OS and Unix – introduction

- Define an OS (Operating System) and identify its functionalities.

- Definition
- Functions
  - Command Interpreter
  - Peripherals Manager
  - Memory Manager
  - Process Manager
- Types of Operating Systems
  - Single User
  - Multi User

Operating System (OS)

- A software program designed to act as an interface between a hardware and the user.

- It controls the hardware, manages system resources and supervises interaction between the system and its users.

Functions of OS

- ## Command Interpreter

  The Operating System interprets command typed in by the user and translates it to a machine language and vice versa, translates the results of the command.

- ## Peripherals Manager

  The Operating System manages the devices attached to the system. Inputs are the input devices like keyboard, mouse, card readers are taken and the result of the command is printed to output devices like printers, VGA etc. This I/O management is Peripherals Management.

- ## Memory Manager

  The processes running on a system require memory (RAM) to perform the tasks which they are intended to. The OS determines the allocation of memory for the processes based on the importance of the process.

- ## Process Manager

  The amount of time spent on a process by a CPU is managed by the OS.

Types of Operating Systems

- Single user

  As the name portrays only one user at any single point of time.

- Multi User

  Multiple users can perform operations on the OS at a singe point of time.

Unix – Evolution and Structure

- ***Stage I:*** Pre 1969
- ***Stage II:*** In 1969 CSRD of Bell Labs used GE's Mainframe 645 with an OS called MULTICS. This had the disadvantage of retaining the code of the previous OS therefore it was slow.
- ***Stage III:*** Ken Thompson was working on an experiment called "Space Travel" and wanted a faster OS, so wrote and OS called Unix in Assembly Language, but the disadvantage was that the OS could not be ported. Ken Thompson developed a language **B** to include portability..this was later renamed to **C** by Dennis Ritchie.
- ***State IV:*** In 1980 UNIX was completely rewritten in C and then evolved on the greatest OS in history UNIX (Uniplexed Information and Computing Service)

Structure of Unix

- Unix OS primarily consists of two parts: kernel and shell.
  - Kernel : Core of the Unix OS. It interacts with the hardware. It is loaded into memory when the system is booted. Its functions are
    - Managing the system resources
    - Allocating time and for users and processes
    - Managing process priorities and performing them.

  Assignment: Describe the Boot process of Linux? Break it down to 4 or 5 points.

  Difference between linux and windows?

Structure of Unix

- Shell: It interacts with kernel and the user.
  It has the following features:
  - Interactive Processing
  - Background Processing
  - Input / Output redirection
  - Pipes
  - Wild Card Patterns
  - Shell Scripts
  - Shell Variables
  - Programming language constructs

Types of Shell

- **Bourne Shell (sh)** -- This shell does not have the interactive facilities provided by modern shells such as the C shell and Korn shell.
- **C Shell (csh)** -- It provides a C-like language with which to write shell scripts - hence its name.
- **Korn Shell (ksh)** -- It provides all the features of the C and TC shells together with a shell programming language similar to that of the original Bourne shell.
- **Bash Shell(bash)** -- Bash provides all the interactive features of the C shell (csh) and the Korn shell (ksh). Its programming language is compatible with the Bourne shell (sh).

Usage of Simple Unix Commands

- date – Command to display and edit system date
- who – Command to find out who's logged into the system
- whoami – Command to display who you are logged in as
- w – Command to display who is doing what
- man – Command to display manual pages of Unix commands
- head – Command to display the first "n" lines in a file
- tail – Command to display the last "n" lines in a file

Usage of Directory Commands

- pwd
- ls
  - -l - Long listing of files
  - -a - List hidden files (files starting with ".")
  - -t - List files based on the ascending order of creation time
  - -d - List directory
  - -p - List files and directories but directories are followed by a "/"
  - -u - List by access time
- mkdir
- cd
- rmdir
- Relative paths and absolute paths

  Path defined from the current directory is Relative path.

  Ex: $ls -l ../newtest/cars

  Path defined from the root (/) is the absolute path.

  Ex: $ls -l /home/peter/newtest/cars

Usage of File Commands

- cat
- cp

  -f – Copy files by force, overwrite in case one exists with the same name.

  -i – Copy files interactively

  -p – Preserve mode, ownerships, timestamps while copying

  -R – Copy files recursively into a directory
- ln
- mv
- rm

  -i – Remove files interactively

  -r – Remove files recursively

Assignment: Difference between softlink and hardlink.

Directory Hierarchy

- Inverted Tree with root on the top and the other files systems below the root.
  - / the root of file system name space
  - /**bin** – symbolic link to /usr/bin is the location of binary files of standard system commands
  - /**dev** – Consists of the logical device files names. They are symbolic links to /devices
  - /**etc** – Consists of system configuration files
  - /**export** – Directory used for sharing file systems
  - /**home** – Home directory of the users
  - /**opt** – Directory for the install of software
  - /**sbin** – Single user bin directory, contains commands used during the booting process and manual system recovery
  - /**tmp** – The directory for temporary files. It is cleared during the boot process
  - /**usr** – Contains scripts, binaries used by all users
  - /**var** – Directory where varying files are such as logs, mail and printer spools are stored
  - **Assignment: Research on Directory Hierarchy in Linux.**

Input/Output Redirection

- Standard Input and Standard Output Files
- Redirection
  - Input redirection
  - Output redirection
  - Redirecting both Standard Input and Standard Output

Wild Card Patterns

- *
- ?
- []

- * # ls n*
- ? # rm abc.??? (abc.txt, replaces each char after .)
- [] # file [a-f]* (lists file starts from a to f

Environmental Variables

- PS1 (Prompt String 1)
- PATH
- TERM
- HOME
- LOGNAME
- MAIL
- PS2
- PATH=$PATH:/home/oracle, export PATH, echo $PATH)
- . bash_profile (detail on next slide)
- set or env

- When bash is invoked as an interactive login shell, or as a non-interactive shell with the --login option, it first reads and executes commands from the file /etc/profile, if that file exists. After reading that file, it looks for ~/.bash_profile and ~/.bash_login, in that order, and reads and executes commands from the first one that exists and is readable.

More file commands

- read
- echo
- date : mkdir Backup_`date +%m-%d-%Y`
  - +%D mm/dd/yy
  - +%H 0-23 hrs
  - +%M 00-59 mins
  - +%S 00-59 secs
  - +%T HH:MM:SS
  - +%w Day the week
  - +%a Abbr. Weekday
  - +%h Abbr. Month
  - +%r Time in AM/PM
  - +%y Last two digits of year
- wc
- find
  - -name
  - -type
  - -mtime
  - -exec
  - -ok -- **find . -name "*.txt" -ok mv {} junkdir \; (interactive move)**
    touch ehis{1..10}.txt, find . -type f -name "ehis*" -exec mv {} mtimeDir \;
  - find . -type f -name "ehis*" -exec ls -ltr {} \;

  - **Assignment**: Create a Directory with name data_datestamp (datestamp should pick current system date/time (for eg: data_08-11-2011).
  - **Push labs 1, 2 and 3.**

File Access Permissions

- Result of ls -l
  - drwxr-xr-x 2 peter staff 512 Mar 20 01:09 chapter1
- Read to display, copy or compile a file
- Write to edit it or delete it
- Execute permission to execute a file
- r=4 ; w=2; x=1
- chmod

Introduction to Pipes and Filters

- Why Pipes and Filters
- Pipe
  - A pipe is a mechanism which takes the output of a command as an input in the next command.
  - Ex: $ who | wc –l

    Here the output of “who” is taken as the input for “wc –l” and the output is displayed
- Filter
  - Filters take input from a standard input or a file, processes it and send it to standard output or a file.
  - Filters are used to extract lines which contain a specific patterns, arrange contents of a file, replace existing characters, merge files etc.

Usage of sort commands

- "sort" arranges the input taken from standard input.
  - -r -sort in reverse alphabetical order
  - -f -force the sort
  - -n -to sort based on numerical value
  - +pos1 -pos2 option – to sort based on position, legacy, you can use -k <fieldname>.
  - -t -to sort files that are delimited other than space or tab space
  - -u -removes duplicate lines from the input
  - -o -sends the output to a file
  - -b -to negate the effect of input characters

sort

- **sort -n**

$ sort -n
9
17
2
Ctrl + D
2
9
17

sort

- sort +pos1 -pos2

$ cat names
george mathew thomas
gideon mark antony
victoria thomas becker
slyvia mary peter
edwin frank winchester

[root@HomeMachine opt]# sort -rn -t ":" +2 -3 sort.txt | grep -v name
dinesh:Denton:15:76909
promod:Irving:12:75056
bhesh:Eules:11:74630
lila:Dallas:10:75094

sort

- **sort -t**

$ cat names_t
george:mathew:thomas
victoria:thomas:becker
slyvia:mary:peter
edwin:frank:winchester
$ sort -r +2 -3 names_t
edwin:frank:winchester
george:mathew:thomas
slyvia:mary:peter
victoria:thomas:becker
$ sort -t ":" +2 -3 names_t
victoria:thomas:becker
slyvia:mary:peter
george:mathew:thomas
edwin:frank:winchester
$

sort

- **sort –u (–u – unique)**

$ more names.txt
george mathew thomas
gideon mark antony
victoria thomas becker
slyvia mary peter
edwin frank winchester
george mathew thomas
$ sort –u names.txt
edwin frank winchester
george mathew thomas
gideon mark antony
slyvia mary peter
victoria thomas becker
$

sort

- sort -o (-o writes output to a file)

```
$ more names.txt
george mathew thomas
gideon mark antony
victoria thomas becker
slyvia mary peter
edwin frank winchester
george mathew thomas
$ sort  -r names.txt -o names2.txt
$ more names2.txt
victoria thomas becker
slyvia mary peter
gideon mark antony
george mathew thomas
george mathew thomas
edwin frank winchester
```

sort

- sort –b (Ignores leading blanks) – this is obsolete and show students how you can use sed to ignore leading blanks)

:%s/\s\{1,}/ (any leading blanks)
grep –v “^$” test.txt > test10.txt (blank lines)

Usage of grep commands

- grep
  - Used to search a pattern string from a file or a Standard Input and display those lines on a Standard Output.
  - Grep stands for "**g**lobal search for **re**gular ex**p**ression"
  - -v -to display lines which do not match a specified pattern
  - -c -display only the count of lines which match the pattern
  - -n -displays the line which matched the pattern with the line number
  - -i -ignores case distinction when doing a search
  - [] -Use of the wild character for search
  - $ -To extract those lines that end with a character
  - ^ - To extract those lines that begin with a character

grep

- grep -v

```
$ ps -ef | grep ssh
root      670     1  0 Mar05 ?       00:00:13 /usr/sbin/sshd
root     9268   670  0 23:12 ?        00:00:00 /usr/sbin/sshd
vikasy   9270  9268  0 23:13 ?        00:00:00 [sshd]
root     9323   670  0 23:29 ?        00:00:00 /usr/sbin/sshd
vikasy   9325  9323  0 23:29 ?        00:00:00 [sshd]
root     9363   670  0 23:35 ?        00:00:00 /usr/sbin/sshd
peter    9365  9363  0 23:35 ?        00:00:00 [sshd]
peter    9410  9366  0 23:44 pts/2    00:00:00 grep ssh
$ ps -ef | grep ssh | grep -v grep
root      670     1  0 Mar05 ?       00:00:13 /usr/sbin/sshd
root     9268   670  0 23:12 ?        00:00:00 /usr/sbin/sshd
vikasy   9270  9268  0 23:13 ?        00:00:00 [sshd]
root     9323   670  0 23:29 ?        00:00:00 /usr/sbin/sshd
vikasy   9325  9323  0 23:29 ?        00:00:00 [sshd]
root     9363   670  0 23:35 ?        00:00:00 /usr/sbin/sshd
peter    9365  9363  0 23:35 ?        00:00:00 [sshd]
```

grep

- **grep -c**

$ more names.txt
george mathew thomas
gideon mark antony
victoria thomas becker
slyvia mary peter
edwin frank winchester
george mathew thomas
$ grep -c george names.txt
2
$

grep

- **grep -n**

$ more names.txt
george mathew thomas
gideon mark antony
victoria thomas becker
slyvia mary peter
edwin frank winchester
george mathew thomas
$ grep -n george names.txt
1:george mathew thomas
6:george mathew thomas
$

grep

- **grep -i**

$ more names
george mathew thomas
victoria thomas becker
slyvia mary peter
edwin frank winchester
george mathew thomas
George mathew thomas
$ grep -i george names
george mathew thomas
george mathew thomas
George mathew thomas

grep

- **grep with [] wild characters**
- **$ more names.txt**

george mathew thomas
victoria thomas becker
slyvia mary peter
edwin frank winchester
george mathew thomas
George mathew thomas
gaorge nick carter

$ grep g[a,e]orge names.txt

george mathew thomas
george mathew thomas
gaorge nick carter

$

grep

- **grep with “$”**

$ more names.txt
george mathew thomas
victoria thomas becker
slyvia mary peter
edwin frank winchester
george mathew thomas
George mathew thomas
gaorge nick carter
$ grep "s$" names.txt
george mathew thomas
george mathew thomas
George mathew thomas

grep

- grep with “^”

```
$ more names.txt
george mathew thomas
victoria thomas becker
slyvia mary peter
edwin frank winchester
george mathew thomas
George mathew thomas
gaorge nick carter
$ grep -i "^g" names.txt
george mathew thomas
george mathew thomas
George mathew thomas
gaorge nick carter
$
```

Usage of other filters

- egrep
  - To extract more than one pattern
    - | –multiple patterns
    - –f – use a file to match
    - ()$ – use a pattern to match at the end
- fgrep
  - To extract a fixed string
- uniq
  - Compares adjacent lines and removes them and sends the output to a standard output or a file
    - –d –list duplicate lines
    - –u –list unique lines
    - –c –count of duplicate lines in front
- pg
  - To paginate a view if the view is more than one page
- more and less (example: more /etc/passwd or less /etc/passwd
  - To paginate a view if the view is more than one page

egrep

- **egrep**

$more countries .txt
atlanta georgia usa
delhi delhi india
dallas texas usa
$ egrep "atlanta|texas" countries.txt
atlanta georgia usa
dallas texas usa
$ more countsearch.txt
atlanta
delhi
$ egrep -f countsearch.txt countries.txt
atlanta georgia usa
delhi delhi india

egrep

- **egrep ()$**

$more countries

atlanta georgia usa
delhi delhi india
dallas texas usa

$ egrep "a$" countries

atlanta georgia usa
delhi delhi india
dallas texas usa

fgrep

- fgrep

Selects the escape characters.

[root@HomeMachine ~]# vi grep-gfrep.txt

[root@HomeMachine ~]# grep "." grep-gfrep.txt

.test

S3S

config

[root@HomeMachine ~]# fgrep "." grep-gfrep.txt

.test

$more names_fgrep

george mathew thomas

victoria thomas becker

slyvia mary peter

edwin frank winchester

$ fgrep "george mathew" names_fgrep

george mathew thomas

uniq

- **uniq -d** (-d displays only duplicate lines)

$ more names_uniq.txt
george mathew thomas
george mathew thomas
victoria thomas becker
slyvia mary peter
edwin frank winchester
george mathew thomas
George mathew thomas
gaorge nick carter
$ uniq -d names_uniq.txt
george mathew thomas

uniq

- **uniq -u** (Prints only unique lines)

```
$ more names_uniq.txt
george mathew thomas
george mathew thomas
victoria thomas becker
slyvia mary peter
edwin frank winchester
george mathew thomas
George mathew thomas
gaorge nick carter
$ uniq -u names_uniq.txt
victoria thomas becker
slyvia mary peter
edwin frank winchester
george mathew thomas
George mathew thomas
gaorge nick carter
$
```

uniq

- **uniq -c** (prefix lines by the number of occurrences, appearing right after eachother)

```
$ more names_uniq
george mathew thomas
george mathew thomas
victoria thomas becker
slyvia mary peter
edwin frank winchester
george mathew thomas
George mathew thomas
gaorge nick carter
$ uniq -c names_uniq
      2 george mathew thomas
      1 victoria thomas becker
      1 slyvia mary peter
      1 edwin frank winchester
      1 george mathew thomas
      1 George mathew thomas
      1 gaorge nick carter
$
```

Usage of other filters

- cut
  - One particular field or character can be extracted from a file using the cut command
    - -d -mention the delimiter in file
    - -c -cut characters
    - -f -cut fields
- paste
  - For pasting files
- tee
  - To temporarily storing data
- tr
  - To convert files from lower case to upper case and vice versa and to truncate spaces
  - -s to truncate spaces in temp input

cut

- cut -d, cut -f and cut -c
- #cat /etc/passwd | cut -d ":" -f1 | cut -c1-4

```
$ more names_t.txt
george:mathew:thomas
victoria:thomas:becker
slyvia:mary:peter
edwin:frank:winchester
$ cut -d ":" -f2 names_t.txt
mathew
thomas
mary
frank
$ cat names_t.txt | cut -f1 | cut -c1-4
geor
gide
vict
slyv
edwi
$
```

paste

- **paste**

$ more fname.txt

mark
ben
bill
david

$ more lname.txt

antony
aflek
joy
korn

$ paste fname.txt lname.txt

mark antony
ben aflek
bill joy
david korn

tee

- tee
- [root@NFS-Server ~]# cat cut1.txt
- george:mathew:thomas
- victoria:thomas:becker
- slyvia:mary:peter
- edwin:frank:winchester

$ cut -d ":" -f2 cut1.txt | tee file2.txt
$ cut -d ":" -f2 cut1.txt | tee file2.txt | cut -c1-3 > out.txt
$ cat out.txt

tr

- **tr**

$ ls –l

```
total 44
-rw-rw-r--    1 peter   peter        57 Apr 19 00:52 countries
-rw-rw-r--    1 peter   peter        14 Apr 19 00:56 countserch
-rw-rw-r--    1 peter   peter        25 Apr 19 01:38 fname
```

$ ls –l | cut –d “ " –f3

```
peter
peter
peter
```

$ ls –l | tr –s “ " | cut –d “ " –f9

```
Countries
countserch
fname
```

$

tr

- tr

```
$ cat lname.txt
antony
aflek
joy
korn
thomson
$ cat lname.txt | tr "[a-z]" "[A-Z]"
ANTONY
AFLEK
JOY
KORN
THOMSON
$
```

Usage of Pipes and Filters together

- $ ls –l | tr –s “” | cut –d “ “ –f3 | more (or pg)
- $ls –l | grep “^d”
- $ cat text| sort | head| cut –d “ ” –f1

Introduction to the vi editor

- The most powerful editor in the Unix world
- “vi” stands for visual editor
- vi is a full screen editor
- vi functions in three modes, insert mode, command mode and ex escape mode

  Usage: $ vi filename

Getting started with vi editor

- Movement of the cursor
  - h or backspace to move left
  - l or space to move right
  - k or – to move up
  - j or + to move down
- :split or :vsplit → split vertically to compare contents (to run script within the vi editor, :!bash <scriptfile>

Getting started with vi editor

- More Cursor Movements
  - w – to move by a word
  - W – to move by a word ignoring punctuations
  - e – to move to the end of a word
  - E – to move to the end of the word ignoring punctuations
  - b – to move backwards by a word
  - B – to move backwards by a word ignoring punctuations
  - ^ – to move to the beginning of the line
  - $ – to move to the end of the line
  - L – to move to the last line of the file
  - H – Beginning of the file
  - G – GOTO, to go the bottom of the file
  - 5w, 5e, 5B etc

Screen commands

- Ctrl F – move forward by a screen
- Ctrl B – move backward by a screen
- Ctrl D – move forward by half screen
- Ctrl U – move backward by half screen
- Ctrl L – clear messages that appear on the ex command line
- Ctrl G – displays status on the status line

Usage fo Some Editing Commands

- Text insertion in vi
  - i/I – invokes insertion mode.
  - O – allows insertion by creating a blank line above the current line
  - o – allows insertion by creating a blank line below the current line
  - a – used to append the text, text is appended after the current cursor position
  - A – used to append text, text is appended at the end of the line

Usage of Some Editing Commands

- Deleting data is vi
  - dd – deletes the line in which the cursor is positioned
  - dw – deletes the word from the cursor position to the end of the word
  - dW – deletes the word from the cursor position to the end of the word ignoring any punctuations that appear with the word
  - x – deletes the character at the current position
  - X – deletes the character before the current cursor position
  - D – deletes the line from the current position to the end of the line
  - yy – Yank lines , nyy can be user to copy n lines
    "p" to paste the lines
  - "np – Where "n" is the number of the delete sequence
    For Example: To undelete the 2nd from the last delete use "2p.

Usage of Some Editing Commands

- The undo command
  - “u”
- Joining lines of a file
  - “J”
- Replacing text
  - R – replace multiple character
  - r – replace single character
- Line numbering in vi
  - :set number
  - :set nonumber
  - :set showmode
  - :set ignorecase or set ic
  - :set all

Usage of Some Editing Commands

- Copy Delete and Move Multiple Lines
  - : 2co5 – copy 2nd line below 5th line
  - : 1,3co7 – copy 1-3 lines to below 7th line
  - : 2mo9 – move 2nd line to the 9th line
  - : 2,5mo$ – move 2-5 to the end of the file
  - : 5,7d – delete 5-7 lines
  - : . +4 d – delete 4th line from the current cursor position
  - : . -5 d – delete 5th line above the current line
  - : .,$ d – delete from the current position to end of the file
  - : sh – to exit out of the vi buffer and come to the command prompt, to come back again to the buffer type “exit” at the command prompt

Usage of Some Editing Commands

- Searching and replacing text
  - /<text> – search for text
  - ?<text> – search for text in reverse order
  - n – repeat the search
  - N – repeat search in the reverse order
  - :s/<oldstr>/<newstr>/ – replace first occurrence in the line where cursor is present
  - :s/<oldstr>/<newstr>/g – replace all the occurrence in a line
  - :50,100 s/<oldstr>/<newstr>/g – replace all the occurances of the string from 50-100 lines
  - :1,$ s/<oldstr>/<newstr>/g – replace all the occurrences of the string in the file
  - :% s/<oldstr>/<newstr>/g – replace all the occurrences of the string in the file

Saving and quitting vi

- :q! – quit without writing
- :wq – write and quit
- :x – replaces the old copy with a new copy and then quits
- ZZ – same as wq

date

- $ date

Wed Mar 24 20:00:17 CST 2004

who

- $ who

```
root     pts/0        Mar 24 20:03 (138.83.40.14)
root     pts/2        Mar 23 15:58 (113.128.156.190)
```

who am i

- $ who am i

root pts/0 Mar 24 20:03 (138.83.40.14)

PS1

- `hostname`{`whoami`}'$PWD:>'

w

- $w

```
20:30:37  up 11 days, 15:05,  2 users,  load average: 0.08, 0.18, 0.17
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     pts/0    192.168.1.20      8:03pm  0.00s  0.24s  0.06s  -bash
root     pts/2    192.168.1.30  Tue 3pm 28:21m  0.52s  0.52s  -bash
```

man

- $man date

DATE(1) FSF DATE(1)

NAME
date - print or set the system date and time

SYNOPSIS
date [OPTION]... [+FORMAT]
date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]

DESCRIPTION
Display the current time in the given FORMAT, or set the system date.

-d, --date=STRING
display time described by STRING, not â€˜nowâ€™

-f, --file=DATEFILE
like --date once for each line of DATEFILE

head

- cat cars

Ferrari
Porsche
Merc
BMW
Honda
Toyota
Nissan
GMC
Cadillac
Ford

- $ head -3 cars

Ferrari
Porsche
Merc

tail

- $ cat cars

Ferrari
Porsche
Merc
BMW
Honda
Toyota
Nissan

- $ tail -3 cars

Honda
Toyota
Nissan

pwd

- $ pwd

/home/peter

ls

- $ ls -l (ls -ltr is imp as it displays the last modified time)

```
drwxrwxr-x   2 peter   peter      4096 Nov 22 01:53 tmp
-rw-rw-r--   1 peter   peter        87 Nov  2 06:48 uniqtest
-rw-rw-r--   1 peter   peter        27 Mar 21 06:11 wctest
lrwxrwxrwx   1 peter   peter         6 Mar 25 22:56 wctest1 -> wctest
```

- $ ls -al

```
drwx------   8 peter   peter      4096 Mar 25 22:56 .
drwxr-xr-x  19 root    root       4096 Mar 21 00:16 ..
-rw-rw-r--   1 peter   peter        25 Nov  2 07:05 bag
-rw-------   1 peter   peter     10502 Mar 24 12:48 .bash_history
-rw-r--r--   1 peter   peter        24 Feb 11  2003 .bash_logout
-rw-r--r--   1 peter   peter       191 Feb 11  2003 .bash_profile
-rw-r--r--   1 peter   peter       124 Feb 11  2003 .bashrc
drwxrwxr-x   2 peter   peter      4096 Nov 22 01:53 tmp
-rw-rw-r--   1 peter   peter        87 Nov  2 06:48 uniqtest
-rw-rw-r--   1 peter   peter        27 Mar 21 06:11 wctest
lrwxrwxrwx   1 peter   peter         6 Mar 25 22:56 wctest1 -> wctest
```

ls : list, ls -ltr or ls -lathr

- $ ls -tl

```
-rw-r--r--   1 peter   peter     11533 Feb 19 20:21 rpmdetails
-rw-rw-r--   1 peter   peter         0 Mar 10 17:05 test
-rw-rw-r--   1 peter   peter        27 Mar 21 06:11 wctest
-rw-rw-r--   1 peter   peter        63 Mar 25 22:14 cars
lrwxrwxrwx   1 peter   peter         6 Mar 25 22:56 wctest1 -> wctest
```

- $ ls -dl

```
drwx------   8 peter   peter      4096 Mar 25 22:56 .
```

- $ ls -p

```
bag     dfoutput  httpd.conf     lname     nation    newtr1              RPMS/     vitest
bin/    edtest    httpd.conf.org Maildir/  nation1   newuniq              sedtest   vitest2
```

- $ ls -ul bag

```
-rw-rw-r--   1 peter   peter        25 Feb 20 00:54 bag
```

- $ ls -al bag

```
-rw-rw-r--   1 peter   peter        25 Nov  2 07:05 bag
```

mkdir

- $ mkdir newtest

cd

- $ pwd

/home/peter

- $ cd newtest
- $ pwd

/home/peter/newtest

rmdir

- $ ls -l

```
drwxrwxr-x   2 peter   peter      4096 Mar 25 23:35 newtest
-rw-rw-r--   1 peter   peter       169 Nov  2 10:36 newtr
-rw-rw-r--   1 peter   peter       169 Nov  2 10:37 newtr1
```

- $ rmdir newtest
- $ ls -l

```
-rw-rw-r--   1 peter   peter       169 Nov  2 10:36 newtr
-rw-rw-r--   1 peter   peter       169 Nov  2 10:37 newtr1
```

cat

- $ cat cars

Ferrari
Porsche
Merc
BMW
Honda
Toyota
Nissan
GMC
Cadillac
Ford
$

cp

- $cp –f file1 file2
- $ ls –l name*

```
-rw-rw-r--   1 peter   peter       100 Nov  2 07:42 names
-rw-rw-r--   1 peter   peter       103 Nov  2 05:20 names1
-rw-rw-r--   1 peter   peter       100 Mar 26 01:36 names2
```

- $ cp –i names names1

```
cp: overwrite `names1'? y
$
```

- $ cp –p names names1
- $ cp –R newtest/* newtest2/*

ln

- $ tail -3 cars

GMC
Cadillac
Ford

$ cd newtest/

$ ln -s ../cars ./newcars

$ ls -l

total 0
lrwxrwxrwx 1 peter peter 7 Mar 26 01:55 newcars -> ../cars

$ tail -3 newcars

GMC
Cadillac
Ford

$

mv

- $ ls -l vi*

```
-rw-rw-r--   1 peter   peter      1523 Nov  8 10:31 vitest
-rw-rw-r--   1 peter   peter         0 Nov  7 05:02 vitest2
```

```
$ mv vitest vitest3
$ ls -l vi*
-rw-rw-r--   1 peter   peter         0 Nov  7 05:02 vitest2
-rw-rw-r--   1 peter   peter      1523 Nov  8 10:31 vitest3
$
```

rm

- $ ls -l vi*

```
-rw-rw-r--    1 peter   peter         0 Nov  7 05:02 vitest2
-rw-rw-r--    1 peter   peter      1523 Nov  8 10:31 vitest3
```

$ rm -i vitest2

```
rm: remove regular empty file `vitest2'? y
```

$ ls -l vi*

```
-rw-rw-r--    1 peter   peter      1523 Nov  8 10:31 vitest3
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

$ ls -l | grep newtest

```
drwxrwxr-x    2 peter   peter      4096 Mar 26 01:55 newtest
```

$ rm newtest

```
rm: cannot remove `newtest': Is a directory
```

$ rm -ir newtest

```
rm: descend into directory `newtest'? y
rm: remove symbolic link `newtest/newcars'? y
rm: remove directory `newtest'? Y
```

$ ls -l | grep newtest

$

Introduction to Shell Scripts

- Introduction
- Creation and Execution
- Shell Variables
  - User defined variables
  - Environment variables
  - Local and Global Variables
  - Special Shell Variables
- Features offered by Shell
- Programming Language Constructs

Introduction to Shell Scripts

- Introduction
  - A shell script is a program that issues or executes a sequence of Unix commands.

Creation and Execution of Shell Scripts

- The sequence of commands can be entered into a file using the vi editor.
- Creation of Shell Scripts
  - $ vi greet.sh
  - #!/bin/bash
    echo “Please enter your name”
    read name
    echo “Hi $name, welcome to the Unix Session”

    N.B.: Please try not to use the names of Shell Scripts with any words that UNIX recognizes as commands.
- Execution of Shell Scripts
  - Shell scripts are executed in 3 ways
  - -bash scriptFileName
    - $ sh ./filename
      or
    - $ chmod u+x filename
      $ ./filename

```
#!/bin/bash
echo "enter the first no."
read a
echo "enter the sec no."
read b
c=`expr $a + $b`
d=`expr $a \* $b`
echo "the sum of the 2 nos. is: $c"
echo "the multiplication of the 2 nos. is: $d“
##checking if c equals d (# means comment, line won’t be read by system)
if [ $c -eq $d ]
   then
echo "Pass"
   else
echo "Fail"
fi
For decimal/floats: echo “$a+$b” | bc
---
source cal.sh
calculator
```

Shell Variables

- User defined variables
  - pet=rabbit
    echo $pet
- Environmental variables
  - PS1, PATH, LOGNAME, MAILDIR etc
- Local Variables
  - Variable local to a shell
- Global Variables
  - Variables available for child shells created by shell scripts
  - Local variables become global variables by exporting the variable with the “export” command.

Features offered by Shell

- The # Symbol
  - This is the comment symbol, on execution this line will be ignored.
- The Escape mechanism
  - Special characters like * if needed to be used in the shell scripts then escape mechanism "\" should be used.

    Ex: echo "The symbol of multiplications is *"
- Command substitution
  - When a command is enclosed within back quotes, the command is replaced by the output it produces, this is called command substitution.
  - $ echo `date`

    Sat Apr 3 01:56:20 CST 2004

Features offered by Shell

- Positional Parameters
  - Unix accepts parameters at the command line so parameters can be passed to the shell scripts from the command line itself.
  - The arguments are named as $1, $2, $3 etc, since they represent their position they are called positional parameters.
  - $0 represents the command itself.
- The shift command
  - The maximum number of positional parameters is $9, if we have more than nine number of positional parameters, the “shift” command is used for moving the positional parameters to the next 9.
- The exit command
  - The exit command is used to exit the shell script and tell us the exit status of the shell. Exit status “0” is success and exit status “1” is failure.

```
------------------------
#!/bin/bash
if [ "$1" != "" ]; then
   echo "Positional parameter 1 contains something"
else
   echo "Positional parameter 1 is empty"
Fi
--------------------------
```

The shell maintains a variable called $# that contains the number of items on the command line in addition to the name of the command ($0).

```
#!/bin/bash
if [ $# -gt 0 ]; then
   echo "Your command line contains $# arguments"
else
   echo "Your command line contains no arguments"
Fi
------
```

Features offered by Shell

- Numerical Comparisons
  - To compare the two numerical values n1 and n2
    - $n1 -eq $n2 Will check if the two integers are equal
    - $n1 -ne $n2 Will check if the two integers are not equal
    - $n1 -gt $n2 Will check of n1 is greater than n2
    - $n1 -lt $n2 Will check of n1 is less than n2
    - $n1 -ge $n2 Will check if n1 greater than equal to n2
    - $n1 -le $n2 Will check if n1 less than or equal to n2
- Logical Operator
  - ! Negates the following expression
  - -a used for indicating “and”
  - -o used for indicating “or”

Features offered by Shell

- Arithmetic Operators and Expressions
  - +, -, * and /
  - expr – This command helps in getting the numerical value of the digital strings
    - $ expr 6 + 4

      10

    N.B.: 1. There should be spaces between the expr, the number and the arithmetic operators

    2. Division is integer division only. The result of the division is truncated to the largest integer.
- Conditional Execution Operators
  - Command execution after successful or failure of another command
  - &&
    - $ grep good gems && echo “pattern found”
  - ||
    - $ grep good gems || echo “pattern not found”

Programming Language Constructs

- if…then….else…fi
- for….do….done
- while…do…done
- until…do…done
- case…esac

The if construct

- This construct is useful to do a set of commands based on the condition being true, we can specify another set of commands if the condition is not true.
- The general form for the representation of the command is

```
if (condition)
then
        commands
else
        commands
fi
```

Ex:

```
# Shell script by SP, 04/03/2004
if (grep read text > /dev/null)
then
    echo "Pattern Found"
else
    echo "Pattern not Found"
fi
```

The if construct

- #!/bin/bash

```
# Shell script by SP, 04/03/2004
if (grep read text.txt > /dev/null)
then
        echo "Pattern Found in Text"
elif (grep read  passage > /dev/null)
then
        echo "Pattern Found in passage"
else
        echo "Pattern not Found in either"
fi
```

The case…esac Construct

- This construct helps in the execution of shell scripts based on our choice.
- The general form of the case statement is

```
case value in
choice1)        commands;;
choice2)        commands;;
.....
.....
esac
```

Here choice1 and choice2 are possible courses of action. If the value is taken as choice1, then the commands in choice1 are executed and similarly for choice2. The right parenthesis is used to identify the label names.

The case…esac Construct

- #!/bin/bash

```
# This is a shell script created by SP, 04/03/2004 for the
   case...esac programming construct
echo "Menu"
echo "1. Your current directory"
echo "2. Today's Date"
echo "3. List of users logged in"
echo "Please enter your choice"
read choice
case $choice in
1)      pwd;;
2)      date;;
3)      who;;
*)      echo "Invalid Choice"
esac
```

The for construct

- This construct is used to perform same set of operations on a list of values.
- The general form of representation of the **for construct** is

for <variable> in value1 value2 value3…
do
command(s)
done

- The variable can be anything. The sequence of commands between do and done are executed for the list of values. When there are no more values to be taken by the variables the loop ends

The for construct

```
#!/bin/bash
for k in 1 2 3 4 5
do
echo "The number is $k"
echo "The square of the number is `expr $k \* $k`"
Done
---------
#!/bin/bash
for i in `seq 1 10`  (or {1..10} can be used)
do
echo $i done
```

The while construct

- The commands within the while loop are executed repeatedly as long as the conditions remains true
- The general form of the while loop is

```
while control command
do
   command list
done
```

Each time when the Shell attempts to execute the control command, if the exit status of the control command is a “0”, then the commands between do and done are executed.

The while construct

```
#!/usr/bin/bash
x=1
while [ $x -lt 5 ]
do
echo "Welcome $x times"
x=$(( $x + 1 ))
done
```

```
#!/bin/bash
COUNTER=0
while [ $COUNTER -lt 10 ]
do
echo The counter is $COUNTER
let COUNTER=COUNTER+1
done
```

NB: The commands inside the loop at some point should become false, otherwise the loop will be an infinite loop.

The until construct

- This loop is similar to the while loop except that it continues as long as the control command fails
- The general form of the until loop is

```
until control command
do
    command (s)
done
```

- If the control command fails, the commands between the do and the done executed. The loop ends once the control command succeeds.

The until construct

- # This is a script SP, 04/03/2004 to illustrate the until loop

```
until ls | grep red > /dev/null
do
echo "File not found"
exit
done
echo "File red is found“
```

Here as long as the execution of the command *ls | grep red > /dev/null* fails, the file not found is displayed. Once the command succeeds, the until loop ends and other echo command execution takes over.

The test command

- To test equalities of two given strings, to test if the if the argument of a shell script is a file or a dir
- The [ ] can be used instead of test
- #!/usr/bin/ksh

```
# This is a script SP, 04/03/2004 to illustrate the test
echo "What is the greatest operating system ?"
read ans
if test $ans = UNIX -o $ans = Unix -o $ans = unix
                    or
if [ $ans = UNIX -o $ans = Unix -o $ans = unix ]
then
        echo "You are Unix Literate"
else
        echo "Please try again"
fi
```

Shell Programming – Examples

- **Program 1**

#!/bin/bash (you can verify the shell using echo $SHELL).

This is a script SP, 04/03/2004 to calculate the HRA of the employees depending on basic

```
echo "Enter employee's basic“
read basic
if [ $basic -gt  -o -eq 5000 ]
then
        HRA=`expr $basic / 5`
elif [ $basic -ge 4000 -a $basic -le 5000 ]
then
        HRA=`expr $basic / 4`
        echo "The HRA of the employee is $HRA"
else
     HRA=`expr $basic /10`
        echo "The HRA of the employee is $HRA"
fi
```

Shell Programming – I examples

- **Program 2**

```
#!/usr/bin/ksh
# This is a script SP, 04/03/2004 to display the menu and perform the
    following appropriate action.

echo "Menu"
echo "1. Displays  the IP address of the server"
echo "2. Displays a long listing of files including hidden files"
echo "3. Displays a current working directory"
echo " Please Enter your choice"
read choice
        case $choice in
        1) ifconfig -a;;
        2) ls -al;;
        3) pwd;;
        *) echo "You have entered an invalid choice, please enter numbers
    between 1-3";;
        esac
```

Shell Programming – I examples

- **Program 3**

```
#!/bin/bash
#This is a shell script by peter, 04/03/2004 to guess a number between 1 and 50"
ans=38
count=0
echo "I'm thinking of a number between 1 and 50"
echo -e "Please guess the number:\c"
read guess
until [ $ans -eq $guess ]
do
        if [ $guess -gt $ans ]
        then
                echo "The number is too high, please try again"
        else
                echo "Too low, please try again"
        fi
        count=`expr $count + 1`
read guess
done
echo "You have found  out the number using $count guesses"
```

Shell Programming – II

- Advanced Features of Shell
- Additional programming language constructs
- Background Processing

Advanced features of Shell

- Command Grouping
  - We can get the output of two commands written on the command line using the ";"
  - More on test command
    - -z to check if the string is of zero length
    - -n to check if the string is of non-zero length

Advanced features of Shell

```
#!/bin/bash
echo "Enter your name"
read name
if [ -z $name ]
then
echo "Name not entered"
exit
else
echo "hi $name!! Have a good day"
fi
```

Test on File Types

- The test command is also used to check for the status of the files.
  - -f to check the existence of the file and to check if is an ordinary file
  - -d to check for the existence of the file and to check if it is a directory
  - -r to check if the file exists and it is readable
  - -w to check if the file exists and it is writable
  - -x to check if the file exists and it is executable
  - -s to check if the file exists and if it not empty
  - Show how Command Line Argument works.

Test on File Types

```
# The user should enter the file name at the command line.
   #!/bin/bash
if [ $# -lt 1 ]
 then
echo "Invalid Usage, usage is $0”
exit
fi
if [ -x $1 ]
then
        echo "$1 has executable permission"
else
        echo "$1 does not have execute permission“
fi
----
```

To run: bash scriptname.sh file1.txt

The set command

- The set command is to find out the existing values of our environmental variables

Shell Functions

- The Shell function is a group of commands that is referred by a single name from the command line. This is similar to shell scripts but they are can be executed directly by the login shell unlike shell script which is executed in a subshell.
- $ funtion home {

  ls –l | grep “^d”

  }

Create all the functions in one file and call them over to the next.

- #!/bin/bash
- source scriptFileName
- functionName

**Use one file just to define your functions, do not call those on the original file, start calling on the sec/third scripts.

Additional Programming Language Constructs

- The while true Loop

The loop will continue to run until and an interrupt character is pressed or exit statement is encountered.

```
ans = " "
while true
do
        echo "Do you wish to enter value(y/Y)?"
        read ans
        if  [ $ans = Y  -o $ans = y ]
        then
                echo "Hi $LOGNAME how are you?"
        fi
                echo "Press Ctrl C key to exit"
done
```

Additional Programming Language Constructs

- The until false loop

  This is a complementary of “while true” loop.

  As long as the condition remains false, the execution of the script continues

```
until false
do
    ps –f
    sleep 5
    echo “Unix at your service”
    echo “Press Ctrl+C  key to exit”
done
```

The break and continue statements

- **The break statement**

This helps in the termination of the loop.

```
Ex: #/bin/bash
#This is a script to illustrate the break statement
while true
do
        echo "Enter your choice:"
        echo "Enter w to quit"
        echo "Menu"
        echo "a. Today's date"
        echo "b. List of users"
        echo "c. Name of the home directory"
        read choice
        case $choice in
                a) clear;date;;
                b) clear;who;;
                c) clear;echo "Your home directory is $HOME";;
                w) break;;
                *) echo "Not a valid choice"
        esac
done
```

The break and continue statements

- **The continue statement**

  This statement is used if we want to skip the remaining commands in the loop and start from the beginning of the loop again.

```
#!/bin/bash
# This is a script to illustrate the usage of continue statement
ans=" "
echo "Do you want to enter a value (Y/y)?"
read ans
while [ $ans = "Y" -o $ans = "y" ]
do
        echo "Enter a name"
        read name
        echo $name >> names
        wish=" "
        echo "Do you wish to continue?"
        read wish
        if [ $wish = Y -o $wish = y ]
        then
                continue
        else
                echo "The contents of the file is:`cat names`"
                exit
        fi
done
```

Important Shell Script Considerations

- Usage of echo statements to get the values.
- All error messages should be directed to the Standard Error.
- Work comfortable with vi editor

Background Processing

- **Unix provides us the facility to background the process that are taking long time to process to help us multitask.**
- **& usage**

  $ ./startWeblogic.sh &
- **ps –ef to obtain the process status**

  $ ps –ef | grep peter

  peter 31696 31694 0 22:20 ? 00:00:00 [sshd]

  peter 31697 31696 0 22:20 pts/0 00:00:00 –bash

  peter 31829 31827 0 23:53 ? 00:00:00 [sshd]
- **nohup command**

  $ nohup ./startWeblogic.sh &

Or# nohup bash startWeblogic.sh &

–use nohup utility which allows to run command/process or shell script that can continue running in the background after you log out from a shell:

- nohup Syntax:
- nohup <command-name> &
- **Terminating the background process**

  $ kill –9 31696

crontab

- To list the cron entries

$crontab -l

```
#################################
# minute   hour   Day of Month  Month  Weekday command
#(0-59)   (0-23)     (1-31)    (1-12)  (0-6;0 being Sunday)
#################################################################
   ##############################
#WEEKLY RESTART THE BIG BROTHER SOFTWARE-r.init rest
0 2 * * 0 sh /path/to/your/script/scriptfile.sh 2 >&1
#################################################################
   ##############################
```

- To edit the cron entries

$ crontab -e

Every Sat: 2 am.

0 2 * * 6 mkdir -p /test

Additional Unix Commands

- tar
- compress and uncompress
- gzip and gunzip
- sed and awk
- telnet and ftp
- ssh and scp
- mailx , mail
- write

tar

- To tar -cvPzf S3S.tar.gz S3S
- To untar: tar -xvPzf S3S.tar.gz
- To archive the files in a directory
  - $ tar -cvf <archivename>.tar <directory name>
- To view the files in a tar archive
  - $ tar -tvf <archivename>.tar
- To extract the files in a tar archive
  - $ tar -xvf <archivename>.tar

compress and uncompress

- To compress a file
  - $compress <filename>

  The extension will be a .Z file
- To uncompress a file
  - $uncompress <compressed file name>

gzip and gunzip

- To compress a file
  - $ gzip <filename>
  - Ex:

```
$ ls -l httpd.conf.org
-rw-r--r--   1 peter   peter     34935 Feb 19 20:17 httpd.conf.org
$ gzip httpd.conf.org
$ ls -l httpd.conf.org.gz
-rw-r--r--   1 peter   peter     12389 Feb 19 20:17 httpd.conf.org.gz
```

- To uncompress a file
  - Ex:

```
$ gunzip httpd.conf.org.gz
$ ls -l httpd.conf.org
 -rw-r--r--   1 peter   peter     34935 Feb 19 20:17 httpd.conf.org
```

sed and awk

- sed
  - A non-interactive stream editor since the input can come in from a program and be directed to a standard output
  - Use sed:
    - To automate editing actions to be performed on one or more files.
    - To simplify the task of performing the same edits on multiple files.
    - To write conversion programs.

sed

- **$ more mailist or cat maillist**

peter Daggett, 341 King Road, Plymouth MA
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury MA
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston MA

$ sed 's/MA/Massachusetts/' mailist

peter Daggett, 341 King Road, Plymouth Massacussets
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury Massacussets
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston Massacussets

sed

- **$ more mailist**

peter Daggett, 341 King Road, Plymouth MA
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury MA
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston MA

$ sed 's/MA/, Massachusetts/' mailist

peter Daggett, 341 King Road, Plymouth , Massachusetts
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury , Massachusetts
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston , Massachusetts

sed

- **Multiple instructions at command line**

  There are three ways to specify multiple instructions on the command line:

  - **Separate instructions with a semicolon.**

    $sed 's/ MA/, Massachusetts/; s/ PA/, Pennsylvania/' mailist

  - **Precede each instruction by -e.**

    $sed -e 's/ MA/, Massachusetts/' -e 's/ PA/, Pennsylvania/' mailist
    (wont' show you the o/p in screen with -e)

  - **Pressing a return after entering a single quote**

    $ sed ' s/ MA/, Massachusetts/
    > s/ PA/, Pennsylvania/
    > s/ CA/, California/' mailist

sed

- sed with script files

- **$ cat sedcmd**

s/ MA/, Massachusetts/
s/ PA/, Pennsylvania/
s/ CA/, California/
s/ VA/, Virginia/
s/ OK/, Oklahoma/

$ sed –f sedcmd mailist (here you are applying sedcmd file against maillist file)

peter Daggett, 341 King Road, Plymouth, Massachusetts
Alice Ford, 22 East Broadway, Richmond, Virginia
Orville Thomas, 11345 Oak Bridge Road, Tulsa, Oklahoma
Terry Kalkas, 402 Lans Road, Beaver Falls, Pennsylvania
Eric Adams, 20 Post Road, Sudbury, Massachusetts
Hubert Sims, 328A Brook Road, Roanoke, Virginia
Amy Wilde, 334 Bayshore Pkwy, Mountain View, California
Sal Carpenter, 73 6th Street, Boston, Massachusetts

awk

- **$ more nameslist**

peter Daggett, 341 King Road, Plymouth MA
Eric Adams, 20 Post Road, Sudbury MA
Sal Carpenter, 73 6th Street, Boston MA
Bill Gates, 1 Network Drive, Redwood, WA
Scott McNeally, 1 Network Drive, Burlington, MA

- **$ awk '{ print $1 }' nameslist**

peter
Eric
Sal
Bill
Scott

awk

- **$ awk '/MA/‘ nameslist**

peter Daggett, 341 King Road, Plymouth MA
Eric Adams, 20 Post Road, Sudbury MA
Sal Carpenter, 73 6th Street, Boston MA
Scott McNeally, 1 Network Drive, Burlington, MA

- **$ awk '/MA/ { print $1 }' nameslist**

peter
Eric
Sal
Scott

awk

- **To change the field separator to ‘’,”**

$awk -F, '/MA/ { print $1}‘ nameslist or awk –F “,” ‘{print $1}’

peter Daggett
Eric Adams
Sal Carpenter
Scott McNeally

- **Multiple commands are separated by semicolons.**

$awk -F, '{ print $1; print $2 }' nameslist

peter Daggett
341 King Road
Plymouth MA
Eric Adams
20 Post Road
Sudbury MA
Sal Carpenter
73 6th Street
Boston MA

Using awk and sed together

- **$ more mailist**

peter Daggett, 341 King Road, Plymouth MA
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury MA
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston MA

- **sed -f sedcmd mailist | awk -F, '{ print $1 }'**

peter Daggett
Alice Ford
Orville Thomas
Terry Kalkas
Eric Adams
Hubert Sims
Amy Wilde
Sal Carpenter

Shell Variables – this slide is moved down.

Special Shell Variables

- $# – The number of positional parameters.
- $- – Shell options
- $? – The exit status of the last executed command
- $$ – The process id of the current shell (echo $$).
- $! – The process id of the last background process.
- $0 – The name of the command being executed.
- $* – The list of positional parameters.
- $@ – Same as $*, except when enclosed in double quotes. All arguments, as separate words.

```
#!/bin/sh (this is a test.sh file)
echo "File Name: $0"
echo "First Parameter : $1"
echo "Second Parameter : $2"
echo "Quoted Values: $@"
echo "Quoted Values: $*"
echo "Total Number of Parameters : $#“
```

- echo $? (The **$?** variable represents the exit status of the previous command.
- Exit status is a numerical value returned by every command upon its completion. As a rule, most commands return an exit status of 0 if they were successful, and 1 if they were unsuccessful.

```
[s3s@dcserver tmp]$ bash test.sh biraj p
File Name: test.sh
First Parameter : biraj
Second Parameter : p
Quoted Values: biraj p
Quoted Values : biraj p
Total Number of Parameters : 2
```

For Advance Scripting Read on: http://www.tutorialspoint.com/unix/unix-what-is-shell.htm

telnet and ftp

- **telnet 192.168.10.60**

telnet 192.168.10.60
Trying 192.168.10.60...
Connected to 192.168.10.60.
Escape character is '^]'.
login:

- **> ftp 192.168.10.60**

Connected to 192.168.10.60
220 192.168.10.60 FTP server (SunOS 5.8) ready.
Name (192.168.10.60:user1): user2
331 Password required for user2.
Password:
230 User user2 logged in.
ftp> ls
200 PORT command successful.
150 ASCII data connection for /bin/ls (113.128.142.1,54452) (0 bytes).

prstat.out
neil
226 ASCII Transfer complete.
169 bytes received in 0.0043 seconds (38.32 Kbytes/s)

telnet and ftp

```
ftp> bin
200 Type set to I.
ftp> !pwd
/
ftp> !ls
bin
cdrom
core
cpf_install.pdf
dev
devices
etc
export
ftp> mget neil
200 PORT command successful.
150 Binary data connection for neil (113.128.142.1,54453) (9 bytes).
226 Binary Transfer complete.
local: neil remote: neil
9 bytes received in 0.0013 seconds (7.01 Kbytes/s)
ftp>
```

ssh and scp

- ssh

> ssh peter@192.168.10.60

peter's password:
Authentication successful.
Last login: Wed Dec 10 2003 14:39:50 -0600 from 192.168.2.31
Sun Microsystems Inc. SunOS 5.8 Generic Patch October 2001
No mail.
Sun Microsystems Inc. SunOS 5.8 Generic Patch October 2001
There are now 15 users on 192.168.10.60.
%

- scp

peter@server1:>scp neil peter@192.168.10.60:/home/peter/

peter@192.168.10.60's password:
neil | 9B | 9B/s | TOC: 00:00:01 | 100%
peter@server1:>

mailx and mail

mailx <- Used in Solaris

mailx -s "This is a test mail" sp@vi.com

This is a test mail for peter

.

mail <- Used in Linux

mail -s "This is a test mail" sp@vi.com

This is a test mail for peter. (Press ctrl d).

If your smtp is running on port 25 you should receive the email from local machine.

write

- write

```
[root@s3s root]# who
peter   pts/0        May 31 12:24 (192.168.0.2)
root    pts/1        May 31 12:25 (192.168.0.2)
[root@s3s root]# write peter
Hi peter,
How are you?
Please log off as we'll be rebooting the system.
.
[root@s3s root]#
##################################################
[peter@s3s peter]$
Message from root@s3s on pts/1 at 12:25 ...
Hi peter,
How are you?
Please log off as we'll be rebooting the system.
.
EOF
```

Questions or concerns? Please feel free to bring up.

Always anticipating your feedback.