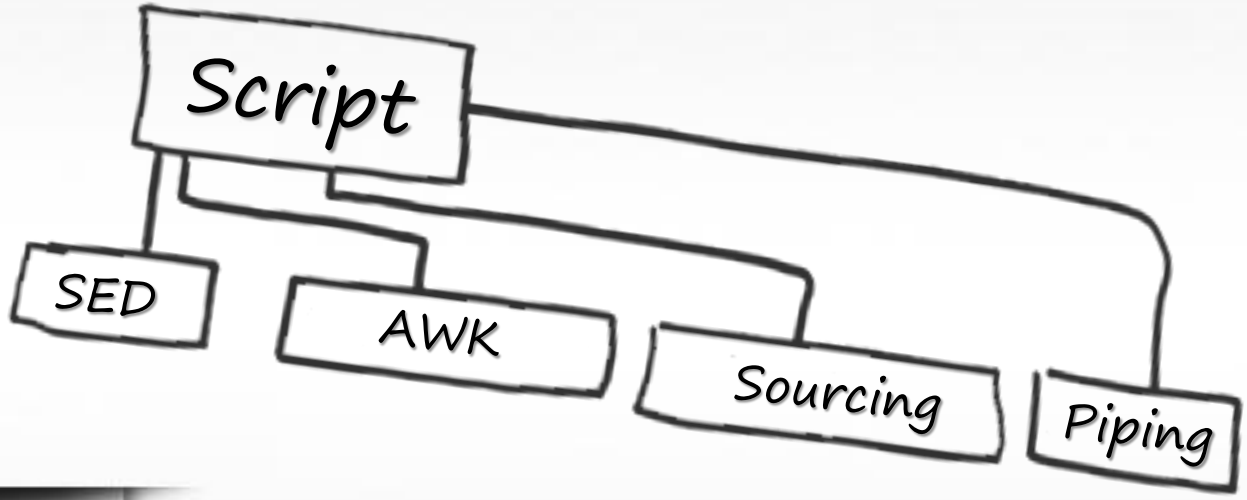
NOW

Shell Scripting



OPEN SOURCE DEPARTMENT

Information Technology Institute

Course Materials

You can access the course materials via this link

http://goo.gl/fbyDAo

Day 1 Contents

- History
- Directory Hierarchy
- Redirection & Piping
- Processes
- sed
- AWK

Introduction

- The UNIX® operating system was developed originally at AT&T Bell Laboratories in 1969.
- It was created as a tool set by programmers for programmers.
- The early source code was made available to universities all over the country.

Introduction

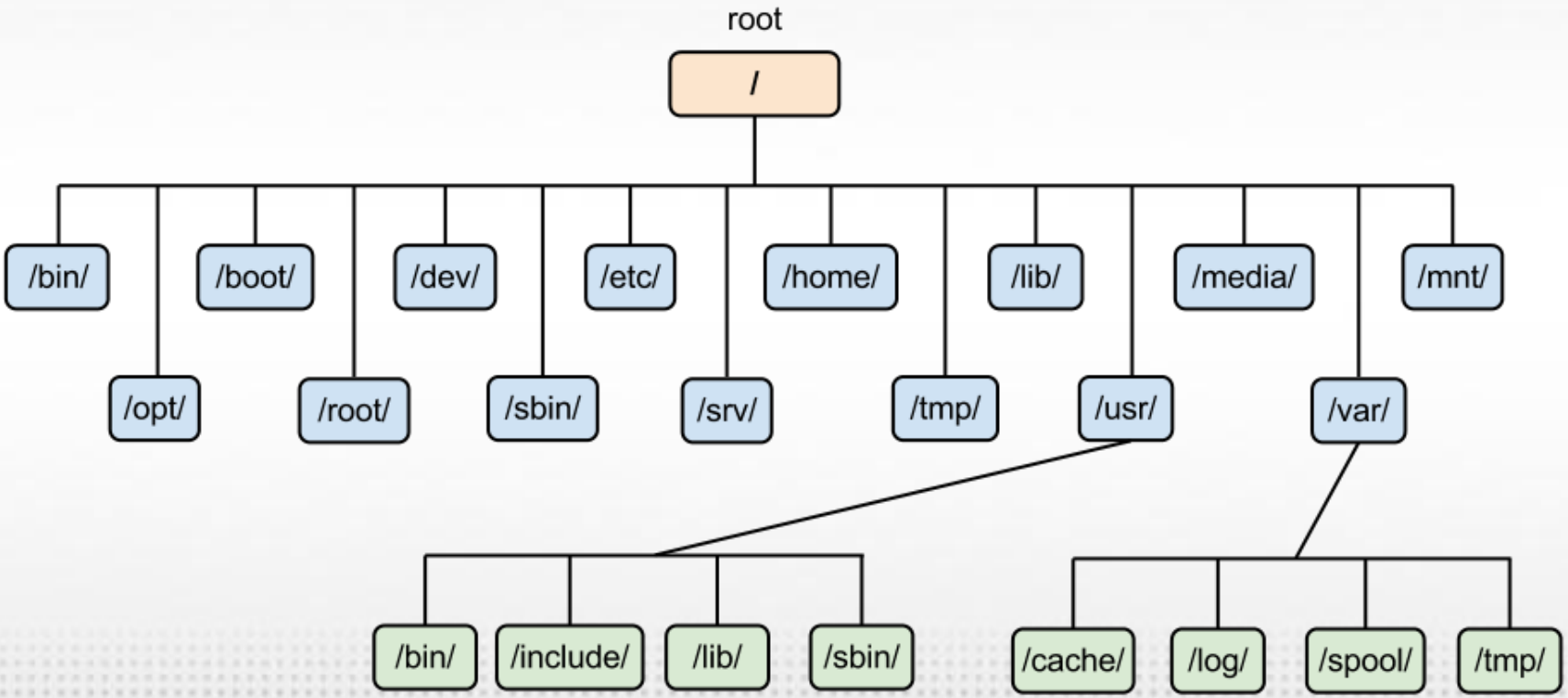
- Programmers at the University of California at Berkeley made significant modifications to the original source code and called the resulting operating system the Berkeley Software Distribution (BSD) UNIX.

- At that time, AT&T's version of the UNIX environment was known as System V.

Unix Essential concepts

- Directory Hierarchy
- Piping and Redirection
- Processes

Directory Hierarchy



Directory Hierarchy

```
 pwd
/home/user1/dir1
 cd ..
 pwd
/home/user1
 cd ../..
 pwd
/
 cd
 cd dir1
 pwd
/home/user1/dir1
 cd ../dir2
 pwd
/home/user1/dir2
```

Directory Hierarchy

- To return to home directory

```
cd
pwd
```

/home/user1

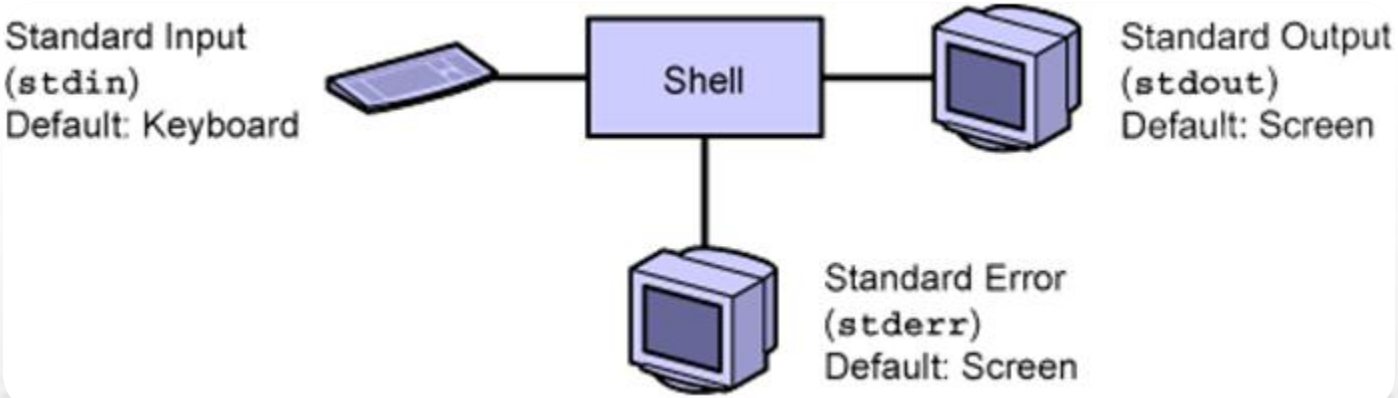
```
cd /home/user1
cd ~user1
pwd
```

/home/user1

```
cd ~/dir1/d1
```

Piping and Redirection

- Types of metacharacters
  - **Redirection** metacharacters
    - You can redirect input to and output from commands by using redirection.



Piping and Redirection

- Each process that the shell creates, works with file descriptors
- File descriptors determine where the input to the command originates and where the output and error messages are sent
  - 0: stdin
  - 1: stdout
  - 2: stderr

Piping and Redirection

- Example

  `cat` (Read from stdin)
  First line (Read from stdin)
  First line (Write to stdout)
  What's going on? (Read from stdin)
  What's going on? (Write to stdout)
  `Control-d` (Read from stdin)

- The default action of the standard input, standard output, and standard error within the shell can be modified by redirecting `stdin`, `stdout`, and `stderr`.

Piping and Redirection

- Redirecting Standard Input

```
command < filename
```

Or

```
command 0< filename
```

- Redirecting Standard Output

```
command > filename
```

Or

```
Command 1> filename
command >> filename
```

- Redirecting Standard Error

```
command 2>/dev/null
command 1> filename 2>&1
```

Piping and Redirection

- The Pipe character redirect the standard output to the standard input of another command

```
command | command
```

- Example

```
who| wc -l
        35
```

Piping and Redirection

```
ls -F /etc | grep "/"
```

X11/

acct/

apache/

apache2/

apoc/

```
head -10 file1 | tail -3 | lp
```

Piping and Redirection

```
ls -F /etc | grep "/"
```

X11/

acct/

apache/

apache2/

apoc/

```
head -10 file1 | tail -3 | lp
```

Processes

- Every program you run creates a process. For example
  - Shell
  - Command
  - An application
- System starts processes called daemons which are processes that run in the background and provide services
- Every processes has a PID
- When a process creates another, the first is the parent of the new process. The new process is called the child process. Parent waits for her child to finish

Viewing a Process

- Process status command

```
ps option(s)
```

- Output
  - PID
  - TTY -> terminal identifier
  - Execution time
  - Command name
- Options
  - -e: all system processes
  - -f: full information

Viewing a Process

- Example

```
ps -ef
UID  PID  PPID  C  STIME  TTY  TIME  CMD
root 0    0    0  Oct 23 pts/6 0:18  -ksh
root 0    0    0  Oct 23 pts/6 0:30  –ps
```

- Where
  - C: CPU utilization for scheduling
  - STIME: Time process started
  - TIME: Execution time for the process

Searching for a Specific Process

- Using ps and grep commands

```
ps –e | grep lp
```

- Using pgrep command

```
pgrep option(s) pattern
pgrep lp
```

- Options
  - -x: exact match
  - -u uid: processes for a specific user
  - -l: display the name with pid

Sending a Signal to a Process

- A signal is a message sent to a process to perform a certain action
- Signals are identified by a signal number and a signal name, and has an associated action.

Signal #	Signal Name	Event	Default Response
15	SIGTERM	Terminate	Exit
9	SIGKILL	KILL	EXIT

Sending a Signal to a Process

- Using kill command
- `kill [-signal] PIDs`
- Examples

```
pgrep -l mail
215 sendmail
12047 dtmail
kill 12047
pgrep -l mail
215 sendmail
```

Sending a Signal to a Process

- Using pkill command

```
pkill [-signal] process_name
```

- Example

```
pkill -9 dtmail
```

Managing a Job

- A job is a process that shell can manage
- Three types of jobs
  - Foreground jobs
  - Background jobs
  - Stopped jobs
- Running a job in the background
- Example

```
sleep 500 &
[1]  3028

[1] + Done
```

Managing a Job

- Listing Current Jobs

```
jobs
```

[1] + Running sleep 500&

- Bringing background job into foreground

```
fg %1
```

sleep 500

- Sending foreground job into background

```
sleep 500
```

^Z[1] + Stopped (SIGTSTP) sleep 500

Managing a Job

```
jobs
```

[1] + Stopped (SIGTSTP) sleep 500

```
bg %1
```

[1] sleep 500&

```
jobs
```

[1] + Running sleep 500

Managing a Job

- Stopping a background job

```
jobs
```
[1] + Running sleep 500&

```
kill -STOP %1
```

```
jobs
```
[1] + Stopped (SIGSTOP) sleep 500&

```
 kill %1
```
[1] + Terminated sleep 500&

```
jobs
```

Useful Commands

`split`

- split a file into n line chunks

`diff`

- compare two files

`line`

- read the first line

Examples

```
$ split -10 /etc/passwd
$ls
```

- xaa xad
- xab xae
- xac xaf

```
$ line < /etc/passwd
```

- root:x:0:1:Super-User:/:/sbin/sh

The Streamlined Editor

- What is sed?
- How sed works
- Addressing
- Commands
- Examples

What is sed?

- It is a streamline, non-interactive editor.
- It performs the same kind of tasks as in vi.
- It doesn’t change your file unless the output is saved with shell redirection.

How does sed Work?

- The sed editor process a file (input) one line at a time and sends its output to the screen.

- The sed stores the line it process in a buffer and once processing is finished the line is sent to the screen (unless command was delete) and the next line is read and the process is repeated until the last line is reached.

Addressing

- Addressing is used to determine which lines to be edited.
- The addressing format can be
  - Number
  - Regular expression
  - Both

* Number represents a line number.

Commands

- The sed commands tell sed what to do with the line:
  - Print it
  - Remove it
  - Change it
- The sed format

```
sed 'command' filename
```

Examples

- To print lines contain the pattern root

```
$sed '/root/p' myfile
```

sherine
maha
root
root
user

- To suppresses the default behavior of the sed

```
$sed -n '/root/p' myfile
```

root

Examples

- To print lines from maha to root

```
$sed –n ‘/maha/,/root/p’ myfile
```

maha
root

- To print lines from 2 to the line that begins with us

```
$sed –n ‘2,/^us/p’ myfile
```

maha
root
user

Examples

- To delete the third line

```
$sed ‘3d’ myfile
sherine
maha
user
```

- To delete the last line

```
$sed ‘$d’ myfile
sherine
maha
root
```

Examples

- To delete lines from 1 to 3

```
$sed '1,3d' myfile
```

user

- To delete from line 3 to the end

```
$sed '3,$d' myfile
```

sherine
maha

- To delete lines containing root pattern

```
$sed '/root/d' myfile
```

sherine
maha
user

Examples

- To substitute islam by iaskar

```
$sed 's/sherine/sbahader/g' myfile
sbahader
maha
root
user

$sed –n 's/sherine/sbahader/gp'
myfile
sbahader
```

Examples

- To issue multi command

```
$sed –e ‘2d’ –e
‘s/sherine/sbahader/g’ myfile
```

sbahader

root

user

The AWK Utility

- What is AWK?
- What does AWK stands for?
- The awk's format
- Records and Fields
- Examples
- BEGIN Pattern
- END Pattern
- Conditional Expressions
- Loops
- Examples

What is AWK?

- awk is a programming language used for manipulating data and generating reports.

- awk scans a file line by line, searching for lines that match a specified pattern performing selected actions

What does AWK stands for?

- awk stands for the first initials in the last names of each authors of the language, Alfred Aho, Peter Weinberger, and Brian Kernighan

Awk's Format

- The awk program consists of
  - awk command
  - Program instructions enclosed in quotes
  - Input file or default stdin.

```
$awk 'instructions' inputfile
```

Records and fields

- By default, each line is called a record and terminated with a new line.
- Record separators are by default carriage return, stored in a built-in variables `ORS` and `RS`.
- The `$0` variable is the entire record.
- The `NR` variable is the record number.

Records and fields

- Each record consists of words called fields which by default separated by white spaces.

- `NF` variables contains the number of fields in a record
- `FS` variable holds the input field separator, space/tab is the default.

Examples

- To print the first field of the file, but as the default delimiter is white space, you have to specify the delimiter

```
$awk –F: ‘{print $1}’ /etc/passwd
```

root
daemon
sherine
…

```
$awk –F: ‘{print “Logname:”,$1}’ /etc/passwd
```

Logname:root
Logname:daemon
Logname:sherine
…

Examples

- To display the whole file (cat)

```
$awk '{print $0}' /etc/passwd
```

root:x:0:1:Super-user:/:/sbin/sh

…

- To display the file numbered (cat -n)

```
$awk '{print NR,$0}' /etc/passwd
```

1 root:x:0:1:Super-user:/:/sbin/sh

…

- To display number of fields (words) in each record (line)

```
$awk -F: '{print $0,NF}' /etc/passwd
```

root:x:0:1:Super-user:/:/sbin/sh 7

…

BEGIN Pattern

- BEGIN Pattern is followed by an action block that is executed before awk process any line from the input file.

- BEGIN action is often used to change the value of the built-in variables, FS, RS, and so forth to assign initial values to user-defined variables and print headers of titles.

- Example

```
$awk 'BEGIN{FS=":"; RS="\n\n"} {print
$1,$2,$3}' myfile
```

END Pattern

- END patterns are handled after all lines of input have been processed.

- It does not match any input line

- Example:

To print the number of lines in a file

```
$awk 'END { print NR } ' testfile
```

Conditional expressions

condition expression1 ? expression2 :expression3

```
if (expression1)
expression2
else
expression3
```

Conditional expressions

```
if (expression1){
statement; statement;...
}
else if (expression2){
statement; statement;...
}
else {
statement
}
```

Relational Operators

Operator	Meaning
<	Less than
<=	Less than and equal
==	Equal to
!=	Not equal to
>=	Greater than and equal
>	Greater than
~	Match regular expression
!~	Not matched by regular expression

Loops

- while Loop

```
$awk -F: '{i=1; while (i<NF)
{print NF,$i;i++}}' /etc/passwd
```

- For Loop

```
$awk `{for ( i=1 ; i<NF; i++)
print NF,$i}' /etc/passwd
```

Examples (cont.)

- The variable max value is set according to compression between the first 2 fields:

```
$ awk '{if ($1>$2)
      max=$1;
else
      max=$2;
print max}' testing
```

- Arithmetical Operations

```
$ awk '{if ($1*$2>100) print $0}'
testing
```

Examples (cont.)

- To display line 4 and 5 only

```
$awk '{if (NR==4 || NR==5)
print NR":"$0
}' /etc/passwd
```

Thanks ☺

SBAHADER@GMAIL.COM