Warning: These notes are not complete, it is a Skelton that will be modified/add-to in the class. If you want to us them for studying, either attend the class or get the completed notes from someone who did

EECS2301

Linux/Unix
part 2

These slides are based on slides by Prof. Wolfgang Stuerzlinger at York University

Special variables

- Special variables starts with $
- $?	The exit status of the last command
- $$	The process id of the shell
- $*	String containing list of all arguments
- $#	Number of argument
- $0	Command line

$* and $@

- Without quotes “” they are the same
- With quotes
  - $* The parameter list becomes a single string
  - S@ each of the parameters is quoted (treated as a single string) unless 2 of the parameters are quoted, they are treated as a single string

$* and $@

```
Set a b c “d e f” g h
for i in $*; do echo $i; done
a
b
c
d
e
f
g
h
```

```
Set a b c “d e f” g h
for i in $@; do echo $i; done
a
b
c
d
e
f
g
h
```

$* and $@

```
Set a b c “d e f” g h
for i in “$*”; do echo $i; done
a b c d e f g h
```

```
Set a b c “d e f” g h
for i in “$@”; do echo $i; done
a
b
c
d e f
g
h
```

Shift

```
#!/bin/sh


echo first arg is $1
shift 1
echo first arg is $1
shift 2
echo first arg is $1
```

```
Shift.sh 1 2 3 4 5 6 7 8 9
First arg is 1
First arg is 2
First arg is 4
```

Special Substitution

- Various special substitutions:
- **${***name***-***word***}** - value of *name* if it exists,
- otherwise “*word*”
- **${***name***+***word***}** - “*word*” if *name* exists, blank otherwise
- **${***name***=***word***}** - if *name* does not exist, sets
- variable *name* to *word*, substitutes value of *name*
- ***${****name***?***word***}** - if *name* does not exist then prints an error (“*word*”) then exits shell - otherwise substitutes value of *name*

Special Substitution

	A B	A B=5	A=4 B	A=4 B=5
${*A***-***B***}**	B		4	4
${*A***+***B***}**			B	B
${*A***=***B***}**	B Now $A=B	B Now $A=B	4	4
${*A***?***B***}**	A: B	A: B	4	4

Echo ${name:?”name is undefined”}

```
name: name is undefined        if not set
Mokhtar                        if set
```

Echo ${name:?}

```
name: parameter null or not set
```

Special substitution

- aboelaze@indigo echo ${v-goodbye}
- goodbye
- aboelaze@indigo v=Hello
- aboelaze@indigo echo ${v-goodbye}
- Hello
- aboelaze@indigo

Read

- So if stdin has '*hello there world*'
- **read a b c**
- (a = 'hello', b = 'there', c = 'world')
- **read a b**
- (a = 'hello', b = 'there world')
- **read a b c d**
- (a = 'hello', b = 'there', c = 'world', d is empty)
- **read –a aa bb cc dd**   #What does it do?

Read

- **read** with just one argument assigns entire line
- **read x**
- This reads a line from stdin and puts it in 'x'.
- **read** is a built-in command with an exit status of 0 on success, or non-zero on failure or EOF
- When reading input, **read** by defaults separates words by space and tab characters
- Can change separator by setting the environment
- variable **IFS**:
- • **IFS=:**

Read

- ***$read a***

  Hello there world
- ***echo $a***

  Hello there world
- ***echo #{a[0]}***

  hello there world
- ***echo ${a[1]}***

  blank

Read

- ***$read –a a***

  Hello there world
- ***echo $a***

  Hello
- ***echo ${a[0]}***

  hello
- ***echo ${a[1]}***

  there

read

- aboelaze@indigo read x
- Hello and goodbye
- aboelaze@indigo echo $x
- Hello and goodbye
- aboelaze@indigo read x y
- hello and goodbye
- aboelaze@indigo echo $x
- hello
- aboelaze@indigo echo $y
- and goodbye
- aboelaze@indigo

Arithmetic operations

- Does this work?
- x=5
- y=$x+1 ## echo $y → 5+1
- y=$x + 1 ## + command not found
- $ z=5
- $ z=`expr $z+1` Need spaces around + sign.
- $ echo $z 5+1
- $ z=`expr $z + 1`
- $ echo $z 6

Arithmetic Operations

- **expr** command supports only integer arithmetic.
- **sum=`expr $a + $b`** SPACES !@#$
- **diff=`expr $a - $b`**
- **prod=`expr $a \* $b`**
- **quot=`expr $a / $b`**
- **remind=`expr $a % $b`**

Arithmetic Operations

- Bash has built in support for arithmetic integer operations, similar to C operators
  - Either **let** or **$((…))**
- **let a=16+5**
- **let "n=$n-1"**
- **echo $(($a*$b))** # no quote for * is necessary
- **echo $(($a--))**
- **echo $(($a**3))** # exponentiation

Arithmetic -- FP

- N=`echo “scale=3; 13 / 2” |bc`
- echo $N

- n=`bc << EOF
› scale=3
› 13/2
› EOF`
- echo $n

Set

- set command re-sets positional parameters (arguments)
- **set apple banana cherry**
- **echo $1, $2, $3**
- **set `date`**
- **echo $1, $2**
- **a='hello world!'**

Testing

- To test various conditions, we use test command:
- for string = , !=, -z string (length is zero) –n string (length is non zero,]), and string (not null).

```
#!/bin/sh
test $1 != Tom
echo $?
```

```
#!/bin/sh
[ $1 != Tom ]
echo $?
```

Testing

- **[-d *file*]** is identical to **test -d *file***
- int1 −eq int2
- int1 −ne int2
- int1 −gt int2
- int1 −gt int2
- int1 −lt int2
- int1 −le int2

Testing

- **[-d file]** - if it is a directory ?
- **[-f file]** - a regular file ?
- **[-r file]** - the file readable ?
- **[-w file]** - the file writable ?
- **[-x file]** - the file executable ?
- **[-s file]** - the file has non-zero size ?
- **[-L file]** - a symbolic link ?
- **[-u file]** - the file has **suid** bit ?

Logical Operators

- **-a** logical AND
- **-o** logical OR
- **!** logical NOT
- **[-w res.txt –a –w score.txt]**
- **[-x op1 –o –x op2]**
- **[! –d Tmp]**
- The Bash extended test operator **[[…]]** allows
- usage of **&&,||,>,<** in an expression.
- **[[$a>$b]]**

Test Subtleties

- The following is bad practice:
- **[$var = rightvalue] && echo OK**
- **[$OSTYPE = "linux"] && echo**
- **Running in Linux**
- Why?

Test Subtleties

- What if “**$var**” is blank? After substitution we get:
- **[= rightvalue] && echo OK**
- which is a “test” syntax error!

Test Subtleties

- An old sh programmer’s trick:
- **["X$var" = "Xrightvalue"] && echo OK**
- **["X$OSTYPE" = "Xlinux"] && echo**
- **Running in Linux**
- Protects against unusual variable values

Test Subtleties

- **[-d $dir] || mkdir $dir**
- creates the directory $dir if it does not already exist

Testing

- **a=010**
- **b=10**
- **[$a = $b]**
- # FALSE as two different strings
- **[$a -eq $b]**
- # TRUE as two numbers

Example

```
read marks
if [ $marks –ge 80 ]; then
    grade=A
elif [ $marks –ge 70 ]; then
    grade=B
elif [ $marks –ge 60 ]; then
    grade=C
else
    grade=D
fi
echo $grade
```

grade = c ?????? testing? Note that there is no compilation

Example

```
myprogram < data.in > result.out
if [ -s result.out ]; then
    echo "Output generated !!"
else
    echo "empty output!!"
fi
```