Creating/Assigning a Variable

- **Name=value**
  - Variable created if it does not exist
  - Need quotes if value contains spaces
    - e.g. `x="one two"`  
  - Access variable with `$` in front
    - e.g. `echo $x`
    - also `echo ${x}3`
Access Variations

• if *name* not set, replace with *word*
  
  • `${name-word}`

• Variable *x* is set, so we see it displayed

  $ echo ${x-three}

  *one two*

• Variable *x* is not set, so we see “three”

  $ echo ${x-three}

  *three*
Access Variations

- if `name` is set, replace with `word`
  - `${name+word}`
- Variable `x` is set
  
  ```
  $ echo ${x+three}
  three
  ```
- Variable `z` is not set, so nothing is printed
  
  ```
  $ echo ${z+three}
  ```
Access Variations

• if name is not set, assign with word
  
  * ${name=word}

• Variable x is set, and printed

  $ echo ${x=three}

  one two

• Variable z is not set, so now it is set to three

  $ echo ${z=three}

  three

  $ echo $z

  three
Access Variations

• if name is set, return it; otherwise, print word to standard error (and quit)
  
  • ${name?word}$

• Variable x is set

  $ echo ${x?three}$

  one two

• Variable w is not set

  $ echo ${w?three}$

  -bash: w: three
Reading from Standard Input

- Command read
- Reads 1 line
- Examples

```bash
$ read v1 v2
one two
$ echo v1
one
$ echo v2
two
```
```bash
$ read v1 v2
one two three four
$ echo v1
one
$ echo v2
two three four
```
Example Reading Multiple Lines

```bash
$ cat read.sh
#!/bin/sh
# read multiple lines
read v1
read v2
echo "you said $v1"
echo "then you said $v2"

$ ./readme.sh
one two
three four five
you said one two
then you said three four five
```
Exporting Variables

- Command export
- Makes variables available in environment
- e.g. export x

    $ v1="one two"
    $ export v1
    $ sh
    sh-3.1$ echo v1
    one two
### Predefined Local Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$@$</td>
<td>an individually quoted list of all the positional parameters</td>
</tr>
<tr>
<td>$#$</td>
<td>the number of positional parameters</td>
</tr>
<tr>
<td>$? ?</td>
<td>the exit value of the last command</td>
</tr>
<tr>
<td>$!</td>
<td>the process ID of the last background command</td>
</tr>
<tr>
<td>$-$</td>
<td>the current shell options assigned from the command line or by the built-in set command</td>
</tr>
<tr>
<td>$$</td>
<td>the process ID of the shell in use</td>
</tr>
</tbody>
</table>
Predefined Locals Variables

$ cat predefined.sh
echo You passed $# parameters
echo These are: “$@”
echo process ID of last background process = $!
echo process ID of this shell = $$
notAcommand
echo Last command returned with $? as the status.

$ ./predefined.sh one two three four
You pressed 4 parameters
There are: one two three four
process ID of last background process =
process ID of this shell = 21712
./predefined.sh: line 7: notAcommand: command not found
Last command returned with 127 as the status.
Arithmetic

- Bourne shell does not directly do math
- Command `expr` evaluates expressions
  - Supports
    - Multiplication (`*`), Division (`/`), Remainder (`%`)
    - Add (`+`), Subtract (`-`)
    - Equal (`=`), Not Equal (`!=`)
    - Less (`<`), Less/Eq (`<=`), Greater (`>`), Greater/Eq (`>=`)
    - And (`&`), Or (`|`)
    - Index (locate substring)
    - Match
expo Command

• `expr` also evaluates expressions
  • Locate substring
    • `Index` string charList
  • Test for a match (returns 0 or length)
    • `Match` string regex
  • Get substring
    • `Subset` string start length
  • Length of sting
    • `Length` string
expr Examples

$ x=1
$ x=`expr $x + 1`
$ echo $x
2

$ x=`expr 2 + 3 \* 5`
$ echo $x
17

$ echo `expr substr "donkey" 4 3`
key
test Command

• Command *test expression*
  • returns 0 if true
  • returns nonzero if false

• Examples
  • File exists: -e filename
  • File has >= 1 size: -s filename

• More expressions in manual pages (*man test*)
Case Structure

case expression in
    pattern1)
    commands
    ;;
pattern2 | pattern3)
    commands
    ;;
...
*)
    commands
    ;;
esac
Case Structure Example

```bash
$ cat test case.sh

```
```
echo "Type out the word for 1 or 2:"
read v1
case $v1 in
  [Oo]ne)
    echo "You entered 1"
  ;;
  [Tt]wo)
    echo "You entered 2"
  ;;
  *)
    echo "Sorry"
  ;;
esac
```

```
$ ./testcase.sh
Type out the word for 1 or 2:
two
You entered 2
```
```
$ ./testcase.sh
Type out the word for 1 or 2:
Two
You entered 2
```
```
$ ./testcase.sh
Type out the word for 1 or 2:
three
Sorry
```
For Loop

- Loop where `name` gets each value in `word`, in turn
- Uses `@` if no word given
- End loop: break
- Go to next iteration: continue

```bash
for name in {word}*
do
    command list
done
```
For Loop Example

$ cat testfor.sh

params=$@
for value in params
do
echo param: $value
done

$.testfor.sh one two three
param: one
param: two
param: three
if statement

- Execute list1
- if last command succeeds, do list2
- if last command (of list1) fails, try list3
- ...

```bash
if list1
  then
    list2
  elif list3
    then
      list4
  else
    list5
fi
```
if Example

$ cat testif.sh
echo "enter a word:"
read v1
if [ -e $v1 ]
then
    echo "A file by that name exists."
else
    echo "No file by that name exists."
fi

$ ./testif.sh
enter a word:
dummy
A file by that name exists.

$ ./testif.sh
enter a word:
two
No file by that name exists.
Responding to a Signal

- Command `trap [[command]{signal}+]`
- Executes command when signal is received
- Signal detail in manual page (man signal)
- Example: `trap 'echo CTRL-C;exit 1' 2`
  - When user types Control-C (signal 2)
  - print “CTRL-C”
  - Exit with status
Until..do..done

- Keep doing list2 until list1 return 0
- Otherwise, do commands in list2
- End loop: break
- Go to next iteration: continue

```
  until list1
do
    list2
  done
```
Until .. do.. done Example

$ cat testuntil.sh
x=1
until [ $x -gt 3 ]
do
  echo x = $x
  x=`expr $x + 1`
done

$ ./testuntil.sh
x = 1
x = 2
x = 2
x = 3
While Loop

- Execute list2 as long as the last command of list1 succeeds
- End loop: break
- Go to next Iteration: continue

```bash
while list1
do
  list2
done
```
While Loop Example

$ cat testwhile.sh
x=1
while [ $x -lt 4 ]
  do
    echo x = $x, less than four
    x=`expr $x + 1`
  done

$ ./testwhile.sh
x = 1, less than four
x = 2, less than four
x = 3, less than four
x = 4, not less than four
null Command

• The `null` command does nothing

• Used in case statements
  • To handle a particular case by doing nothing
Grouping Commands

- Use parenthesis to group commands
  - Executed as a subshell
  - Redirect-able as a group
  - Pipe-able as a group
Review

• Variable assignment and access
• Reading standard input
• Arithmetic and pattern matching (expr)
• Control structures (case, for, is, while, until)