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

CSE2301

Unix/Linux Introduction

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

Introduction

- In this part, we introduce
  - OS (Linux)
  - File system
  - Shell commands
  - Pattern matching
  - Shell programming

Unix

- What does an OS do?
  - File management
  - Scheduling
  - Memory management
  - I/O management
- Examples

Unix

- OS includes
- Kernel: Performs key OS functions
- System programs: various tools
- Shell: Interface to the user

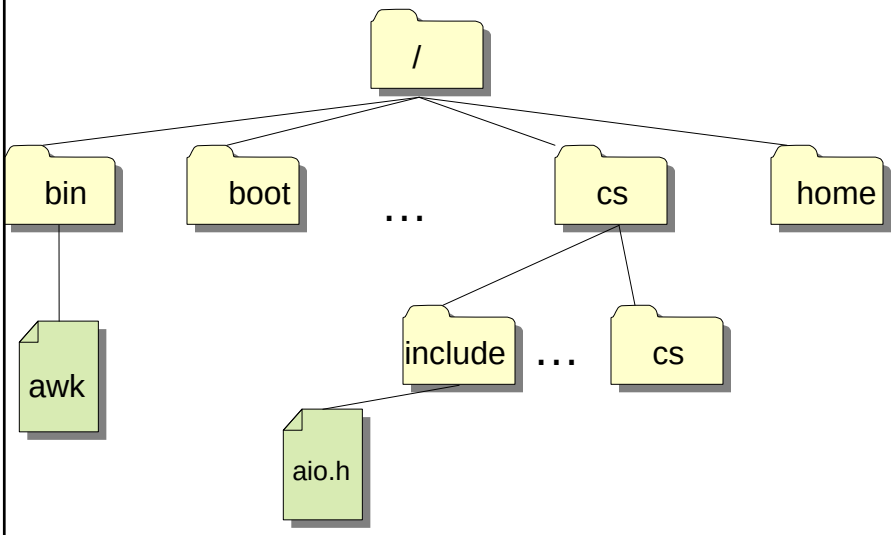
Processes

- Each program running is called a process
- Each process has its own identification PID
- If the program is running twice, even by the same user, these are 2 different processes.

File System

- In Unix, the files are organized into a tree structure with a root named by the character ’/’.
- Everything in the file system is a file or subdirectory

Our File System



File System

- File names could be relative (with respect to the current directory) or using full path name (relative to /) for example aio.h or /cs/include/aio.h
- Your home directory is ~username, so in my case ~aboelaze/test.c is equivalent to /cs/home/aboelaze/test.c

Devices

- /dev contains devices, just like any other file (fopen, fread, fwrite, ..) but it communicate with a device.
- /dev/tty
- /dev/null
- /dev/zero

Unix Commands

- ls cp mv rm mkdir cd pwd cat less more head tail ….
- bg, fg, CTRL-C, CTRL-Z
- kill ps od diff ln echo …
- Redirection and pipes Examples

- tigger 215 % ls –las
- total 44
- 4 drwx------ 2 aboelaze faculty 4096 Nov 29 13:44 ./
- 4 drwx------ 9 aboelaze faculty 4096 Nov 29 14:47 ../
- 4 -rw------- 1 aboelaze faculty 184 Nov 18 13:30 data
- 4 -rw------- 1 aboelaze faculty 23 Nov 28 19:52 file1
- 4 -rw------- 1 aboelaze faculty 24 Nov 28 19:52 file2
- 4 -rw------- 1 aboelaze faculty 481 Nov 29 12:27 mergefiles.awk
- 4 -rw------- 1 aboelaze faculty 178 Nov 28 19:32 p1
- 4 -rw------- 1 aboelaze faculty 1245 Nov 18 13:29 prchecks.awk
- 4 -rw------- 1 aboelaze faculty 83 Nov 14 17:46 t
- 4 -rwx------ 1 aboelaze faculty 35 Nov 21 13:08 test.sh*
- 4 -rw------- 1 aboelaze faculty 50 Nov 1 18:31 unmatched
- chmod 744 file What does it mean?
- chmod [ugo][+-][rwx] chmod ug+rw p1

Shell Pattern Matching--Wild Cards

- The character * matches any string of characters
- ? Matches a single character
- [0-9] matches any digit
- [a-z] matches any small case letter
- \c matches c only
- a|b matches a or b **in case expression only**

Shell Variables

- set x = 3 -- csh
- x=3 -- sh (no spaces around the “=“)
- echo x
- echo $x what is the difference
- B=5 C=3 D=2 -- That is O.K.
- Valid variables begin with a letter, contains letters, numbers and _ a5_6

PATH path

- The shell searches in PATH looking for the command you typed
- echo $PATH .:/usr/local/bin:/usr/ucb: /usr/bin /usr/etc:/etc:/bin:/usr/bin/X11
- set path = ($path /a/b/c) --csh
- PATH=$PATH:/a/b/c --sh
- Aliases and startup files

Shell scripting

```
#!/cs/local/bin/sh
echo “Hello World”
```

```
tigger 397 % script1
Hello World
tigger 398 %
```

```
echo  -n “Hello
World”
```

```
tigger 393 % script1
Hello Worldtigger 394 %
```

```
#!/cs/local/bin/sh
echo "Now I will guess your OS"
echo  -n "Your OS is : "
uname
```

```
tigger 399 % script1
Now I will guess your OS
Your OS is : Linux
tigger 400 %
```

Shell Scripting

```
#!/cs/local/bin/sh
echo -n "Please enter your first name : "
read FNAME
echo -n "Last name pelase : "
read LNAME
MESSAGE=" Your name is : $LNAME , $FNAME"
echo "$MESSAGE"
```

```
tigger 439 % script3
Please enter your first name : Mokhtar
Last name pelase : Aboelaze
 Your name is : Aboelaze , Mokhtar
```

Shell Scripting

```
#!/cs/local/bin/sh
read FNAME
echo "1-> $FNAME123"
echo "2-> ${FNAME}123"
```

```
tigger 454 % script4
abcd
1->
2-> abcd123
tigger 455 %
```

Shell Scripting

```
# Set the initial value.
myvar=abc
echo "Test 1 ======"
echo $myvar        # abc
echo ${myvar}      # same as above, abc
echo {$myvar}      # {abc}
echo "Test 2 ======"
echo myvar         # Just the text myvar
echo "myvar"       # Just the text myvar
echo "$myvar"      # abc
echo "\$myvar"     # $myvar
echo "Test 3 ======"
echo $myvardef     # Empty line
echo ${myvar}def   # abcdef
```

```
$ sh var_refs
Test 1 ======
abc
abc
{abc}

Test 2 ======
myvar
myvar
abc
$myvar
Test 3 ======

abcdef
```

Shell Scripting

```
echo "Test 4 ======"
echo $myvar$myvar      # abcabc
echo ${myvar}${myvar}  # abcabc
echo "Test 5 ======"
# Reset variable value, with spaces
myvar=" a   b    c"
echo "$myvar"         # a    b    c
echo $myvar           # a b c
```

```
Test 4 ======
abcabc
abcabc
Test 5 ======
a    b    c
a b c
```

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

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

- 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

- **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

- 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?
- **LOOP=0**
- **LOOP=$LOOP+1 ## does not work**
- In /bin/sh, must use **expr** and back-quotes ``!
- **LOOP=0**
- **LOOP=`expr $LOOP + 1`**
  - increment loop counter

Arithmetic Operations

- **expr** command supports only integer arithmetic.
- **sum=`expr $a + $b`**
- **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

Set

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

Testing

- To test various conditions, we use test command:
- **test** *arg* **=** *arg* ('=' compares strings in Bourne shell)
- **test** *arg* **!=** *arg*
- This commands sets the exit status
- Many other operators
- **test -d** *file*
- - true if *file* is a directory

Testing

- In sh, there is a special “shortcut” syntax for the test command:
- **[-d** *file* **]**
- is identical to **test -d** *file*
- Works for all test options (not just -d)
- This form is more commonly used than ‘test’

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 "**$var**" is blank? After substitution we get:
- **[= rightvalue] && echo OK**
- which is a "test" syntax error!
- What if "**$var**" is "**-d**"? After substitution:
- **[-d = rightvalue] && echo OK**
- which does the wrong test!

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

- **=** - the two strings are equal ?
- **!=** - the two string are not equal ?
- **-z** - the length of string is zero ?
- **-n** - the length of string is nonzero ?
- **[$EDITOR = "vi"]**
- **[$USER = "Tom"]**
- **[-z $EDITOR]**
- **[$EDITOR] # if string is null**

Testing

- **-eq** - two numbers (in strings) are equal?
- **-ne** - two numbers are not equal?
- **-gt** - first number is greater than second
- **-lt** - first number is less than second
- **-le** - first number is less than or equal to
- second
- **-ge** - first number is greater than or equal to
- second

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

Example

```
for SCRIPT in /path/to/scripts/dir/*
do
        if [ -f $SCRIPT -a -x $SCRIPT ]
        then
            $SCRIPT
        fi
done
```

Testing

- #!/usr/bin/env bash
- # cookbook filename: strvsnum
- #
- # the old string vs. numeric comparison dilemma
- #
- VAR1=" 05 "
- VAR2="5“
- printf "%s" "do they -eq as equal? "
- if ["$VAR1" -eq "$VAR2"]
- then
- echo YES
- else
- echo NO
- fi
- printf "%s" "do they = as equal? "
- if ["$VAR1" = "$VAR2"]
- then
- echo YES
- else
- echo NO
- fi

```
$ bash strvsnum
do they -eq as equal? YES
do they = as equal? NO
$
```

Testing– Dealing with failures

- You want to do 2 commands, but the second one depends on the success of the first `cd testdir then rm *.dat`
- One solution is to use the double ampersand operator (run the second only if the first succeed
- `$cd testdir && rm *.dat` OR
- `cd testdir`
- `if(($?)); then rem *.dat; fi`

Testing– Dealing with failures

- Another way is to set the –e option exit the first time encounter error (i.e. a non zero exit status) from any command in the script except while loops and if statement
- set –e
- cd testdir
- rm *.dat

Conditions

- We want to do more than just execute other
- programs
- *command1* **&&** *command2*
- executes command1 - if command1 has an exit
- status of 0 (true), then command2 is executed
- Like C, this is short-circuiting

Conditions

- We have an ‘or’ operator as well:
- *command1* **||** *command2*
- executes command1 - if command1 has an exit
- status of non-zero (false), then command2 is
- executed
- Note that there is no space between the characters in ‘||’ and ‘&&’

Testing– Dealing with failures

- Another way of doing it
- cmd || printf “%b” “cmd failed \n” How does that work?
- cmd || { printf “%b” “cmd failed \n”; exit;}