
User Level Unix

Agenda

- Introduction to Unix operating system
- Understanding Kernel and shell.
- Booting process.
- Understanding the file and directory structure.
- Commands.
- inittab and crontab.
- Configuration files.
- Shell variables
- vi editor
- Shell scripting.
- Exercises

Out of Scope

- Systems programming.
- Administrator level commands
- Network level commands
- Configuration aspects.
- Programming.

Introduction

- UNIX operating system is made up of three parts; the kernel, the shell and the programs.

Kernel

- Core of the Unix system.
- Interacts directly with the OS
- Largely written in C, with some parts written in Assembly language.
- Insulates other parts of OS from hardware.
- Performs all low level functions.
- All the applications deal with Kernel.
- Handles all the devices.

Kernel Functions

- Memory management.
- Process scheduling.
- File management and security.
- Interrupt handling and error reporting.
- Input / Output services.
- System accounting

Shell

- Acts as an interface between the user and the kernel.
- Is the command interpreter of UNIX.
- Provides powerful programming capabilities.
- System administration.
- Text processing.
- File management.
- UNIX Utilities and Applications.
- User can customize his/her own shell.
- Users can use different shells on the same machine.

Shell

Some of the common features that most all shells have to offer:

- Filename completion
- Command-line history.
- Multiple job control.
- Redirection (program to file).
- Pipes (program to program).
- Storage variables.

Booting Process

- Hardware configuration check.
- The operating system file /vmunix (the UNIX kernel) gets loaded.
- The kernel brings up the swapper and the init program.
- Jobs of init are:
  - Takes the system into the multi-user mode.
  - Checks the integrity of the file systems.
  - Mounts the file systems
  - The appears. login process gets executed.
  - After successful login, the shell

Run Levels

- UNIX system has several modes of operation called system states (or run levels). These are:

0	: shutdown state
1	: administrative state
s or S	: single user state
2	: multi-user state
6	: stop and reboot state

- These states are passed to the init program, which operates the system in the appropriate state.

Files

- A file is a collection of data. They are created by users using text editors, running compilers etc.
- UNIX stores all files in an identical manner.
- File contents are treated as series of bytes.
- Devices are also treated as special file.
- File sizes can grow dynamically (limited only by disk space available).
- Internally each file is assigned a unique identification number called Inode.
- Directories are also files of special type.
- Provides security to file access.

File Types

Unix recognizes the following types of files:

- **Regular file**: Text or binary data (e.g. an image).
- **Directory**: Contains other files and directories.
- **Executable file**: File with the "execute bit" set. Most commands are executable files.
- **Symbolic link**: File is a "shortcut" that points to another file.
- **Device special file**: An interface to a piece of hardware, such as a printer. You don't have to worry about this.
- **Named pipe**: An interface to a network program. You don't have to worry about this.

File Naming Conventions

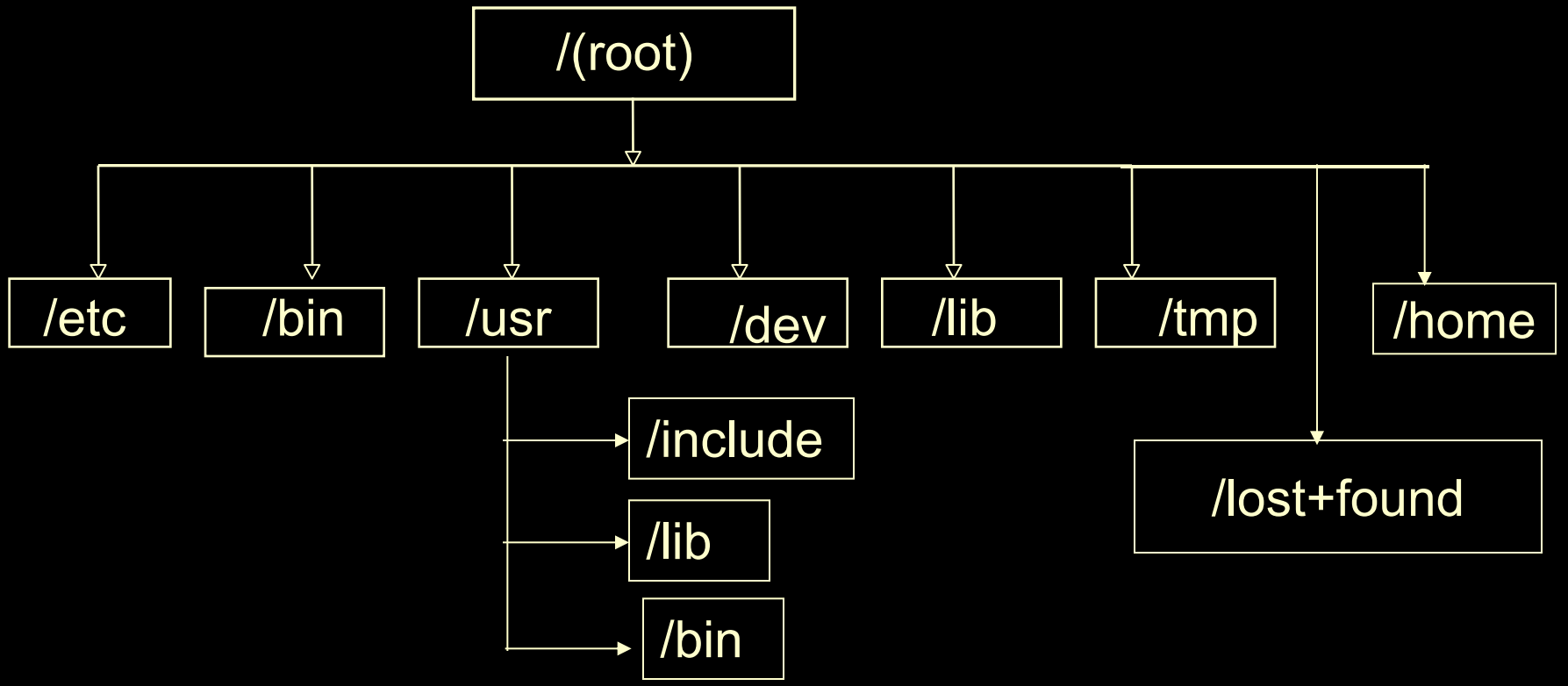
- Maximum file length depends upon the kernel configuration. This is contained in the system structure statvfs. In our system it is 255.
- No concept of primary and secondary name.
- File names are case sensitive.
- Name may contain alphabets, digits, dots and underscores.
- Embedded space and tab names are not allowed.

Directory

- All the files are grouped together in the directory structure.
- The file-system is arranged in a hierarchical structure, like an inverted tree.
- The top of the hierarchy is traditionally called root.
- Each directory on a UNIX file system contains information about its contents, which may be a mixture of other directories (sub-directories) and/or ordinary files.

Directory Structure

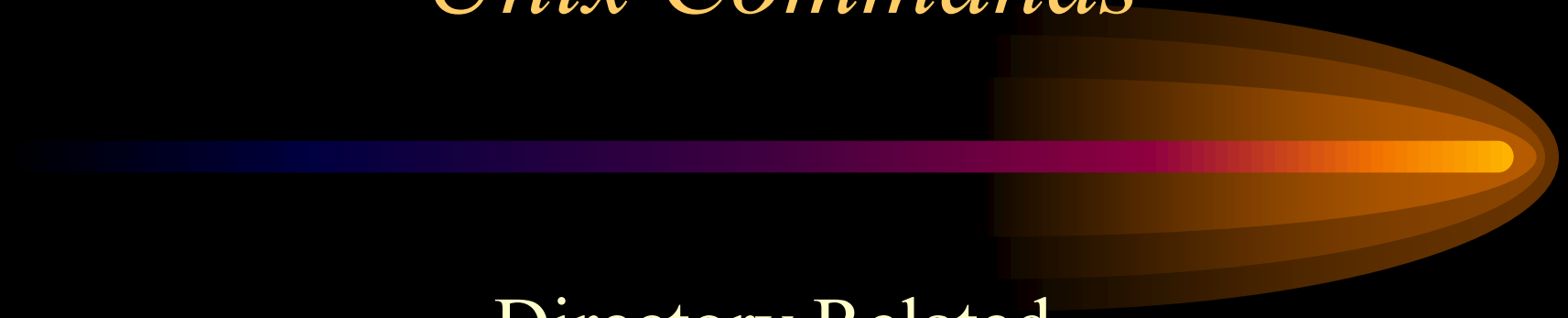
- Each brand of UNIX has their own method of laying out the operating system directory structure, however, most UNIX's lay them out something like this:



Contents

Directory	Typical contents
/	The top of the directory hierarchy, traditionally called the "root" directory.
/etc	UNIX System configuration and information files
/usr	Stuff intended more for the users using the system than for the UNIX system itself.
/bin, /usr/bin	Programs for low-level system functions and higher-level user functions respectively
/lib, /usr/lib	Program libraries for low-level system programs and higher-level user programs respectively
/tmp	Temporary file storage space which any of the users on the system can use
/home	Contains the users' home directories. Each directory is usually named after the username of the user

Unix Commands



Directory Related.

- Creation of directory

$ mkdir <pathname>

- Change directory

$ cd <directory>

- Remove directory

$ rmdir <directory>

- Print Current working directory

$ pwd

Pathname

- Pathname gives the exact location of a file in a tree hierarchy.
- Absolute pathname: Referring to a file from the root directory.
- Relative pathname: Referring to a file from the current directory.
- '.' (Single dot) represents the current directory.
- '..' (double dots) represent the parent directory.
- '~' (tilde) refers to the home directory.

ls – list directory

- The ls command lists the contents of your current working directory.
- ls does not, in fact, cause all the files in your home directory to be listed, but only those ones whose name does not begin with a dot (.)
- Files beginning with a dot (.) are known as hidden files and usually contain important program configuration information.

ls – Contd…

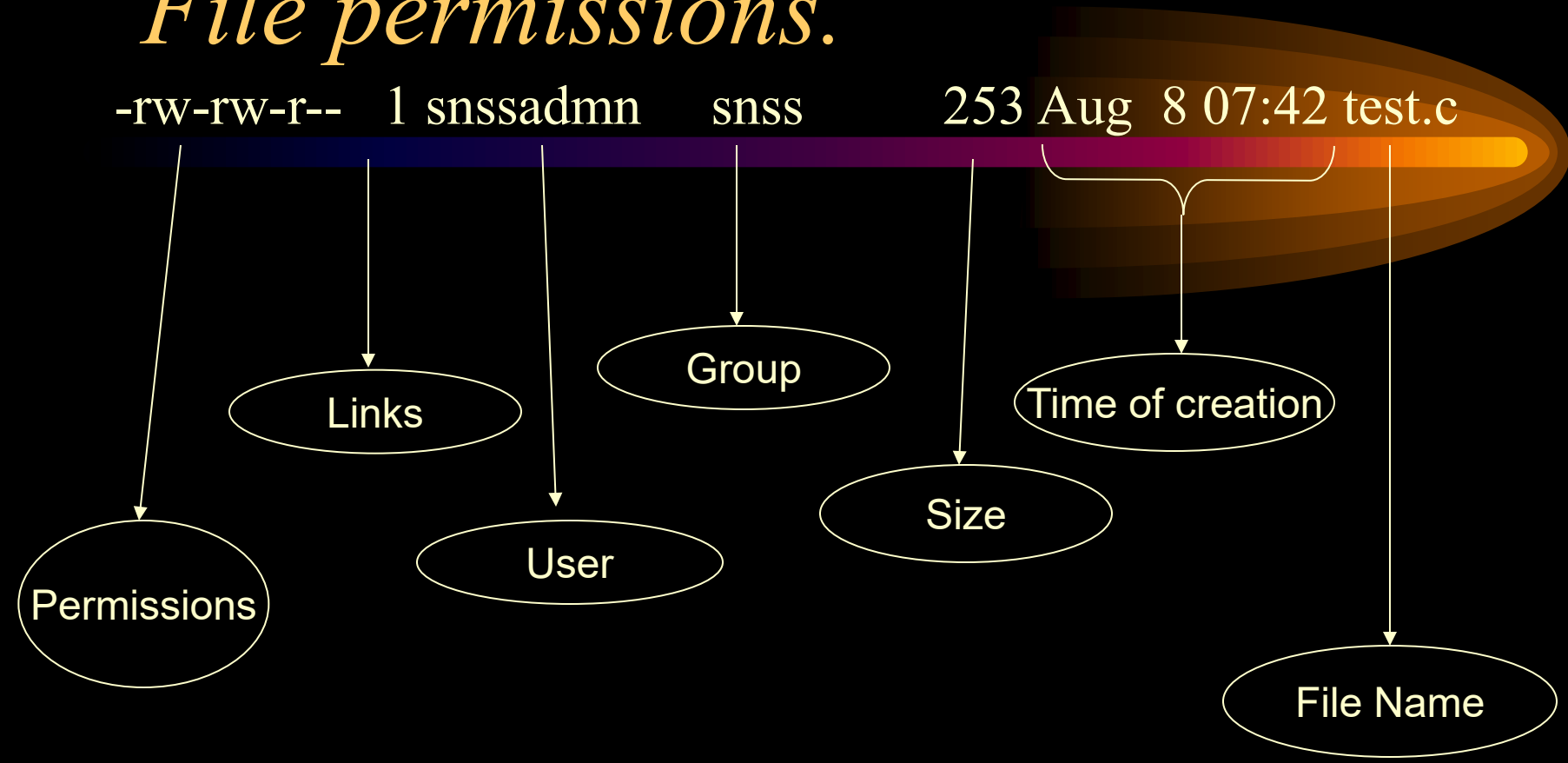
$ ls [option...] [pathname]

-l	list in long format.
-a	all files including those which begin with a dot (.)
-x	sorts multi column output across the page
-c	displays files in columns (wider format)
-R	recursively list all directories and sub-directories
-t	
-u	Use time of last access instead of last modification for sorting (-t option) or printing (-l (ell) option).
-F	regular files are listed normally, directories get a "/" appended to the end, symbolic links get an "@" appended to the end, and executable files get a "*".

Unix Commands

Permissions

File permissions.



Permissions – Contd..

drwxr-xr-x 2 snssadmn snss 1024 Jul 30 04:59 debug/

flags	owner	group	world
d	rwx	r-x	r-x

Permissions – Contd..

Flags:

"d" = directory

"l" = link

"p" = pipe

"b" = block special device

"c" = character special device

Owner: Permissions for the owner of the file/directory:

"r" = read permission

"w" = write permission

"x" = execute permissin

Group: Permissions for the group owner of the file/directory.

World: Permissions for everyone else

Permissions – Contd..

- For files:
  - read, write, and execute permission do what you think they might do.
- For directories:
  - read permission lets you list the contents of the directory.
  - write permission lets you create or delete files in the directory.
  - execute permission lets you cd to the directory.

Changing Permissions

- chmod
- chown
- chgrp

chmod:

- Used to change the permissions of files and directory.
- Only the owner or super user can use chmod to change the permissions of a file.

Options with chmod..

Symbol	Meaning
u	user
g	group
o	other
a	all
r	read
w	write (and delete)
x	execute (and access directory)
+	add permission
-	take away permission

Using numbers

- 1 - Execute
- 2 - Write
- 4 – Read

Example:

Original permission for ae_socket.c:

-rw----rw-

Thus in order to give read write permission to owner, group and only read permission to others we would give:

Using character modes:

```
chmod o-w 1.c
chmod g+rw 1.c
```

Using numbers:

```
chmod 664 ae_socket.c
```

Changing Permissions….

- **chown**

Change the owner of the file.

$ chown <ownername> <filename>

Example:

```
$ chown ashish ae_socket.c
```

- **chgrp**

Change the group of the file.

$ chgrp <groupname> <filename>

Example:

```
$ chgrp dev ae_socket.c
```

Unix Commands

File handling Related.

- **cp**

Command makes a copy of the existing file.

The general command line format is:

$ cp <file1> <file2> <Enter>

Where: file1 is the source file and file2 is the destination file.

i	Interactive
p	Preserve permission
r	Recursive Copy
f	Forcefully

- **mv**

Command moves a file from on place to another. It is also used for renaming.

The general command line format is:

$ mv <file1> <file2> <Enter>

Where: file1 is the source file and file2 is the destination file.

- **rm**

Command deletes the existing file.

i	Interactive
r	Recursive
f	forcefully

The general command line format is:

$ rm <file1> <file2> <Enter>

Multiple files can be deleted in one command.

- **cat**

Command used to display the contents of the file.

The general command line format is:

$ cat <file name> <Enter>

Example:

```
cat ae_socket.c
```

To create a file:

```
cat mytest
This is a test file,
that I am creating
<ctrl> D
```

- **ln**

Command creates multiple links to file.

The general command line format is:
$ ln <old file> <linked file> <Enter>
Multiple files can be deleted in one command.

Example:

```
ln ae_socket.c new_file.c
```

This will create a link new_file.c pointing to the file ae_socket.c.

i	Interactive
s	Symbolic
f	forcefully

- All links to a file access the same physical file.
- Modifying contents from any access changes the file.
- Links can be of two type:
  - Hard link.
    - Created using **`ln <old file> <link name>`**
    - Cannot create a link to directory.
    - Can not span file systems.
  - Soft Link.
    - Created using **`ln -s <old file> <link name>`**
    - Can also create link to directory.
    - Can span file systems.

- **head**

Command by default displays first 10 lines of the file.

The general command line format is:

$ head [-n] <file> <Enter>

Example:

`head ae_socket.c` → `Will display first 10 lines`

`head -5 ae_socket.c` → `Will display first 5 lines`

- **tail**

Command by default displays last 10 lines of the file.

The general command line format is:

$ tail [+/-n] <file> <Enter>

Example:

```
tail ae_socket.c      → Will display last 10 lines
tail -5 ae_socket.c   → Will display last 5 lines
tail +5 ae_socket.c   → Will display lines starting
   from 5th line
                        till end of file
```

- **touch**
- Updates the access, modification, and last-change times of each argument.
- The file name is created if it does not exist.
- If no time is specified, the current time is used.

The general command line format is:
$ touch <file>

Example:

touch ae_socket.c → Will set the modification time to current time.

- **Options with touch:**

-a	Change the access time.
-m	Change the modification time.
-c	Prevent touch from creating a file. If it does not exist previously
-t <time>	Use the specified time instead of current time. The format is: [[CC]YY]MMDDhhmm[.SS] *CC – First two digits of year.* *YY – Last two digits of year* *MM – month of the year (01-12)* *DD – day of the month (01-31)* *HH - hour of the day (00-23)* *MM - minute of the hour (00-59)* *SS - second of the minute (00-61)*

- **wc**

Command counts the number of words, characters and lines present in a file.

The general command line format is:

$ wc [-c] [-m] [-lw] <file> <Enter>

Example:

wc ae_socket.c → Will display output like:

```
                342    1028      7861
  ae_socket.c
                Lines  Words   Bytes/characters
```

- **file**

  This classifies the named files according to the type of data they contain. For example ascii (text), pictures, compressed data, etc.

The general command line format is:

$ file <file> <Enter>

Example:

`file ae_socket.c` → `The output will be:`

`ae_socket.c: c program text`

- **find:**
- Used to find a file in directory(s).
- find command recursively descends the directory hierarchy.
- By default, find does not follow symbolic links.

The general command line format is:

find pathname_list [expression]

Example:

`find ./ -name ae_socket.c` → `This will start searching from the current directory. It will recursively descend to subdirectories and print out the status.`

- **Options with find:**

-depth	Search in subdirectory also (default).
-name file	Name of file to search.
-path file	Same as -name except the full path is used instead of just the base name.
-type c	Specify which type of file to search: **f – Regular file, d – directory, b – block special file,** **c – character special file, p – FIFO (named pipe), l – symbolic link** **s – socket, n – network special file, m – mount point**
-print	Causes the current path name to be printed (default).
-exec cmd	Execute the command cmd.

Wild Cards

- ls *
- “*“-- Match any string of zero or more characters in file name.
- “?“ -- Match any single character.
- [list] -- Match any one character from the list.

- **"*" metacharacter:** Called a wildcard, and will match against none or more character(s) in a file (or directory) name

`ls *.c` will display all files ending with '.c‘

- **“?” metacharacter**: ? will match exactly one character.

`ls -l chap?` will display a long listing of all the files whose first four characters are "chap" followed by any single character.

`ls?.?` will list all three-character filenames with an embedded ".".

- **“[]” metacharacter**: Matches one of the characters included inside the [ ] symbols

`ls chap[123]` will display files named as chap1, chap2 and chap3 only.

`ls chap[A-Z]` will display the files chapA, chapB, chapC, chapD.....chapZ only.

Unix Commands

Compare and Contrast.

Compare and Contrast

- **comm**

Display common lines in two files.

Produces a three-column output:

Column 1: Lines that appear only in file1,
Column 2: Lines that appear only in file2,
Column 3: Lines that appear in both files.

The general command line format is:
$ comm [-[1] [2] [3]] file1 file2

- **cmp**
- Compare two files.
- No comment if the files are the same.
- If they differ, it announces the byte and line number at which the difference occurred.

The general command line format is:
$ cmp file1 file2

- **diff:**
- Compare directories:
  - diff sorts the contents of the directories by name.
  - then runs the regular file diff algorithm on text files that have the same name in each directory but are different.
- Compare file:
  - diff tells what lines must be changed in the files to bring them into agreement.
  - Option:
    - -b: ignores trailing blanks (spaces and tabs) and other strings.
    - -i: ignore uppercase and lower case differences.
    - -w: ignore all white spaces.

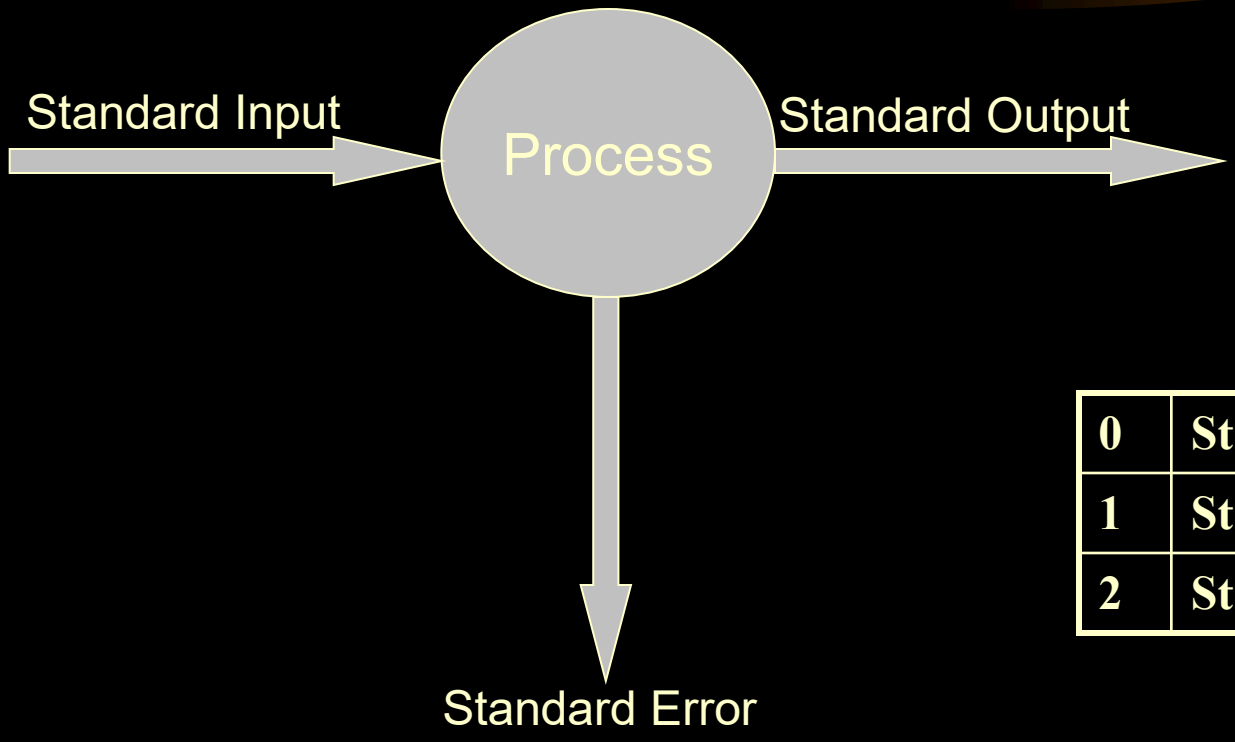
Unix Commands

Redirection

Redirection

- For every process in UNIX, three files are automatically opened: standard input, standard output, and standard error.
- By default, the command reads the input from the standard input file, gives the output to standard output file and the errors to standard error file.
  - Standard input (sometimes abbreviated as ***stdin***) is usually mapped to the keyboard.
  - Standard output (sometimes abbreviated as ***stdout***) is usually mapped to the screen.
  - Standard error (sometimes abbreviated as ***stderr***) is similar to standard output, except that it is reserved for error messages.

- *You can use redirection to send the input or output of a process to a location other than the standard one*



0	**Standard Input**
1	**Standard output**
2	**Standard Error**

- Redirecting the output:
  - > symbol is used to redirect the output of a command.
  - For example

  To redirect the output of ls command to a file rather then stdout.

  ```
  $ ls -l > out.txt
  ```

  >> is used to append the file.

- Redirecting the input:
  - < symbol is used to redirect the input of a command.
  - For example

  Sort command sorts the input given to it from stdin. To read the input from a file file_a:

  ```
  $ sort < file_b
  ```

- Redirecting the error:
  - By default it is assigned to terminal.
  - > symbol is used to redirect using the file descriptor 2.
  - For example

To redirect all the error messages to a file instead of terminal:

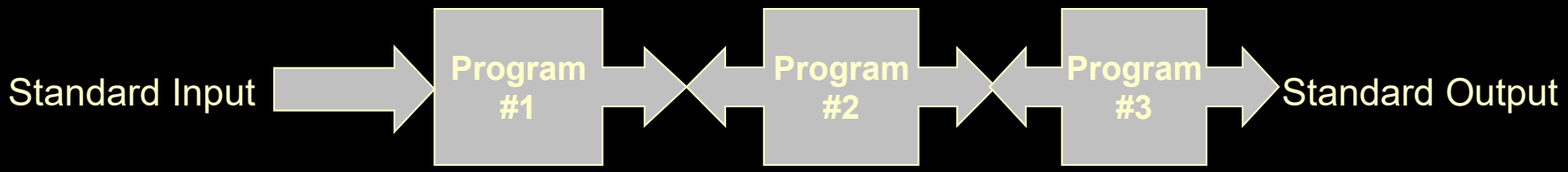
```
$ ls -l test.c 2> err.txt
```

Unix Commands

Pipe and Filters.

Pipe

- Connects two or more commands together.
- Output of first goes to the second.
- This allows for sequencing the commands together.



- Denoted by symbol | called the pipe.

Example:

```
$ sort < names.txt
```

Using pipe:

```
$ who | sort
```

Filter

- Filter is a command that takes its input from standard input, processes (or filters) it and sends its output to the standard output.
- Commands such as ls, date, pwd etc. can not be used as filters as they do not require any input.
- Some commonly used filters are:
  - grep, sort, cut, paste, head, tail, wc, pg, more, tee, tr.

grep

- **grep:**
- *Global Search for Regular Expression and Print* utility, is the standard file searching and selection utility.
- Used to search for a particular pattern of characters and display all the lines containing the pattern.
- Examines the input file line by line for the given pattern. When a match is found, the line is copied to the standard output file.
- No output is produced when the desired pattern is not found.

The format of the grep command is:

$ grep [option] <pattern> <filename>

- Options:
  -n : prints line numbers
  -v : the reverse search criterion
  -c : display only a count of matching patterns

- Regular expressions:
  “^“ : beginning of line
  "$" : end of line
  "." : any single character
  [...] : any one character from the list
  [^...] : String beginning from any characters after ^ will be excluded.
  \ : Used in conjunction with above. Negates meaning of special characters following it.

Example:

Display the line(s) containing the string aethos.

```
$ grep "aethos" ae_socket.c → No Output
$ grep "AETHOS" ae_socket.c → Correct
  output. Since grep is case sensitive.
```

Now give:

```
$ grep -i "aethos" ae_socket.c → Correct
  output
```

To search for a phrase or pattern, you must enclose it in single quotes

```
$ grep -i 'Lookup services' ae_socket.c
```

- **Options with grep:**

-v	display those lines that do NOT match
-n	precede each matching line with the line number
-c	print only the total count of matched lines
-i	Ignore case
-l	Only name of files with matching lines are displayed.

grep – Examples

- Consider a file containing employee details.

To display the records of the employee whose last name begins with B

```
$ grep "^B" employee
```

Display the records of all the employees whose basic pay is between 6000 and 6999

```
$ grep "6...$" employee
```

To display records of all employees whose employee code is between 9000 and 9099

```
$ grep "90[0-9][0-9]" employee
```

- Using grep as filter:

To find out if a given user has logged in:

```
$ who | grep "snssadmn"
```

Other variants of grep:

- egrep
- fgrep

Sorting

- **sort:**
- Takes lines from input file and sorts them.
- Displays the output on the standard output device.
- By default the sorting is done on the first field in ascending order.
- Each line from the input file is treated as a series of one or more fields separated from each other by a space, or tab by default or any other field delimiter if specified.

The command line format is:

$ sort [- option] [+pos] [-pos] [-o output file name] [filename]

sort - Example

- Let us take a file containing names.

To sort the names alphabetically:

```
$ sort names.txt
```

To sort in reverse order:

```
$ sort –r names.txt
```

To sort the file in numeric order rather than ASCII mode, we use -n option (note that this is required when the entries in the number field are not right justified):

sort - Options

- To arrange each record in order of a specific field and not in the usual order of the first character onwards we must specify the field using a command line format as follows:

$ sort [+pos1] [-pos2] <filename>

Where:

- sort skips pos1 fields and stops considering fields after pos2th field.
- If pos2 is omitted it extends the fields from pos1 to the end of the line.

sort - Example

- By default, spaces and tabs are considered as the field separators.
- The -t option is used to specify the field delimiter, in case the delimiter is other than a tab or space character.

Example:

The passwd file in /etc directory has colon (:) as field separator. Therefore, to sort this file on the third field we must specify the field separator as colon.

```
$ sort -t ":"  +2 -3 /etc/passwd
```

User-id:password:numeric user id:numeric group id:real name:home directory:shell

cut

- **cut:**
- To retrieve selected fields from a file.
- Useful when data is in tabular format (rows and column).
- This command picks up selected columns or fields from a file, by specifying a list of character positions or a list of fields.

The command line format is:

$ cut [options] <filename> <Enter>

Options:

-c	selects columns specified by list
-f	selects fields specified by list
-d	field delimiter (default is tab)

cut - Example

Examples:

To display the employee code, firstname and basic salary from the employee file created earlier.

```
$ cut -d" " -f2,3,8  emp_new.txt
```

Where:

-d is used to specify the field delimiter. A space in this case.

-f represents fields and the numbers after it are the field positions.

The specified columns are copied to the standard output.

cut - Examples

- To display characters 4 to 9 both inclusive, from each record of the file employee:

```
$ cut -c4-9  emp_new.txt
```

- To display from the second to the fourth fields:

```
$ cut -d" " -f2-4 emp_new.txt
```

- To display from the fourth field onwards:

```
$ cut  -d" " -f4- emp_new.txt
```

paste – Horizontal merging of files

- **paste:**
- prints files side by side in a columnar manner.
- takes two tables and combines them side by side forming a single wide table as the output.
- This command uses tab as a field delimiter. However, with the -d option a different delimiter can be given

The command line format is:

$ paste [option] <filename1> <filename2> <Enter>

Options:

-d field delimiter (default is tab)

paste – Example

- Example:

Consider the files fruit and pricelist containing the following data.

fruit.txt	price.txt
Apple	45
Orange	21
grapes	69

To print the fruit name along with their price, we need to paste the two files:

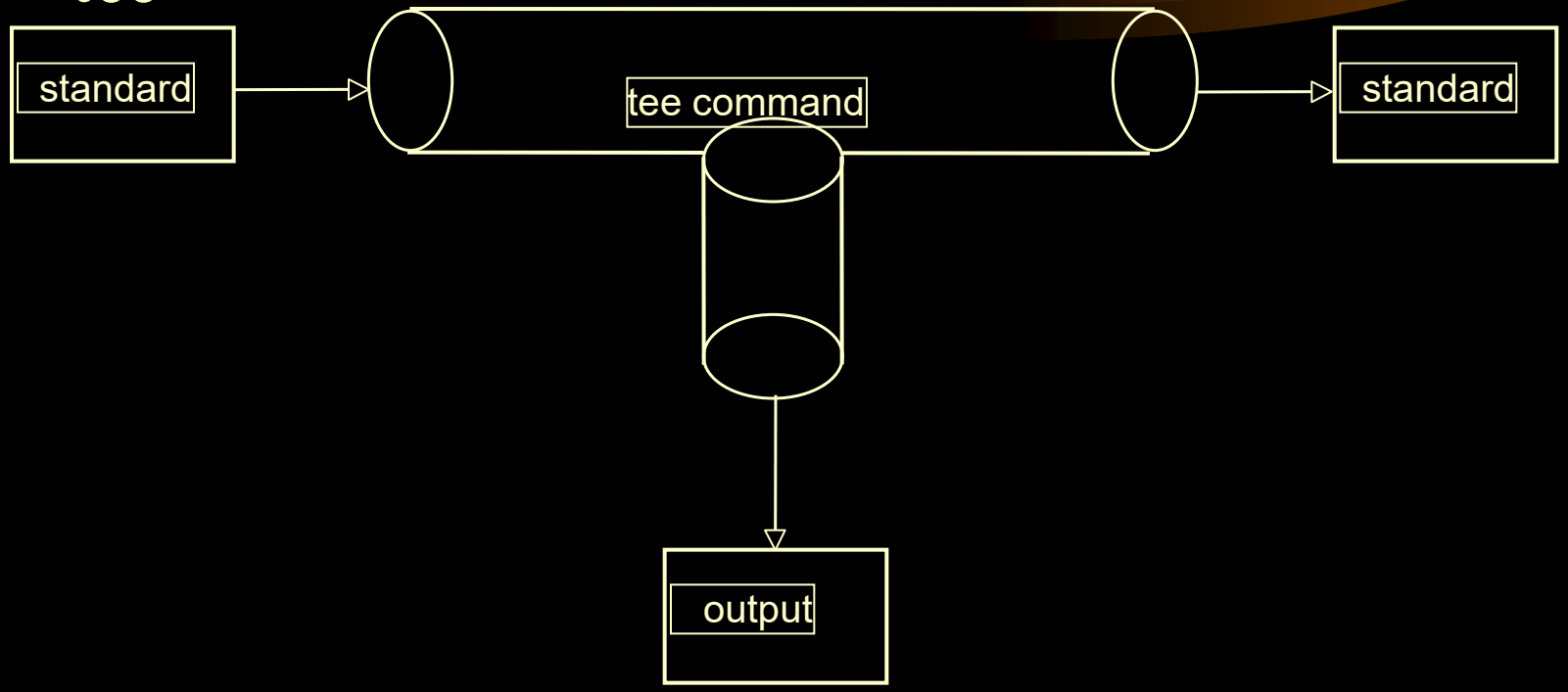
```
$ paste fruit.txt price.txt
```

tee – Pipe Fitting

- In case the output is required to be sent to two different destinations at the same time, the redirection operator can not be used.
- tee command is used to send the output to more than one destination.
- tee transfers data without any change from its standard input to its standard output, but at the same time it directs a copy of the data into another file, that is specified at the command line.
- can be used anywhere in a pipeline.

tee

- **tee**



tee – Example

$ <command1> | tee <file> | <cmmand2>

The standard output of the command1 becomes the standard input for tee. This input is then passed on as the standard input for command2 and a copy is diverted to file at the same time.

For example, the command:

```
$ sort names.txt |  tee tempfile  |  pg
```

Will sort the employee file, display the result on the screen pagewise and store it in tempfile.

tr – Translate Characters

- **tr:**
- Used to squeeze the repetitive characters from the input and translate the input to some other form.
- Useful for converting files between Dos/Windows and Unix.
  - Dos/Windows convention; Each line ends with carriage return followed by new line character.
  - Unix convention: Each line ands with line feed
  - `$ tr -d '\r' < dosfile > unixfile`

The format of tr command is:

$ tr <str> <newstr> < <file|stdin>

tr - Example

Examples:

- To convert all lower case alphabets into upper case and display them on the terminal:

```
$ cat  emp.txt  | tr "[a-z]" "[A-Z]"
```

- Display all the occurrences of small "a" as "A":

```
$ ls -l | tr "a" "A"
```

- To produce the file size and names in descending order of size:

```
$ ls -l | tr -s " " | cut -d" " -f 5,9 | sort
  -nr
```

Note: The output of ls -l is in a tabular form, where the columns are separated by one or more spaces, so we used –s to squeeze the extra space.

Beauty of Filters and Pipe

- Question:

Count the frequency of different words appearing in a text file. wc command is not be be used.

Solution:

```
$ cat poem | tr -s " " | tr " "   "\012" | sort |
  uniq –c
```

- The cat command will produce the input for the first tr command
- All the consecutive spaces are squeezed into a single space.
- The second tr will translate each space into a new line character, thus producing one word per line.
- sort command will sort the result and pass it to the uniq command
- uniq will count the number of occurrences of each word and prints the number along with the word.

File perusal filters

- pg
- more
- page

Unix Commands

Process Handling

View the Processes

- **ps:**
- A process is an executing program identified by a unique PID (process identifier).
- To see information about your processes, with their associated PID and status, type:
  - `$ ps`

-e	Select all processes.
-f	Show columns user, pid, ppid, cpu, stime, tty, time, and args, in that order.
-l	Show columns flags, state, uid, pid, ppid, cpu, intpri, nice, addr, sz, wchan, tty, time, and comm, in that order.
-x	Shows the command line in extended format.

Background jobs

- To background a process, type an & at the end of the command line.

For example, the command sleep waits a given number of seconds before continuing. Type

```
$ sleep 10
```

- This will wait 10 seconds before returning the command prompt $. Until the command prompt is returned, you can do nothing except wait.

To run sleep in the background, type:

```
$ sleep 10 &
```

- The & runs the job in the background and returns the prompt straight away, allowing you do run other programs while waiting for that one to finish.

- **nohup:**
- The Kernel terminates all active processes by sending a hang-up signal at the time of logout. Nohup command is used to continue execution of a process even after the user has logged out.

Format of command:

nohup <command> <Enter>

- Any command that is specified with nohup, will continue running even after the user has logged out.

- **Chaining multiple commands:**

$ command1;command2;command3

- You can suspend the process running in the foreground by holding down the [control] key and typing [z] (written as ^Z) Then to put it in the background, type

  `$ bg`

  Note: Do not background programs that require user interaction

- **top:**

  Display and update information about the top processes on the system.

- To see the background jobs running in the terminal:

$ jobs

- To restart (foreground) a suspended processes, type

$ fg %jobnumber

- For example, to restart sleep 100, type

```
$ fg %1
```

- Typing fg with no job number foregrounds the last suspended process.

kill – Killing a Process

- Occasionally a need may arise to stop the process adhoc
- This is done by sending the software, a termination signal to kill the process.

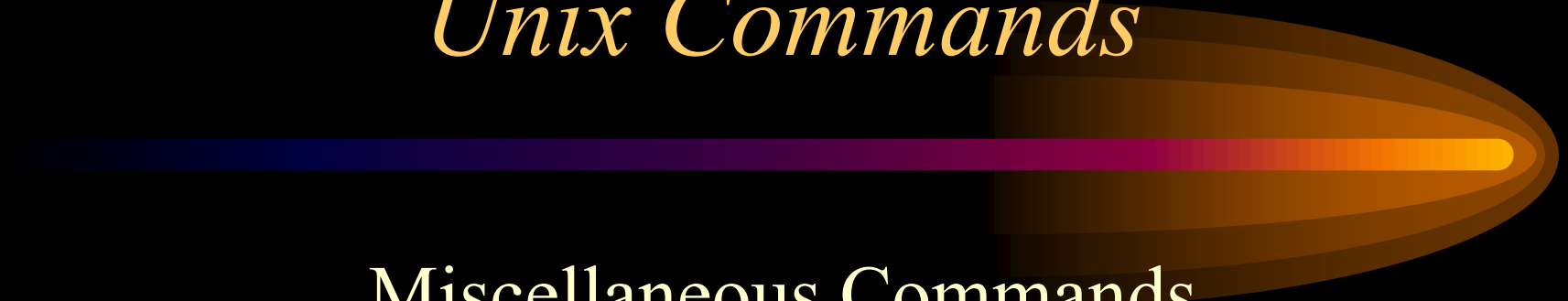
$ kill <pid>

- Some programs are designed to ignore the interrupts.
  - We can request the kill command to send a sure kill signal.
  - The sure kill signal is represented by including a minus nine option (-9) along with the kill command.

```
$ kill -9 1020
```

- Only the owner of the process or the super user can kill a process.

Unix Commands



Miscellaneous Commands

- **man**

Invoking Unix help.

Command Format:

man <command>

- **who**

Lists, who all have logged in to the system.

- **whoami**

Lists, your user name, i.e. name of the current user.

- **which**

Locating a program file including paths and aliases.
Command Format:
which <command>

- **whereis**

locate source, binary, and/or manual for program
Command Format:
which <command>

- **uname**

Display information about computer system.

- **rlogin**
  - Used for remote logging.
  - Command connects your terminal on the local host to the remote host.

Command Format:

$ rlogin <host name> -l <user name>

```
$ rlogin baileys -l snadmin
```

- **du**

Outputs the number of kilobyes used by each subdirectory. Useful if you have gone over quota and you want to find out which directory has the most files.

Example:

$ du -a

- **df**

Reports on the space left on the file system.

Command Format:

df [options] [special directory]

Example:

```
$ df -k ~ → Will print information about the
  home directory.
```

- **bdf**

Developed by Berkeley. It displays the free size in a more presentable manner.

- **gzip**

This also compresses a file, and is more efficient than compress.

For example:

```
$ gzip ae_socket.c
```

This will zip the file and place it in a file called ae_socket.c.gz

- **gunzip**

To unzip the file, use the gunzip command.

```
$ gunzip ae_socket.c.gz
```

- **Other compression commands:**
  - compress and uncompress
  - tar and untar

- Creating aliases
  - alias <new name>=<old name> → bourne/korne shell
  - alias <new name> <old name> → C shell

Unix Configuration

crontab and inittab

- **crontab:**
  - Allows you to schedule a task to be run at a predefined time.
  - Reads from a file.
  - email you http statistics, email your quota, clear logs every week, etc.

- **inittab:**
  - Script for the boot init process.
  - Located in /etc
  - inittab file supplies the script to the boot init daemon.

Unix Configuration

Important configuration files.

- **.cshrc**
  - Configuration file for the c shell.
  - Read each time a c shell is executed.
- **.login**
  - .login file is a file executed only the C-shell like shells. It is executed only for the very first shell started immediately after you login.
  - is started *after* the .cshrc has run.
- **.profile**
  - Configuration file for bourne shell and korn shell.
  - file is executed for each new korn/bourne shell you start
- **.exrc**
  - Configuration file for vi editor.
  - Is read each time vi is invoked

Unix Configuration

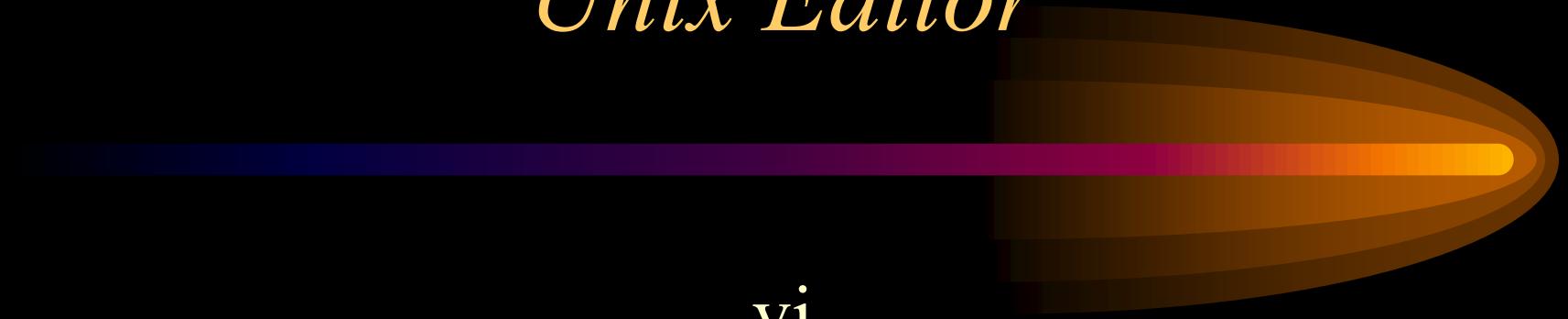
Shell Variables

- Two kinds
  - Shell
  - Environment

- **Environment**
  - Example of an environment variable is the PATH variable.
  - Other examples:
    - HOME (the path name of your home directory).
    - DISPLAY (the name of the computer screen to display X windows)
  - Generally in upper case.
  - To list all the environment variables type **env.**

- **Shell**
  - Example of of a shell variable is the history variable
  - Generally in lower case
  - Other examples:
    - cwd (your current working directory)
    - path (the directories the shell should search to find a command).
- **Differences between the two**
  - environment and shell variables can have the same name.
  - On changing the shell variables, the corresponding environment variables receive the same values. However vice-versa is not there.

Unix Editor



vi

vi

- Visual editor
- Works in three modes
  - Command
  - Insert

- Escape key is used to come back to command mode.

- Command mode
  - Cursor movements
  - Screen movements
  - Replace text
  - Delete text
  - Change to other modes

To invoke the **vi** editor:

```
$ vi   [filename]    <Enter>
```

- Cursor Movement

Moving around the screen	h, j, k, l, nh, nl, +
Moving within line	$, ^
Moving over words	w, W, b, B, e, E
Moving over paragraphs	{, }, n{, n}
Moving over sentences	(,)

- Screen Movement

Ctrl + d	Scroll down half window
Ctrl + u	Scroll up half window
Ctrl + b	Move to previous window
Ctrl + f	Move to next window

- Enter text

a, A	Append
i, I	Insert
o, O	Open line

- Delete text

x, nx	Delete character(s)
dw, ndw	Delete word(s)
dd, ndd	Delete line(s)

- Modifying text

r, R	Replace
cw, ncw	Change word(s)
cc, ncc, C, nC, c$	Change line(s)

- Cut and Paste

yy, nyy, yw, y$	Copy
p, P	Paste
u, U	Undo changes

- Insert mode

w, w <file>	Save
wq, q, q!, ZZ	Quit
!, sh	Execute commands

- Pattern matching

/pattern, ?pattern	Search
//, ??, n, N	Repeat search
1,$ s/pattern/text/g	Substitute

Shell Scripting

Scripting

- Similar to batch files in Unix.
- Frequently used to automate the work.
- Main features
  - I/O interaction.
  - Structured language construct.
  - Subroutines.
  - Variables
  - Arguments
  - Interrupt handling

Steps

- Open vi editor
- Write Unix commands.
- Save and exit vi.
- Execute the script:

```
$ sh script_name
```

A simple script file:

```
echo "execution starting"
ls -l
```

The variables in Shell are classified as:

User defined variables	Defined by the user for his/her use
Environmental variables	Defined by the shell for its own operations
Pre-defined variables	Reserved variables used by the shell and UNIX commands for specifying the exit status of commands, arguments to shell scripts, the formal parameters, etc.

- Command substitution
  - Way of placing values into variables where the output of a command is placed into a variable as its content.
  - `$ echo `date``
  - `$ cur_dir=`pwd``

- Evaluation of shell variables

$var	value of the variable 'var‘.
${var-value}	value of the variable, if defined, otherwise 'value‘.
${var=value}	value of the variable, if defined, otherwise 'value’ is assigned to the variable.
${var+string}	value of string, if var is defined, otherwise nothing.
${var?messg}	value of the variable, if defined, otherwise the shell exits after printing the 'messg‘ message.

- Computation on shell variables
  - expr
  - expr val1 op val2
  - expr $var1 op $var2
  - var3=`expr $var1 op $var2`
- Example:
  - **`$ var = `expr $a + $b``**

- Condition variables
  - if...then...elif...else…fi
- Testing the condition
  - **test**

eq, ne, gt, lt, le, ge	Operators on numeric variables
=, !=, z, n	Operators on string variables
s, f, d, w, r, x	Operators on files
a, o !	Logical comparison operators

In place of test we can use […]

Example:

```
if  test  "$HOME" = "/usr/home"
```

OR

```
if  [  "$HOME" = "/usr/home"  ]
```

- **Iterating constructs**
- for <variable> in <list>
  do
  commands
  done
- while command
  do
  commands
  done
- until command
  do
  commands
  done

Examples

(1)

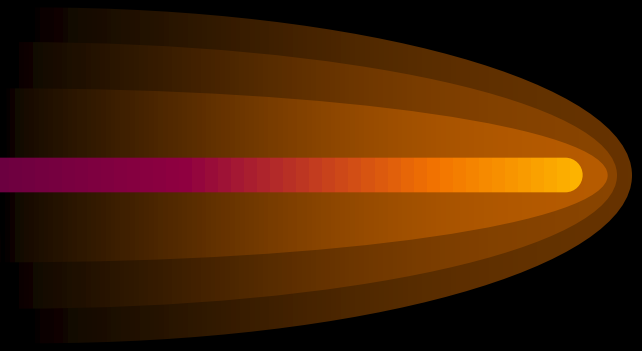
```
for i in 1 2 3 4 5 6 7 8 9 0
do
  echo $i
Done
```

2)

```
num ( =1
while  [  $num  -le  10  ]
do
  echo $num
  num=`expr $num +  1`
done
```

(3)

```
num=1
until  [  $num  -gt  10  ]
do
  echo $num
  num=`expr $num + 1`
done
```



Loop handling:

continue : to return control to the beginning of a loop

break : to exit a loop

- **The case....esac Construct**

```
case  word  in
pattern) command;;
pattern) command;;
-----
*) default;;
esac
```

- **Parameters to shell scripts**

$#	Parameter count
$*	All parameters
$0	The command name
$1, $2.....$n	Positional parameters
shift	Shifting parameters
$$	PID of current shell
set	Listing shell variables

- **shift**
- Shell cannot accept more than 9 parameters.
- shift command is used to shift the parameters one position to the left.
- The first parameter is overwritten by the second, the second by the third and so on.

- Example: Write a script which will accept different numbers and find their sum. The number of parameters can vary.

```
sum=0
while  [  $#  -gt  0  ]
do
  sum=`expr $sum +  $1`
  shift
done
echo sum is $sum

$ disp_test <1> <2> ……
```

- **set**
- Used to display all the shell variables and the values associated with them
- Executes a command that provided as a parameter, breaking the output of the command into words and store them in different variables, starting from $1 to $n.

- Example

```
$ set `date`
```

Bibliography

Unix Shell Programming
-Yeshwanth Kanetkar
Advanced Unix Programming
-Steven Prata
Unix Operating System
-Sumitaba Das