



Bash Scripting Basics

IT 4423

Unix/Linux Administration

J.G. Zheng
Spring 2012



Overview

◆ Bash scripting language basics

- Script files
- Language elements
 - ◆ Variable
 - ◆ Operators
 - ◆ Control flow
 - ◆ Function

◆ Scripting apps

- Text and number processing
- File processing

Script File

◆ A set of commands grouped together in a file

◆ Bash script file

- .sh (suffix not required)

- **Add "#!/bin/bash" as a directive in the first line**

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

```
var1="morning" #this is a variable
```

```
echo "Good $var1"
```

◆ One statement per line

- ; is optional

- Use ; to separate statements if multiple commands are on the same line

◆ Use # for comments

Variables

◆ Variable naming

- Variable names are case sensitive
- All-caps names typically suggest environment variables or variables read from global configuration files
- Local variables are all-lowercase with components separated by underscores

◆ Assignment

- All variables are of the string data type when assigned

```
#> var1="today"
```

no space around the = symbol

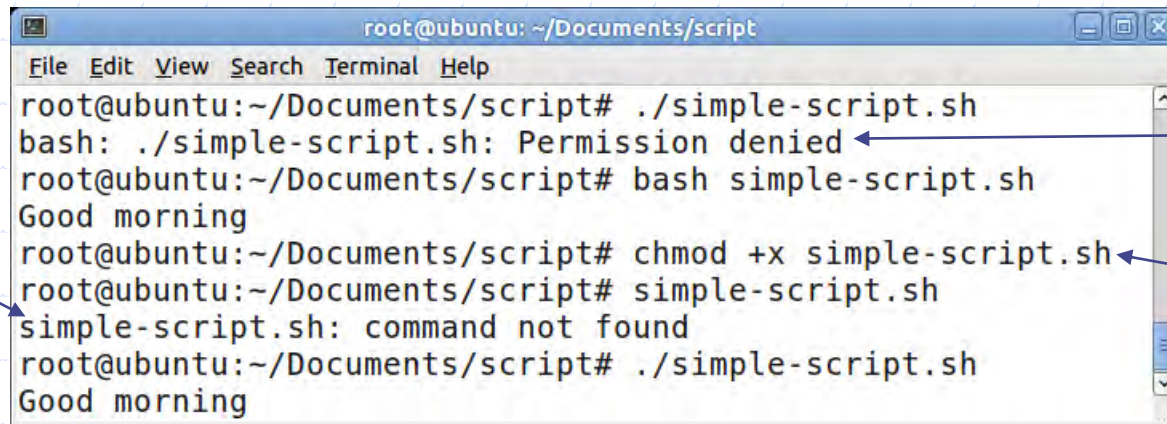
◆ Referencing variables

- Use the "\$"+variable name
- Use {} around variable name optionally

```
#>var1=1000; printf "User input: $var1\n"  
User input: 1000
```

Script Execution

- ◆ Use the bash process to execute scripts
 - `bash [script file name]`
- ◆ Directly running a script file as a separate process
 - Use "`chmod +x`" to allow the file executable



```
root@ubuntu: ~/Documents/script
File Edit View Search Terminal Help
root@ubuntu:~/Documents/script# ./simple-script.sh
bash: ./simple-script.sh: Permission denied
root@ubuntu:~/Documents/script# bash simple-script.sh
Good morning
root@ubuntu:~/Documents/script# chmod +x simple-script.sh
root@ubuntu:~/Documents/script# simple-script.sh
simple-script.sh: command not found
root@ubuntu:~/Documents/script# ./simple-script.sh
Good morning
```

Initially, the file does not have execution permission.

Add execution permission

For security reasons, the current directory is not in the search path. Add `./` to force search the current directory

Script Arguments

- ◆ Script arguments can be supplied at the command line separated by space
- ◆ Arguments are stored in order as \$1, \$2
 - \$0 refers to the command/script name
 - \$# is the number of arguments

```
root@ubuntu: ~/Documents/script
File Edit View Search Terminal Help
root@ubuntu:~/Documents/script# ./script-args.sh
Number of arguments: 0
./script-args.sh

root@ubuntu:~/Documents/script# ./script-args.sh it4423 linux
Number of arguments: 2
./script-args.sh
it4423
linux
root@ubuntu:~/Documents/script#
```

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

```
echo "Number of arguments: $#"  
echo "$0"  
echo "$1"  
echo "$2"
```

Arithmetic Operations

◆ Basic operators

- +, -, *, /, %
- ++, --, +=, etc.

◆ Arithmetic operations

- Use "let"
- Use \$[...]
- Use \$((...))
- Expr

◆ More

- <http://hacktux.com/bash/math>

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

```
var1=5
```

```
var2=10
```

```
var3=var1+var2
```

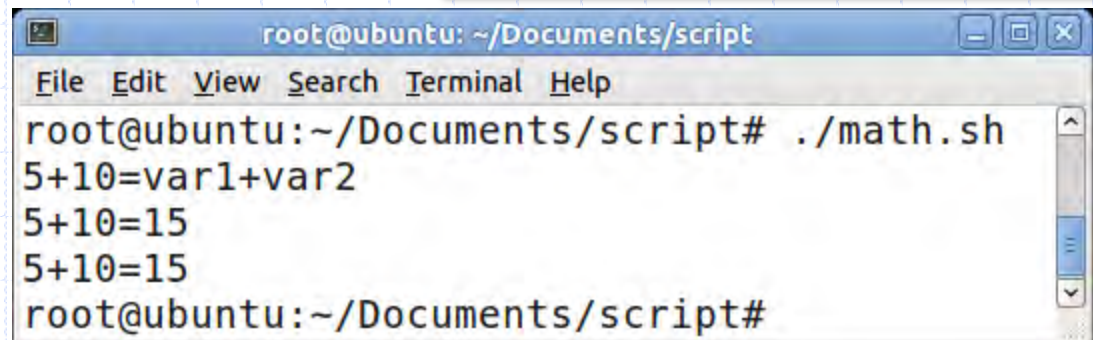
```
echo "$var1+$var2=$var3"
```

```
let var3=var1+var2
```

```
echo "$var1+$var2=$var3"
```

```
var3=$((var1+var2))
```

```
echo "$var1+$var2=$var3"
```



```
root@ubuntu: ~/Documents/script
File Edit View Search Terminal Help
root@ubuntu:~/Documents/script# ./math.sh
5+10=var1+var2
5+10=15
5+10=15
root@ubuntu:~/Documents/script#
```

Comparison Operations

Table 2.2 Elementary bash comparison operators

String	Numeric	True if
<code>x = y</code>	<code>x -eq y</code>	x is equal to y
<code>x != y</code>	<code>x -ne y</code>	x is not equal to y
<code>x < y</code>	<code>x -lt y</code>	x is less than y
<code>x <= y</code>	<code>x -le y</code>	x is less than or equal to y
<code>x > y</code>	<code>x -gt y</code>	x is greater than y
<code>x >= y</code>	<code>x -ge y</code>	x is greater than or equal to y
<code>-n x</code>	-	x is not null
<code>-z x</code>	-	x is null

Use `\` before `>`
and `<`

!!
Leave a space around
`=`; otherwise it is
assignment

Table 2.3 bash file evaluation operators

Operator	True if
<code>-d file</code>	<i>file</i> exists and is a directory
<code>-e file</code>	<i>file</i> exists
<code>-f file</code>	<i>file</i> exists and is a regular file
<code>-r file</code>	You have read permission on <i>file</i>
<code>-s file</code>	<i>file</i> exists and is not empty
<code>-w file</code>	You have write permission on <i>file</i>
<code>file1 -nt file2</code>	<i>file1</i> is newer than <i>file2</i>
<code>file1 -ot file2</code>	<i>file1</i> is older than <i>file2</i>

Flow Control – If

◆ Basic structure

```
if [ ... ]; then
```

```
...
```

```
elif [ ... ]; then
```

```
...
```

```
else
```

```
...
```

```
fi
```

!!
Leave a space
before] and
after [

Use [] for
test
conditions

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

```
var1=5
```

```
var2=5
```

```
if [ $var1 = $var2 ]; then
```

```
    echo "Same"
```

```
else
```

```
    echo "Different"
```

```
fi
```

Nested If

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

```
if [ $# -eq 1 ]  
then
```

```
    if [ -d $1 ]; then
```

```
        echo "$1 exists and is a directory!"
```

```
    elif [ -f $1 ]; then
```

```
        echo "$1 exists and is a file!"
```

```
    else
```

```
        echo "$1 does not exist!"
```

```
    fi
```

```
else
```

```
    if [ -d $PWD ]; then
```

```
        echo "$PWD is a directory"
```

```
    fi
```

```
fi
```

Check
number of
arguments

This script checks if
the provided filename
exists and is a file or a
directory.

Compound Conditions

◆ Use `[[...]]` to evaluate compound conditions

- `&&`: logical AND
- `||`: logical OR

◆ `[...] && [...]`

◆ `[... -a ...]`

- `-0`

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

```
var1=5  
var2=10
```

```
if [[ $var1 -lt 10 && $var1 -gt 4 ]]; then  
    echo "Within range"
```

```
else
```

```
    echo "Out of range"
```

```
fi
```

```
if [[ $var2 -eq 10 || $var2 -eq 20 ]]; then  
    echo "Multiples of 10"
```

```
else
```

```
    echo "Not multiples of 10"
```

```
fi
```

!!
Leave a space
before `]]` and
after `[[`

Flow Control – Case

◆ Basic structure
case \$variable in
 condition1) ...;;
 condition2) ...;;
 *) ... ;;
esac

```
#!/bin/bash  
  
var1=7  
  
case $var1 in  
    5) echo "five";;  
    6) echo "six";;  
    *) echo "other";;  
esac
```

Flow Control – For Loop

◆ Basic structure

```
for (( i=0 ; i < $limit ; i++ ))  
do  
    ...  
done
```

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

```
for (( i=0 ; i < 3 ; i++ ));
```

```
do
```

```
    echo "$i"
```

```
done
```

Flow Control – For ... In Loop

◆ Basic structure

```
for var1 in ... #arguments separated by space
do
    ...
done
```

```
#!/bin/bash

for var1 in 1 2 3 4
do
    echo "Number: $var1"
done
```

Flow Control – While Loop

◆ Basic structure

```
counter=0
```

```
while [ loop condition ]
```

```
do
```

```
...
```

```
=$((counter++))
```

```
done
```

!!
Leave a space
before] and after [

```
#!/bin/bash  
  
counter=0  
while [ $counter -lt 10 ]  
do  
  
    echo "Count: $counter"  
    let counter++  
  
done
```

Flow Control – Until Loop

◆ Basic structure

```
counter=0
until [ loop stop condition ]
do
    ...
    ${counter++}
done
```

!!
Leave a space
before] and after [

```
#!/bin/bash

counter=0
until [ $counter -ge 10 ]
do
    echo "Count: $counter"
    let counter++
done
```

Functions and Parameters

◆ Basics

- Functions in bash do not return values
- Functions have to be declared before they can be called
- Parameters do not need to be declared; they are referred to as \$1, \$2, etc. in the function

```
#!/bin/bash
function add_int
{
    echo "Add numbers:" ${1+}$2;
}
get_diff()
{
    let r=${1-$2}
}
var1=8
var2=5
add_int var1 var2
get_diff var1 var2
echo "Difference: " $r
```

Function parameters

An alternative way to define a function

Pass parameters

A work around to have functions return values

Summary

◆ Key terms

- Scripting mode
- Script file
- Argument
- Function

◆ Bash operators and symbols

- Arithmetical operations: `$((...))` `$[...]` `let` `+` `-` `*` `/` `**` `%`
- Comparison operators: `-eq` `-ne` `-gt` `-lt` `-d` `-e` `-f` `&&` `||`
- Other operators: `;` `\` `#` `$`
- Control flow structures: `if`, `case`, `for`, `for...in`, `while`, `until`
- function, `{ }`, `$#`, `$1`, `$2`, etc.

Good Readings and Resources

◆ Scripting reference

- http://linuxconfig.org/Bash_scripting_Tutorial

◆ Advanced Bash-Scripting Guide

- <http://tldp.org/LDP/abs/html/index.html>