช้คำสั่ง export.

	**การตั้งตัวแปร (ใน vi)**
Syntax	**variable = value**
Example	**การสร้างตัวแปรใน File testenvironment.bash**

	การเปลี่ยนตัวแปรธรรมดาให้เป็นตัวแปรสภาพแวดล้อม (ใน prompt)
Syntax	**$ export variable**
Example	**การเปลี่ยนตัวแปร ENV ให้เป็นตัวแปรสภาพแวดล้อมใน prompt**



จะเห็นได้ว่าผลลัพธ์นั้นต่างกัน เพราะ การรัน shell script แบบแรกจะเป็นการเรียก shell (สร้างโปรเซสใหม่) และ shell ตัวใหม่จะเป็นตัวกระทำการ shell ตัวใหม่นี้คือ #!/bin/bash ที่เขียนใน file testenvironment.bash บรรทัดแรก shell (โปรแกรม) ที่เกิดใหม่สืบทอดตัวแปรสภาพแวดล้อม ENV มาด้วยทำให้แสดงผลได้ แต่ไม่สามารถแสดงค่าของตัวแปร VAR ได้เพราะตัวแปร VAR ไม่ใช่ตัวแปรสภาพแวดล้อมจึงไม่สามารถสืบทอดค่าที่อยู่ในตัวแปรไปใช้ในโปรแกรมอื่นได้

ส่วนการรัน แบบที่สองใช้การรันแบบ “.” (dot) เป็นการใช้ shell ที่ทำงานอยู่ในปัจจุบันอ่านไฟล์ testenvironment.bash จึงสามารถแสดงค่าของตัวแปรทั้งสองตัวได้

เราสามารถรู้ได้ว่ามีตัวแปรสภาพแวดล้อมอะไรบ้างที่อยู่ในระบบ โดยการใช้คำสั่ง export ซึ่งเป็นคำสั่งเดียวกันกับที่ใช้ประกาศตัวแปรสภาพแวดล้อม

	การดูตัวแปรสภาพแวดล้อมที่อยู่ในระบบ (ใน prompt)
Syntax	**$ export**

เมื่อลองใช้คำสั่งดูจะเห็นได้ว่ามีตัวแปรสภาพแวดล้อมมากมายที่ตั้งค่าไว้อัตโนมัติในระบบ ถ้าต้องการเปลี่ยนตัวแปรสภาพแวดล้อมเป็นตัวแปรธรรมดาสามารถทำได้โดยใช้ option –n ต่อท้ายคำสั่ง export

	การเปลี่ยนตัวแปรสภาพแวดล้อมเป็นตัวแปรธรรมดา (ใน prompt)
Syntax	**$ export –n variable**

ตัวแปรพิเศษ(environment variable)

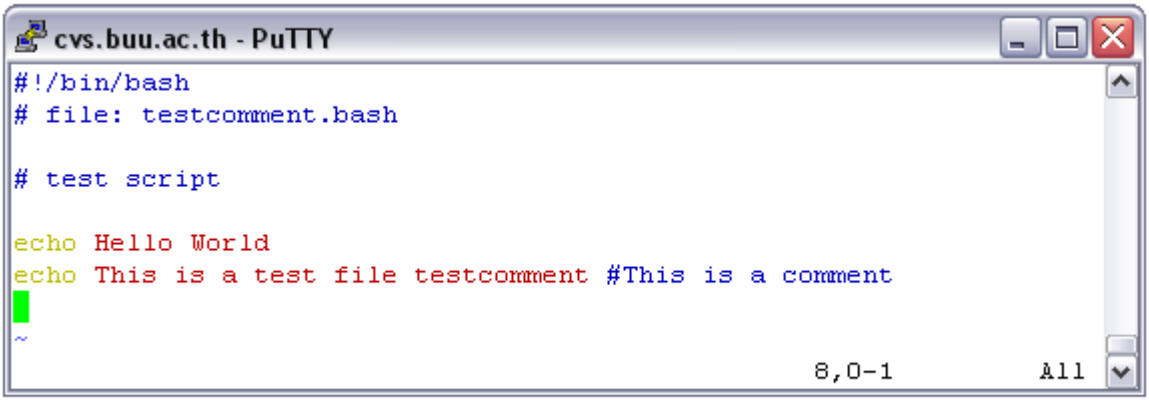
ตัวแปรพิเศษเหล่านี้โดยปรกติเชลล์จะเตรียมตัวแปรเหล่านี้ไว้ให้อยู่แล้วไม่ต้องสร้างเอง และมักจะเป็นตัวแปรที่ไม่สามารถเปลี่ยนค่าได้(read-only)

	ตัวแปรพิเศษที่นิยมใช้กันบ่อยๆ
$0 $1 $2	**ใช้สำหรับอ้างอิงชื่อ shell script และ Argument ของ shell script เรียกว่า position parameter**
$#	**ใช้บอกจำนวน Argument ที่อยู่ใน shell script นั้นๆ**
$*	**แทน Argument ของ script เรียงกันทั้งหมด**
$@	**คล้ายกับ $* แต่จะใช้ช่องว่างคั่นระหว่าง position parameter**
$?	**ใช้แสดงสถานการณ์จบการทำงานครั้งสุดท้าย ถ้าสั่งได้ถูกต้องไม่มี error จะแสดงค่า 0 ออกมา แต่ถ้าผิดพลาดจะแสดงค่าที่ไม่ใช่ 0 ออกมา**
	ตัวแปรพิเศษที่นิยมใช้กันบ่อยๆ
$!	**ใช้แสดง process ID ของ shell ที่ทำงานอยู่**
$hostname	**ใช้แสดงชื่อเครื่องที่กำลังใช้งานอยู่**
$pwd	**ใช้แสดง directory ที่ทำงานอยู่**
$oldpwd	**ใช้แสดง directory ก่อนหน้า directory ปัจจุบัน**
$randow	**ใช้ในการสุ่มตัวเลข ตั้งแต่ 0 ถึง 32767**
$home	**ใช้แสดงชื่อ home directory**

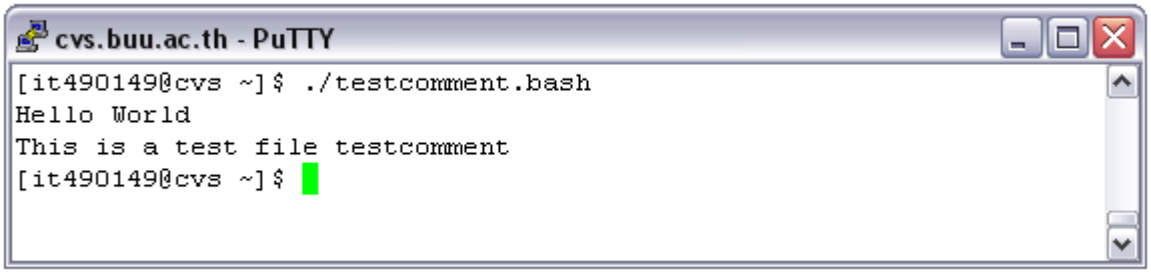
หมายเหตุ (comment)

เครื่องหมาย “ # ” ใช้แสดงหมายเหตุ เขียนไว้ตรงไหนในเชลล์สคริปต์ก็ได้ (ยกเว้นบรรทัดแรก) ใช้สำหรับการเขียนบันทึกใน script โดย shell จะถือว่าข้อความที่อยู่หลังเครื่องหมาย “ # ” เป็นหมายเหตุไม่มีผลต่อตัวโปรแกรม

	**การใช้เครื่องหมาย “ # ” ในการ comment (ใน vi)**
Syntax	**# text**
Example	**File testcomment.bash**



	**ผลลัพธ์ที่ได้**

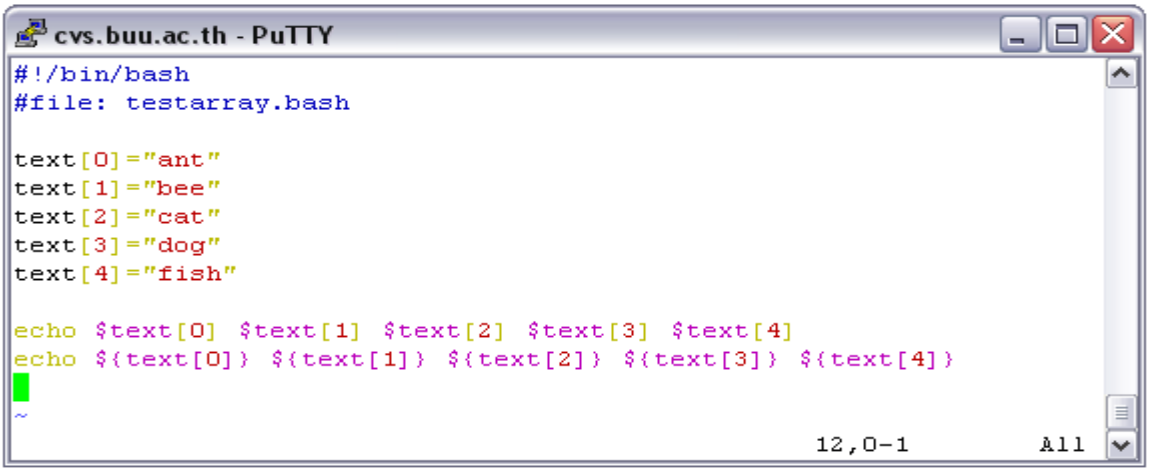


จะเห็นได้ว่าข้อความที่อยู่หลังเครื่องหมาย “ # “ ไม่มีผลใดๆทั้งสิ้นเมื่อรันโปรแกรม เมื่อรันโปรแกรม โปรแกรมจะไม่แสดงข้อความที่ comment นั้นออกมา

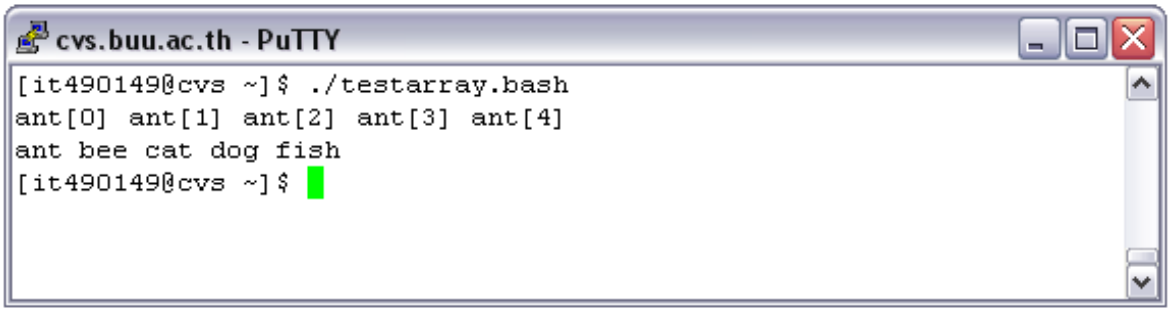
ตัวแปรแถวลำดับ (array variable)

ตัวแปร array นั้นจะมี index เป็นตัวเลขเริ่มตั้งแต่ 0 เหมือนกับตัวแปรในภาษา C หรือภาษาอื่นๆ แต่เวลาที่ต้องการเรียกใช้ตัวแปรต้องใช้เครื่องหมาย “{ }” ช่วยจับกลุ่มด้วย เพราะไม่เช่นนั้นค่าที่ออกมาจะออกมาเฉพาะค่าแรกเท่านั้น ดูได้จากตัวอย่าง

	**รูปแบบการใช้ตัวแปร array (ใน vi)**
Syntax	**variable[index]=value**
Example	**File testarray.bash**

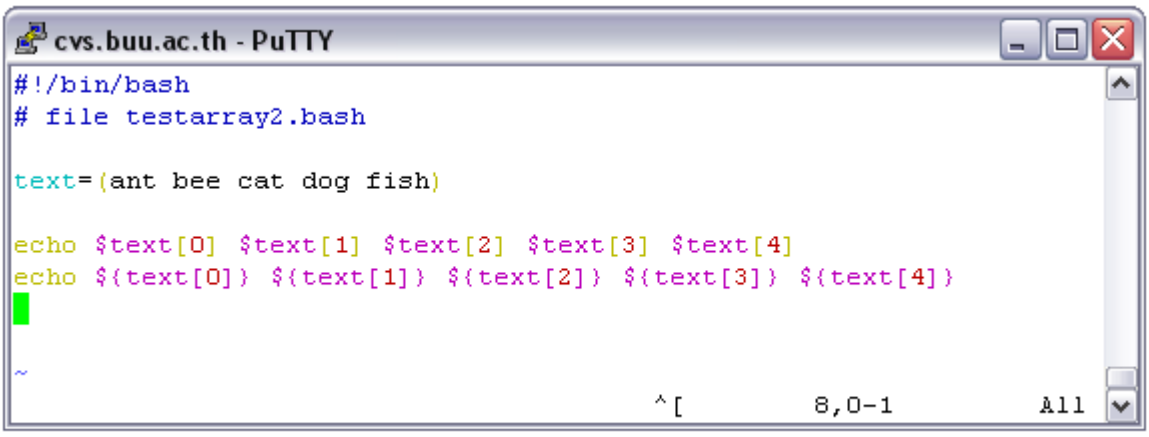


	**ผลลัพธ์ที่ได้**

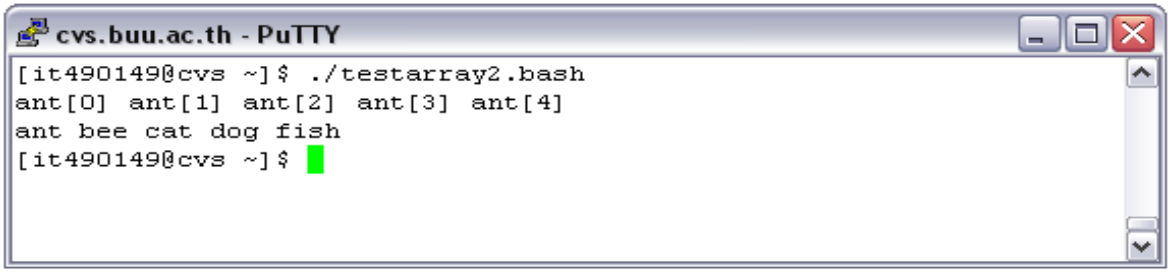


เราสามารถประกาศตัวแปรพร้อมกับการตั้งค่าไปในตัวได้ ดังตัวอย่างต่อไปนี้

	รูปแบบการใช้ตัวแปร array (ใน vi)
Syntax	**variable=(value value value value)**
Example	**File testarray2.bash**



	ผลลัพธ์ที่ได้



จะเห็นได้ว่าสามารถใช้ได้เหมือนกันกับการใช้แบบ index

test และ expression

ใน bash และ ksh จะมีคำสั่ง test และการเขียนเงื่อนไขในการตรวจสอบ expression ต่างๆ ในการเขียนประโยคเงื่อนไข

	การใช้คำสั่ง test (ใน vi)
Syntax	**test expression หรือ [expression]**

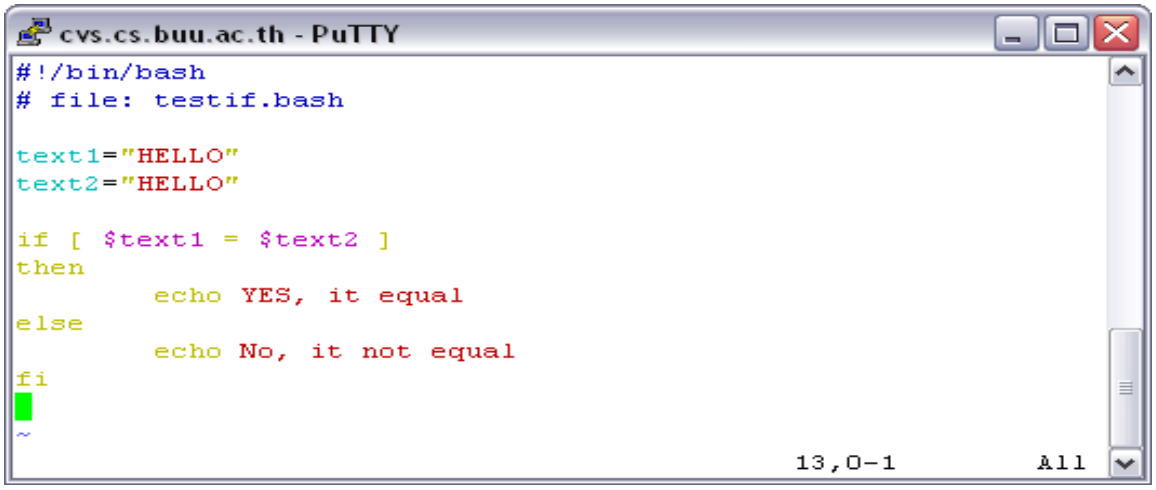
	Expression
คำสั่งเกี่ยวกับจำนวนเต็ม	
int1 –eq int2	**เป็นจริงเมื่อ int1 เท่ากับ int2**
int1 –ne int2	**เป็นจริงเมื่อ int1 ไม่เท่ากับ int2**
int1 –gt int2	**เป็นจริงเมื่อ int1 มากกว่า int2**
int1 –ge int2	**เป็นจริงเมื่อ int1 มากกว่า หรือ เท่ากับ int2**
int1 –le int2	**เป็นจริงเมื่อ int1 น้อยกว่า int2**
int1 –lt int2	**เป็นจริงเมื่อ int1 น้อยกว่า หรือ เท่ากับ int2**
คำสั่งที่เกี่ยวกับ string	
str1 = str2	**เป็นจริงเมื่อ str1 เหมือนกับ str2**
str1 != str2	**เป็นจริงเมื่อ str1 ไม่เหมือนกับ str2**
Str	**เป็นจริงเมื่อ str ไม่เป็นค่าว่าง**
-n str	**เป็นจริงเมื่อ str มีความยาวมากกว่า 0**
-z str	**เป็นจริงเมื่อ str มีความยาวเป็น 0**
คำสั่งเกี่ยวกับ file	
-d filename	**เป็นจริงเมื่อ filename เป็น directory**
-f filename	**เป็นจริงเมื่อ filename เป็น file**
-r filename	**เป็นจริงเมื่อ filename อ่านได้โดยโปรแกรม**
-w filename	**เป็นจริงเมื่อ filename เขียนได้โดยโปรแกรม**
-x filename	**เป็นจริงเมื่อ filename run ได้โดยโปรแกรม**
-s filename	**เป็นจริงเมื่อ filename มีขนาดไม่เป็น 0**

คำสั่งเกี่ยวกับ Logical อื่นๆ	
!expr	เป็นจริงเมื่อ exp เป็นเท็จ
exp1 –a exp2	เป็นจริงเมื่อ exp1 และ exp2 เป็นจริง
exp1 –o exp2	เป็นจริงเมื่อ exp1 หรือ exp2 เป็นจริง

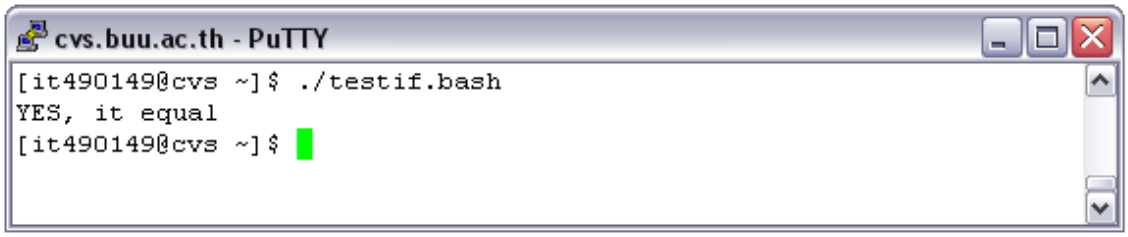
การใช้เงื่อนไข (if)

คำสั่ง if เป็นคำสั่งที่ใช้ในการเปรียบเทียบ

	รูปแบบคำสั่ง if (ใน vi)
Syntax	if [expression] then commands elif [expression] then commands else commands fi
Example	File testif.bash



	ผลลัพธ์ที่ได้



ในตัวอย่างแสดงการเปรียบเทียบตัวอักษรของตัวแปร text1 กับ text2 ว่าเหมือนกันหรือไม่ ถ้าเหมือนกันให้แสดงข้อความว่า " YES, it equal " หรือถ้าต่างกันให้แสดงข้อความว่า " No, it not equal " ทั้งนี้เงื่อนไข if จะมี หรือ ไม่มี elif กับ else ก็ได้

การใช้เงื่อนไข (case หรือ switch)

คำสั่ง case เป็นคำสั่งที่ใช้ในการเปรียบเทียบเช่นเดียวกับคำสั่ง if แต่มีรูปการทำงานที่ต่างกัน

	รูปแบบคำสั่ง case (ใน vi)
Syntax	**case string in** **str1)** **commands;;** **str2)** **commands;;** **str3)** **commands;;** ***)** **commands;;** **esac**
Example	**File testcase.bash**

cvs.cs.buu.ac.th - PuTTY

```
#!/bin/bash
# file: testcase.bash

text="cat"

case $text in
        "ant")
        echo "It is an ant";;
        "bee")
        echo "It is a bee";;
        "cat")
        echo "It is a cat";;
        *)
        echo "I don't Know";;
esac
~
~
~
                                                    15,5          All
```

ผลลัพธ์ที่ได้

cvs.cs.buu.ac.th - PuTTY

```
[it490149@cvs ~]$ ./testcase.bash
It is a cat
[it490149@cvs ~]$
```

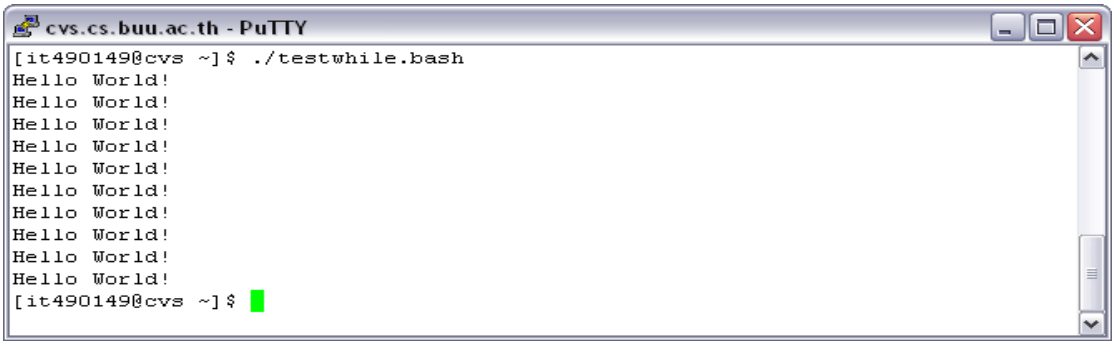
ในตัวอย่างแสดงการเปรียบเทียบตัวอักษรของตัวแปร text ว่าตรงกับ string ที่เป็นเงื่อนไขของ case หรือไม่ ถ้าเหมือนกันก็ให้แสดงข้อความที่ตั้งไว้ในเงื่อนไข

การใช้ loop while

คำสั่ง while เป็นคำสั่งที่ใช้ในการวนรอบการทำงานของโปรแกรม โปรแกรมจะวนรอบจนกว่าเงื่อนไขที่กำหนดไว้จะเป็นเท็จ

	**รูปแบบคำสั่ง while (ใน vi)**
Syntax	**while [expression]** **do** **commands** **…….** **Done**
Example	**File testwhile.bash**

	**ผลลัพธ์ที่ได้**

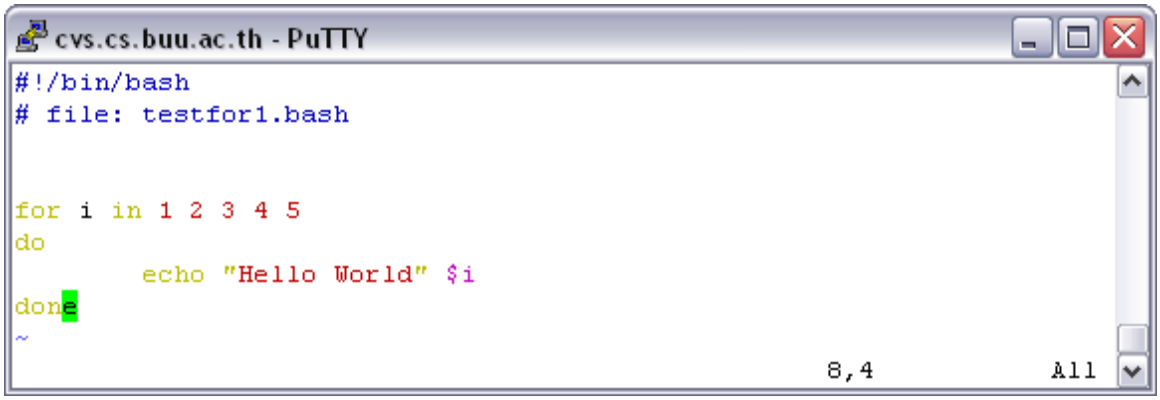


ในตัวอย่างเป็นการแสดงข้อความ “ Hello World ” 10 บรรทัด โดยให้ตัวแปร num มีค่าเป็น 1 ต่อจากนั้นก็เข้าสู่ loop while โดยมีเงื่อนไขว่า $num น้อยกว่าหรือเท่ากับ 10 loop while จะทำการแสดงข้อความ “ Hello World ” โดยวนรอบการทำงานไปเรื่อยๆ และจะทำการเพิ่มค่า num ทีละ 1 ก่อนจะวนรอบใหม่ เมื่อโปรแกรมวน loop ครบตามเงื่อนไขแล้ว จะได้ผลลัพธ์เป็นการแสดงข้อความ “ Hello World ” 10 บรรทัด

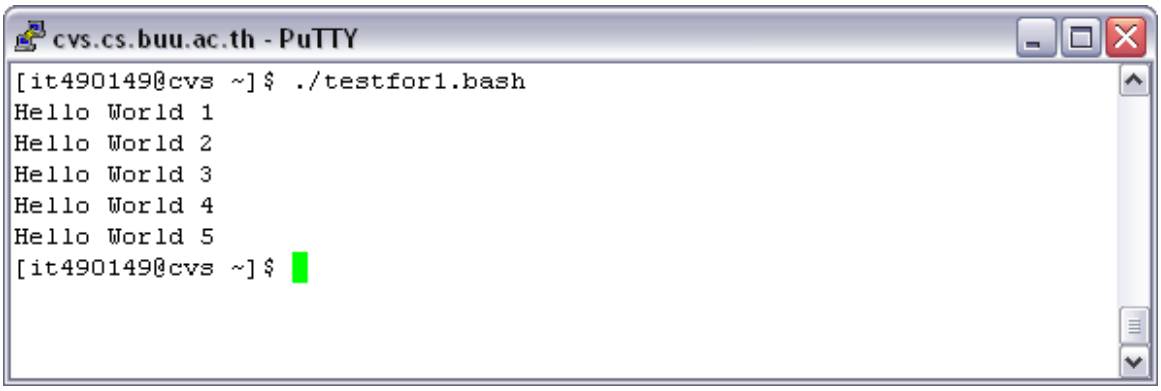
การใช้ loop for

คำสั่ง for เป็นคำสั่งที่มีการทำงานคล้ายคลึงกับ คำสั่ง while แต่จะต่างกันตรงที่การเขียนเงื่อนไขที่ใช้ในการวน loop

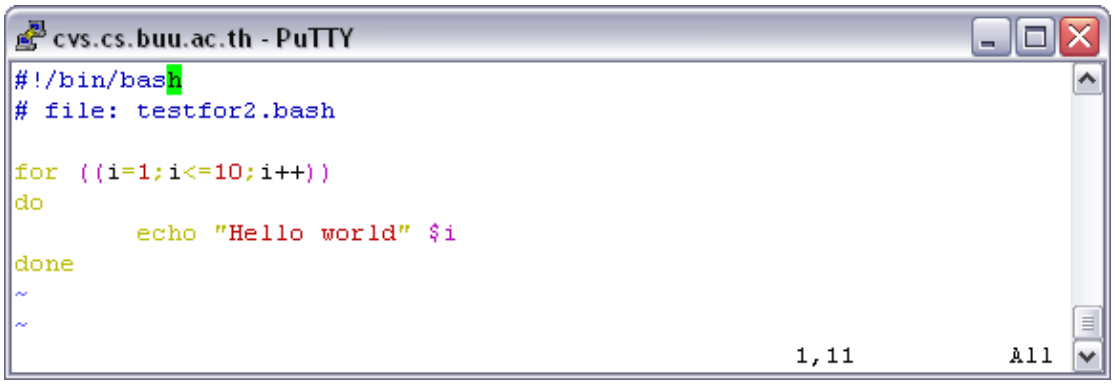
	**รูปแบบคำสั่ง for (ใน vi)**
Syntax	**for var in list** **do** **commands** **…….** **Done**
Example	**File testfor1.bash**



	ผลลัพธ์ที่ได้



ในตัวอย่างเป็นการแสดงข้อความ “Hello World ” ตามด้วยตัวเลข ในเงื่อนไข for นั้น “ i ” จะรับค่าตัวแปรที่อยู่หลัง “ in “ เข้ามาเก็บไว้ในตัวเอง ซึ่งตั้งไว้เป็น “ 1 2 3 4 5 “ มาทีละตัว (จะใช้ช่องว่างในการแบ่งว่ามีกี่ตัว หรือกี่ argumentในเงื่อนไข) loop for จะทำการแสดงข้อความตามจำนวน argument ที่อยู่หลัง “ in ” ซึ่งในโปรแกรมกำหนดเป็น “ 1 2 3 4 5 “ ซึ่งมีอยู่ห้าตัว โดยถูกแบ่งด้วยช่องว่าง เมื่อรันโปรแกรมแล้วจะเห็นได้ว่า ข้อความ “ Hello World ” ตามด้วยจำนวนตัวเลข ถูกแสดงออกมา 5 บรรทัด โดยที่ตัวเลขนั้นมาจากตัวแปร “ $i ” ที่อยู่ในบรรทัด echo “Hello World ” $i นั่นเอง Loop for สามารถเขียนได้อีกอย่างหนึ่งดังนี้



	ผลลัพธ์ที่ได้

จะเห็นได้ว่าเงื่อนไขนั้นสามารถเขียนได้เหมือนในภาษา C เพียงแต่ต้องใส่วงเล็บเพิ่มเข้าไปอีกชั้นหนึ่ง