Module 2

Introduction to Shells

The Shell The Shell

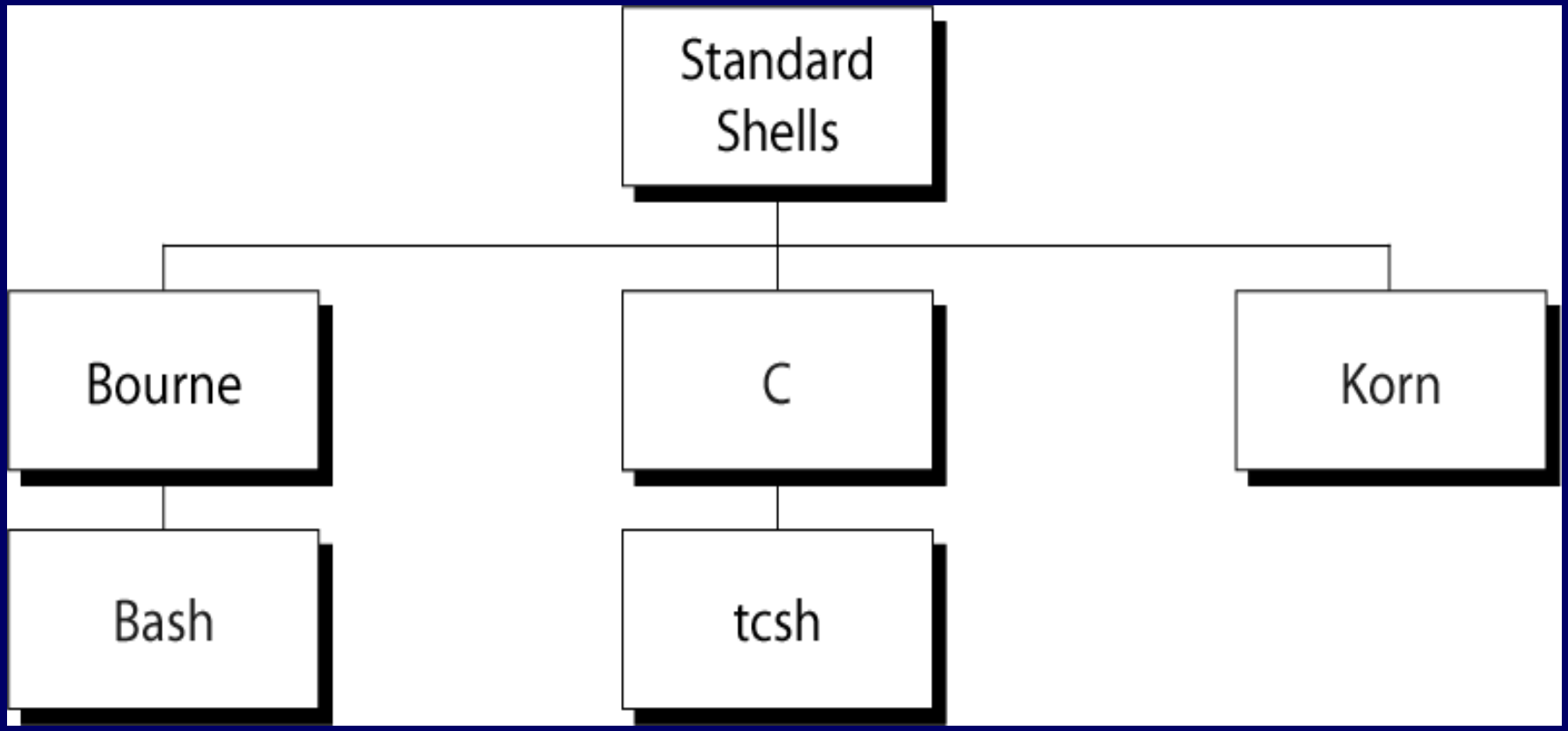
The shell is the part of UNIX that is most visible to the user.

It receives and interprets the commands entered by the user.

A shell script is a file that contains shell commands that perform a useful function.

It is also known as a shell program

Five Standard UNIX Shells



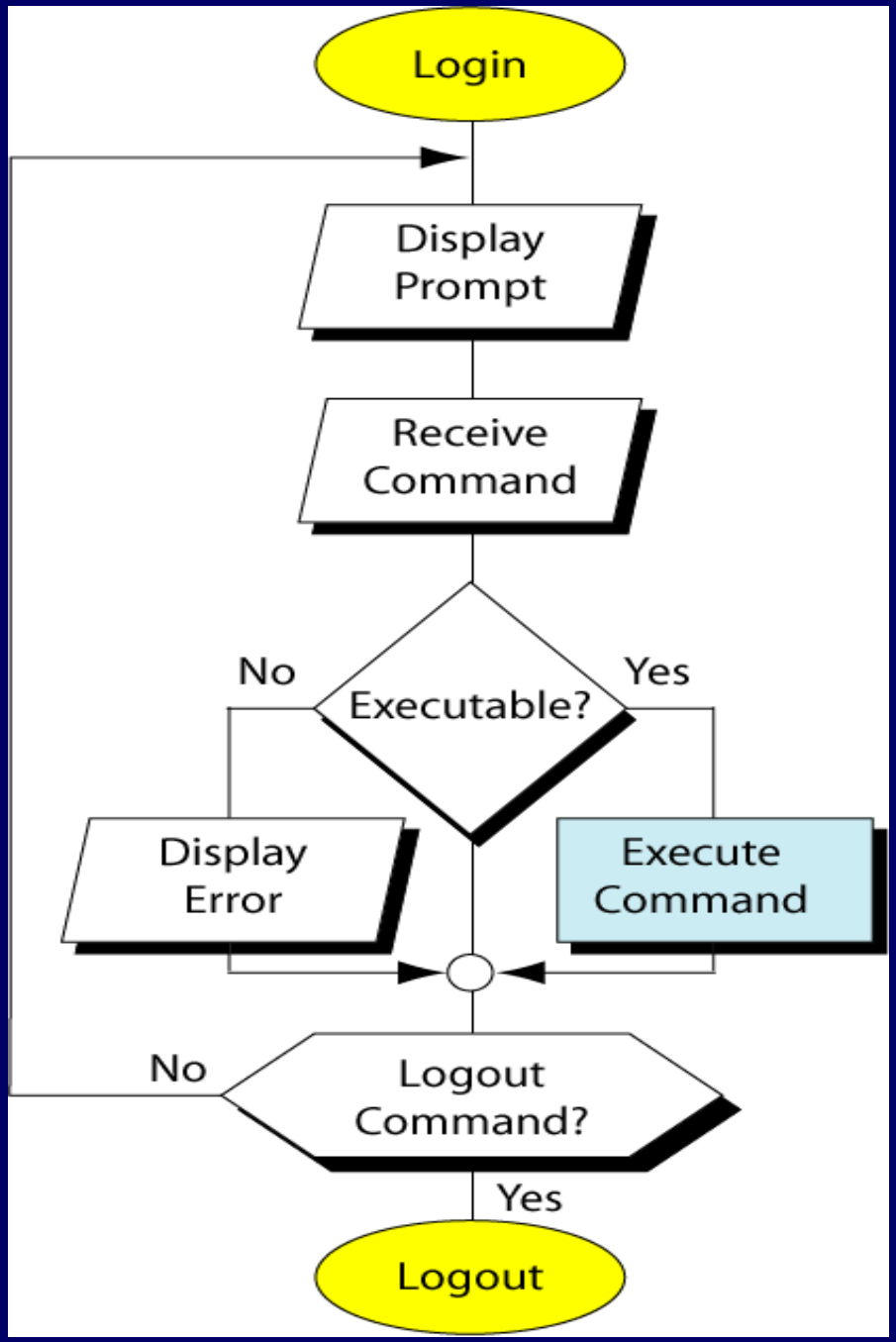
UNIX Session

When you log in, you are in one of five shells.

The system administrator determines which shell you start in.

You can switch to another shell by

- bash
- ksh
- csh



Login Shell Verification

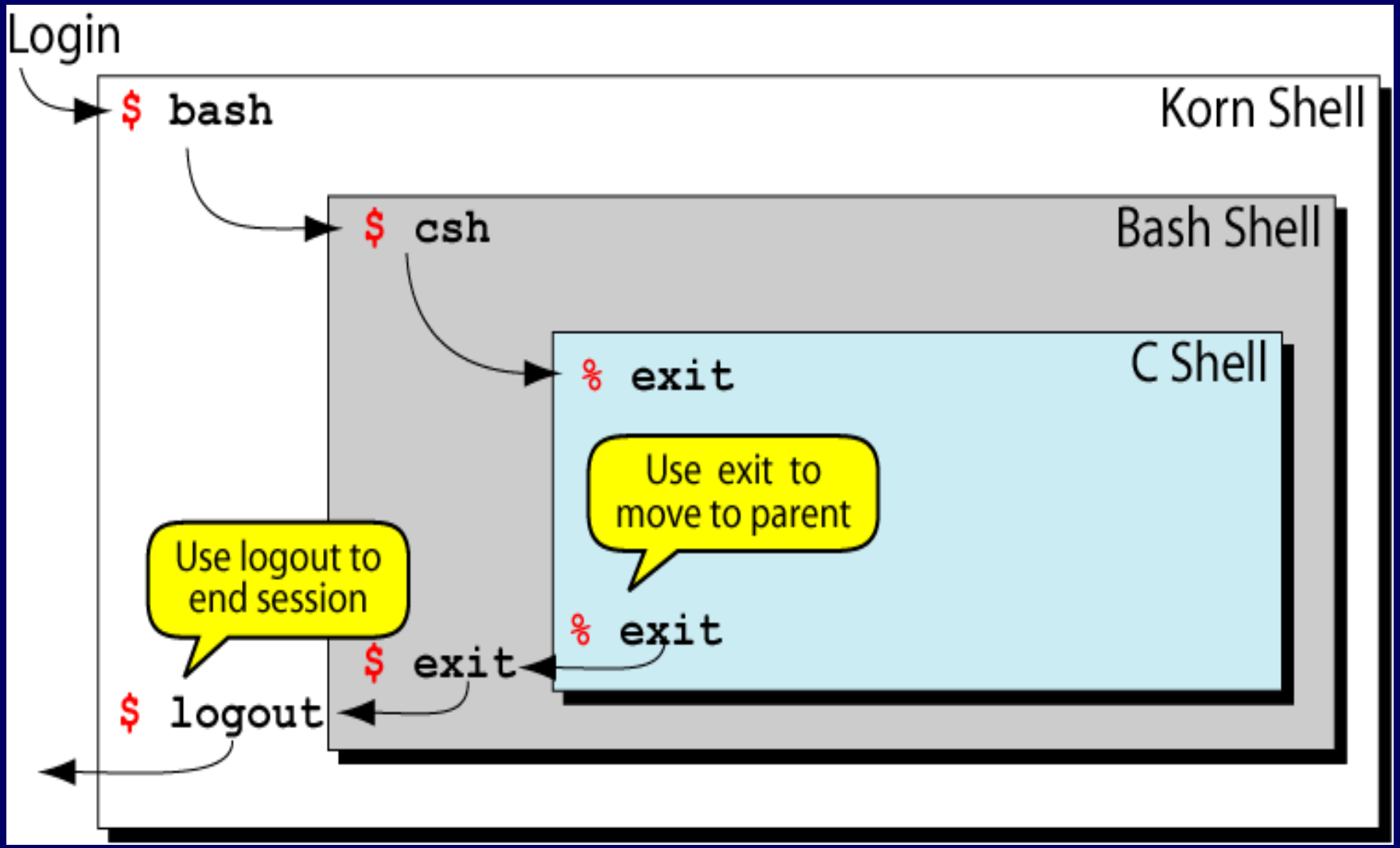
echo $SHELL identifies the path to your login shell

Current Shell Verification

echo $0 determined your current shell, but work only with the Korn and Bash shells; it does not work for the C shell.

This work for the C shell on r2d2 as well.

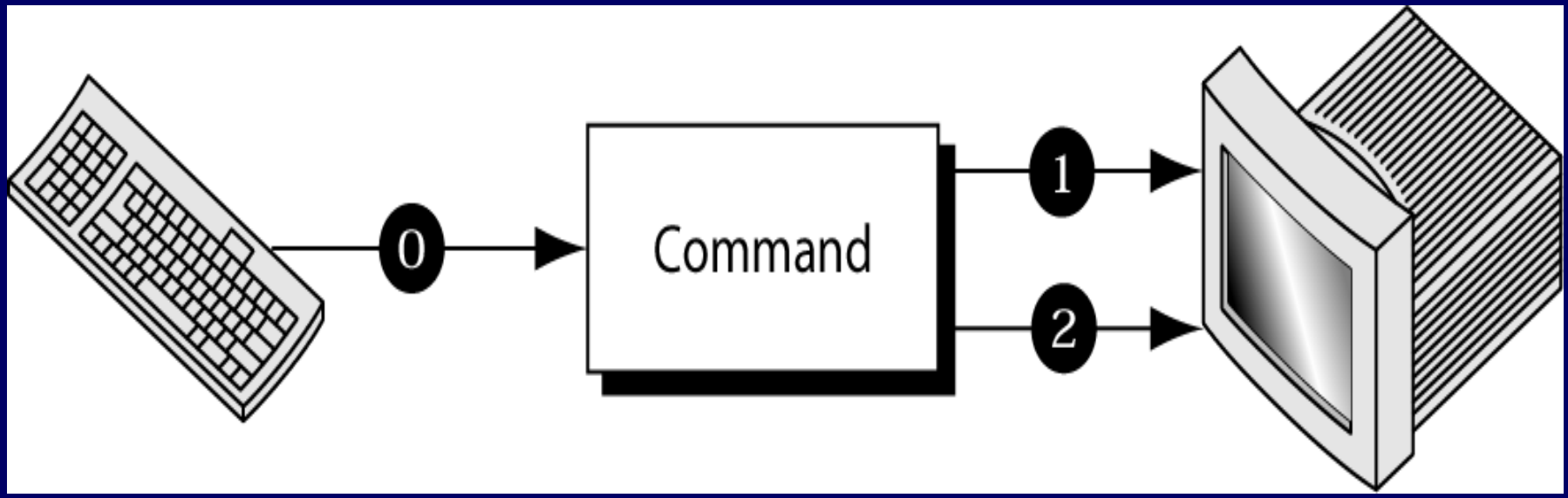
Shell Relationship



Standard Streams

There are three standard streams: standard input (0), standard output (1) and standard error (2).

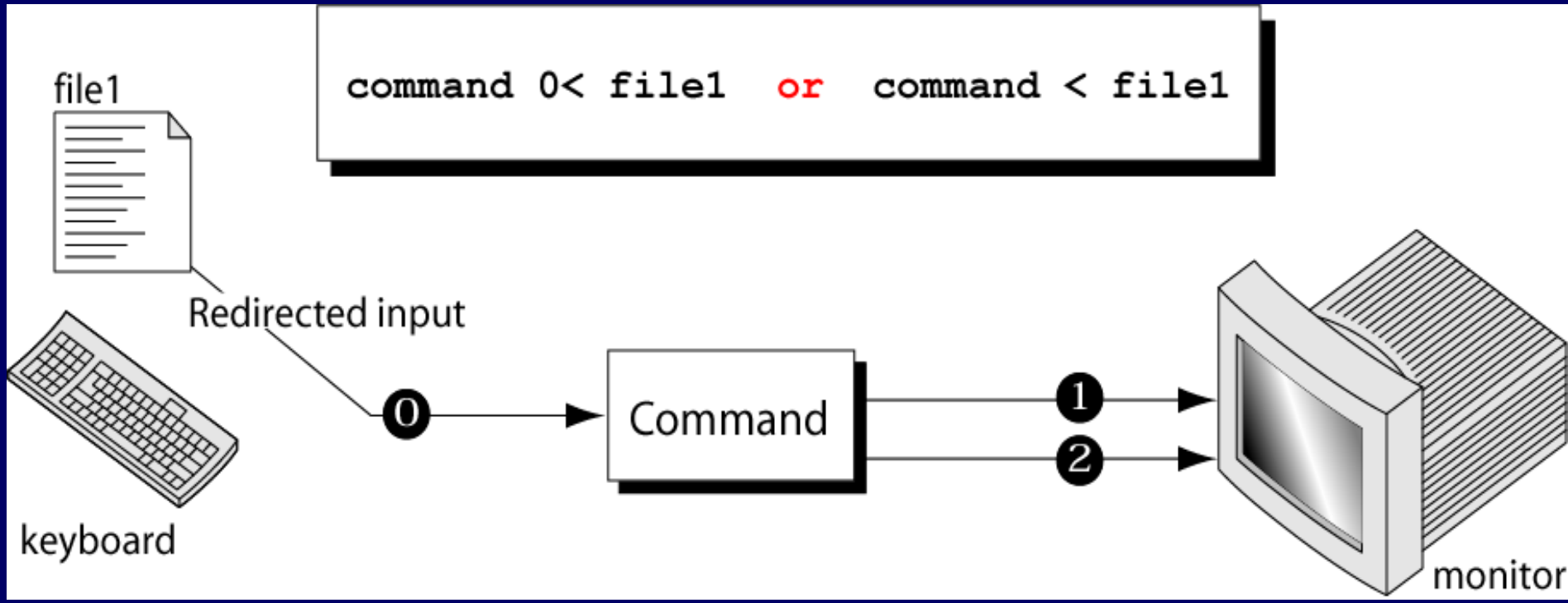
We can change the default assignments using redirection.



Redirecting Standard Input

We can redirect the standard input from the keyboard to any text file.

Thinks of it as an arrow pointing to a command, meaning that the command is to get its input from the designated file.



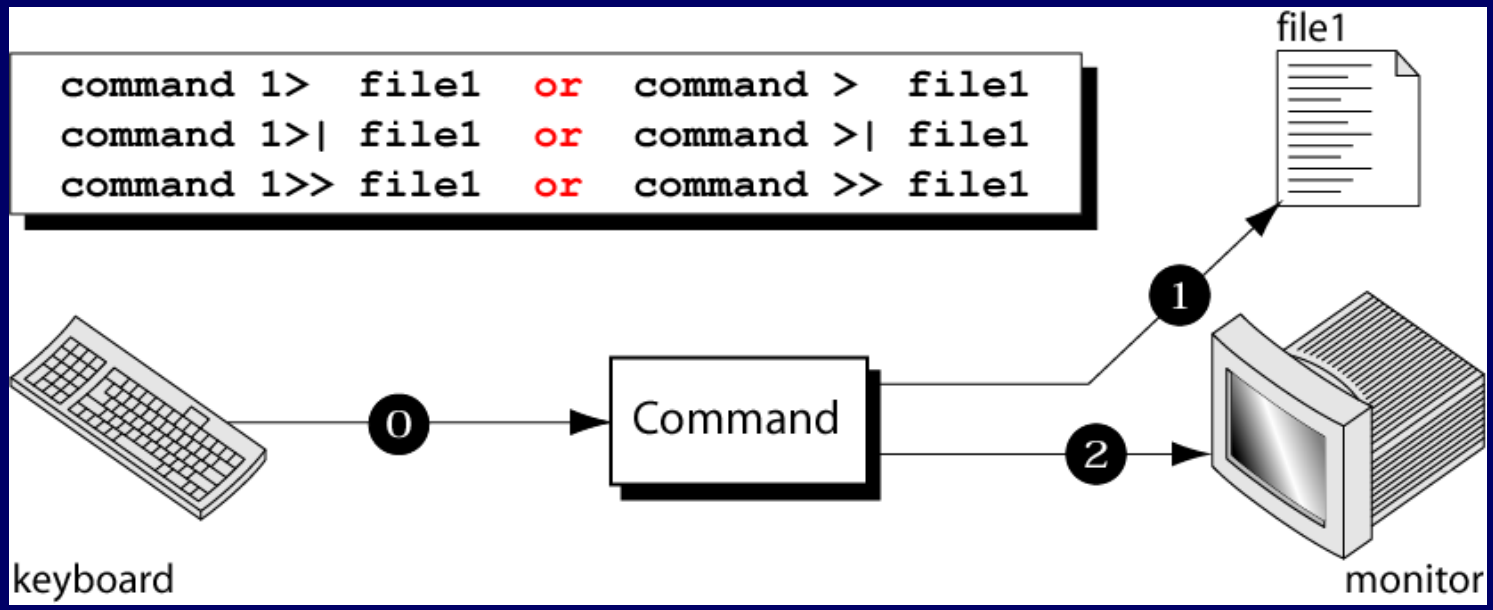
Redirecting Standard Input Example

```
Mail -s "Subject" to-address < Filename
```

Example: The mail program in Linux can help you send emails from the Terminal.
You can type the contents of the email using the standard device keyboard. But if you want to attach a File to email you can use the input re-direction operator in the above format.

Redirecting Standard Output

The command's output is copied into a file rather than displayed on the monitor.



Redirecting Standard Output example

```
ls -al > listings
```

What if the file “listings” do not exist?

Will it be appended?

Will it be over written?

Redirecting Standard Output (cont)

> If the file already exists, depending on a UNIX option it may create an error.

>| The file is emptied and then the new output is written (>! For C shell).

>> Appends the output to the file.

Redirecting Errors

We can do error redirection using

command 2> file

command 2 >| file to overwrite

command 2 >> file to append

If we want to output and errors to send to different files, we use

command 1> fileOut 2> fileErr

These commands do not work for the C shell.

Redirecting Errors example

- Create a file called “myfile1”
- Run the command
  ls –l myfile1 NoFile
- Note that “NoFile” is not available in the disk.
- ls –l myfile1 NoFile 1> output 2> error

Redirecting Errors (cont)

If we want both output and errors to be written to the same file, we use

command 1> file 2> &1

command 1>| file 2> &1

example “ls –l myfile1 NoFile 1> result 2> &1”

for the Korn and Bash shells

command >& file

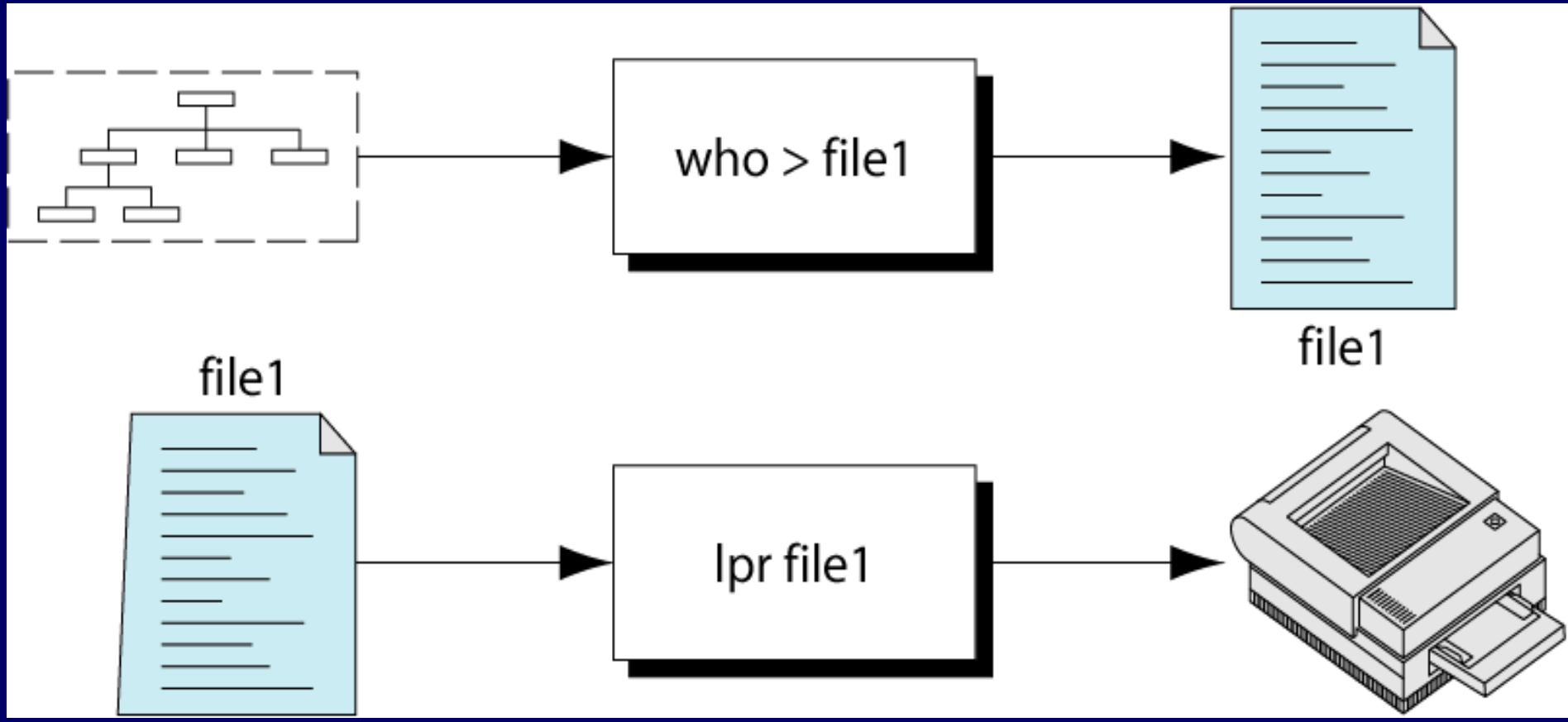
command >&! file

for the C shell

Pipes

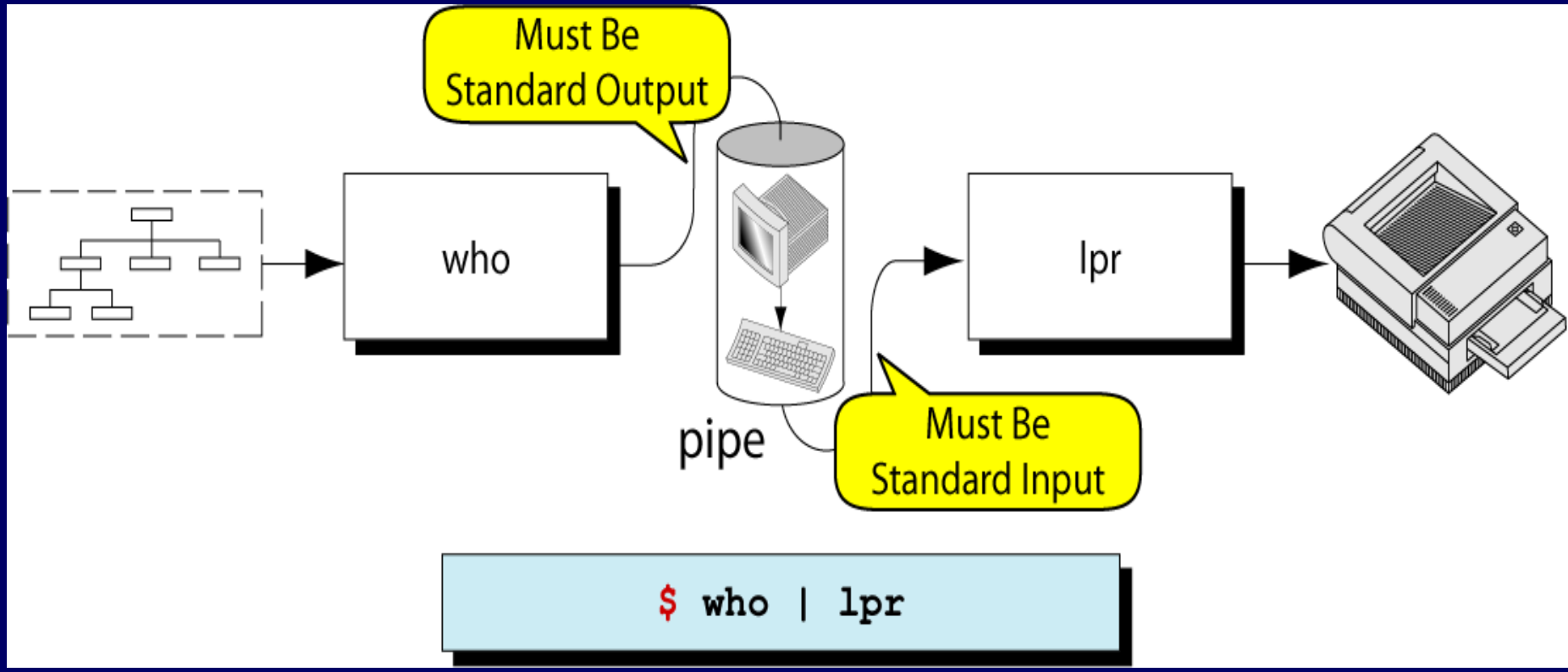
We often need to a series of commands to complete a task.

Example: to print a list of users logged into the system.



Pipes (cont)

Pipe is an operator that temporarily saves the output of one command in a buffer that is used at the same timeas an input of the next command (example “ls -l | more”)



The tee Command

The tee command copies standard input to standard output and at the same time copies it into one or more files.

To prevent files from being overwritten, we can use –a option, which tells tee to append the output to the existing files.

Instead of using the keyboard, we can feed the tee command through a pipe.

Command Execution

Sometimes we need to combine several commands.

There are four formats for combining commands into one line: sequenced, grouped, chained, and conditional.

Sequenced Commands

A sequence of commands can be entered on one line. Each command must be separated from its predecessor by semicolon.

There is no direct relationship between the commands.

command1; command2; command3

Grouped Commands

If we apply the same operation to the group, we can group commands.

Commands are grouped by placing them into parentheses.

Example:

echo “Month” > file; cal 10 2000 > file

(echo “Month” ; cal 10 2000) > file

Chained Commands

To chain the commands, we pipe them. The output of the first becomes the input of the second.

Conditional Commands

We can combine two or more commands using conditional relationships AND (&&) and OR (||).

If we AND two commands, the second is executed only if the first is successful.

If we OR two commands, the second is executed only of the first fails.

```
cp file1 file2 && echo “Copy successful”

cp file1 file2 || echo “Copy failed”
```

Command line Editing

There is a way to edit and execute previous commands.

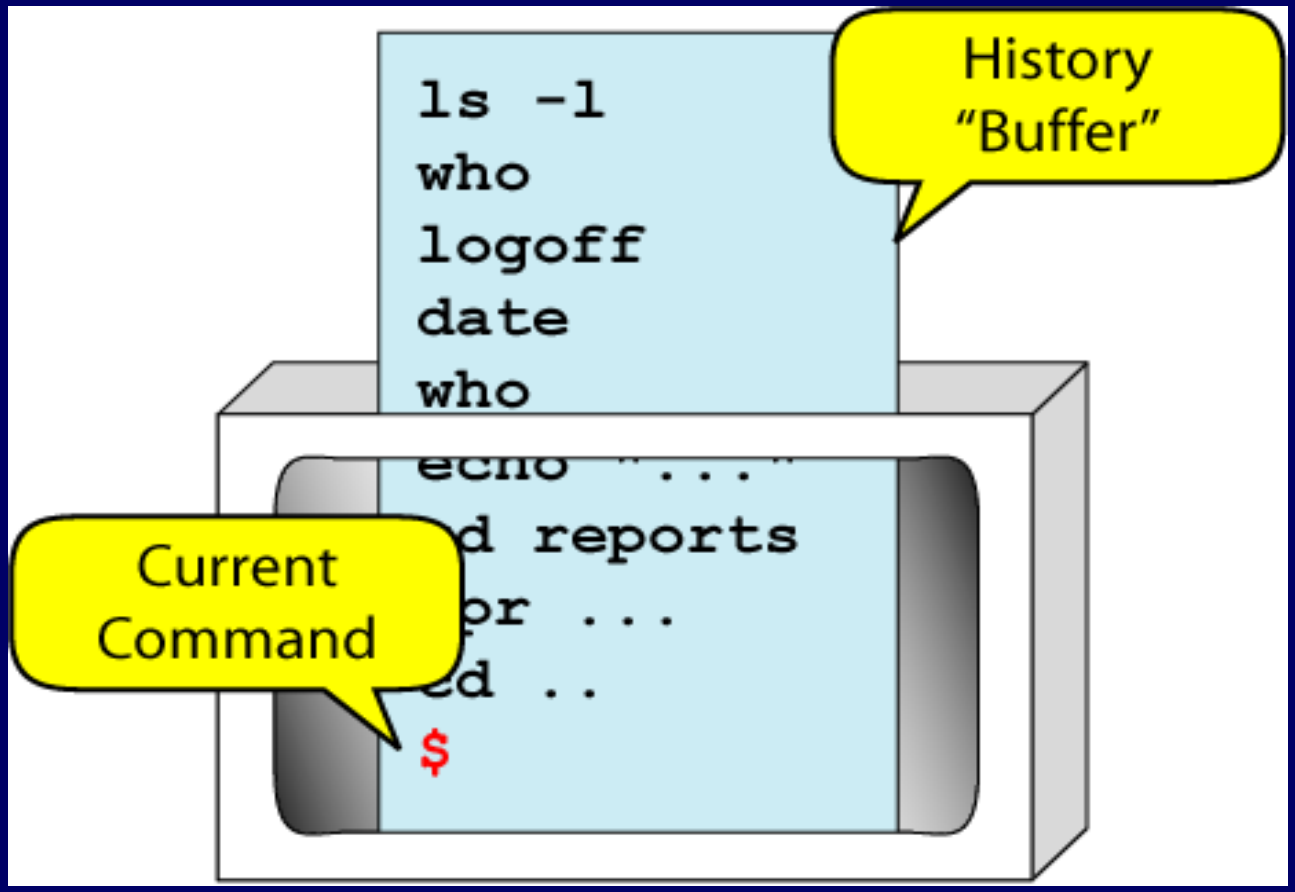
In the Korn and Bash shells, we can use the history file or command-line editing.

In the C shell, we can use only the history file.

As each command is entered on the command line, it is copied to a special file.

Command Line Editing Concept

With command-line editing we can edit the commands using vi or emacs without opening the file.
To set the editor we use set –o vi



vi Command Line Editor

vi command line editor opens in the insert mode.

This allows us to enter commands easily.

vi editor treats the history file as though it is always open.

To move to the command mode we must use Escape key.

We can use the cursor moving commands:

k or up arrow to move up the list to an older line

j or down arrow to move down the list.

Executing a Previous Line

1. Move to the command mode by keying Escape key.
2. Move up the list using the Move-up key.
3. When command has been located, key Return to execute it.

After the command has been executed, we are back at the bottom of the history file in the insert mode.

Edit and Execute a Previous Line

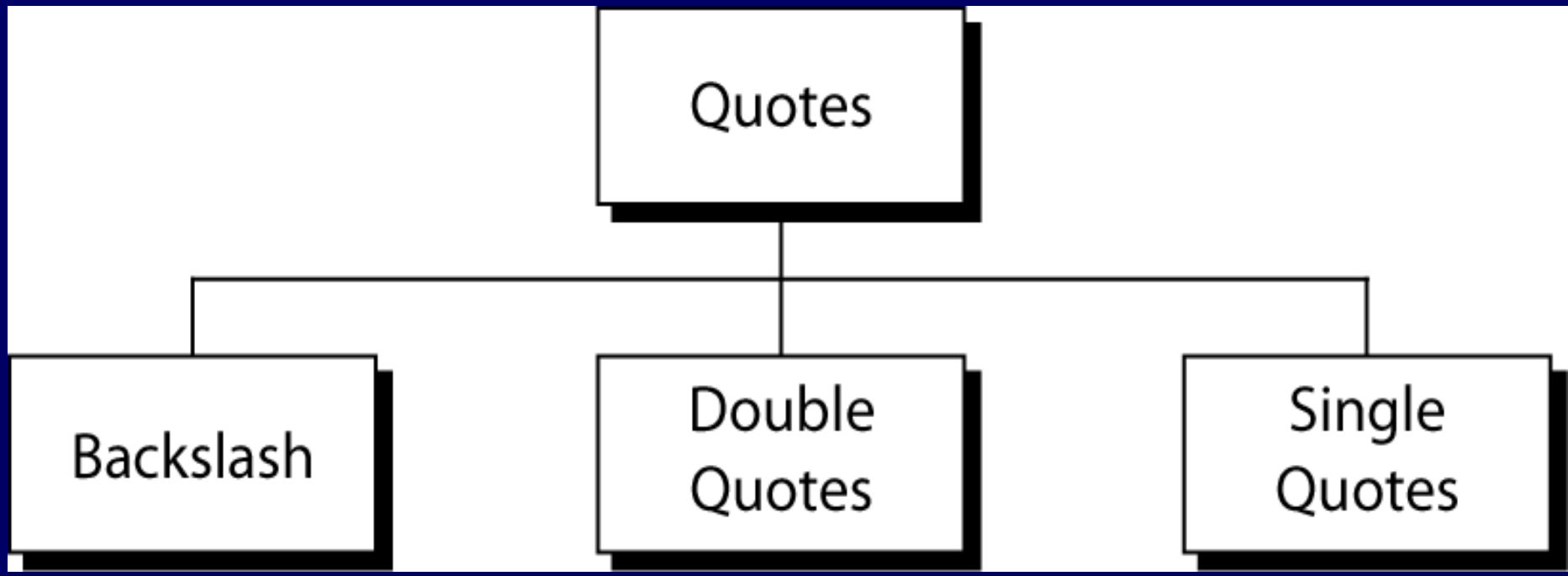
1. Move to the command mode by keying Escape key.
2. Move up the list using the Move-up key, recall the previous line.
3. Edit the line using vi editor rules.
4. When command has been located, key Return to execute it.

After the command has been executed, we are back at the bottom of the history file in the insert mode.

Quotes

Metacharacters are characters that have a special interpretation, for example the pipe |.

We need a way to tell the shell interpreter when we want to use them as text characters.



Backslash \

The backslash converts literal characters into special characters and special characters into literal characters.

Example: < is input redirection operator

\< is less than character

n is a letter

\n is new line character

This command displays special characters

echo \< \> \” \’ \\ \$

< > “ ‘ \ $

Double Quotes

When we need to change the meaning of several characters, we use double quotes.

```
echo  “< > ‘y’ ? &”
    < > ‘y’ ? &

echo  “Use quotes “inside” the quotes”
    Use quotes inside the quotes

echo  “Use quotes \“inside\” the quotes”
    Use quotes “inside” the quotes
```

Single Quotes

Single quotes operate like double quotes, but their effect is stronger.

Any enclosed metacharacters are treated as literal characters.

echo “Use quotes “inside” the quotes”

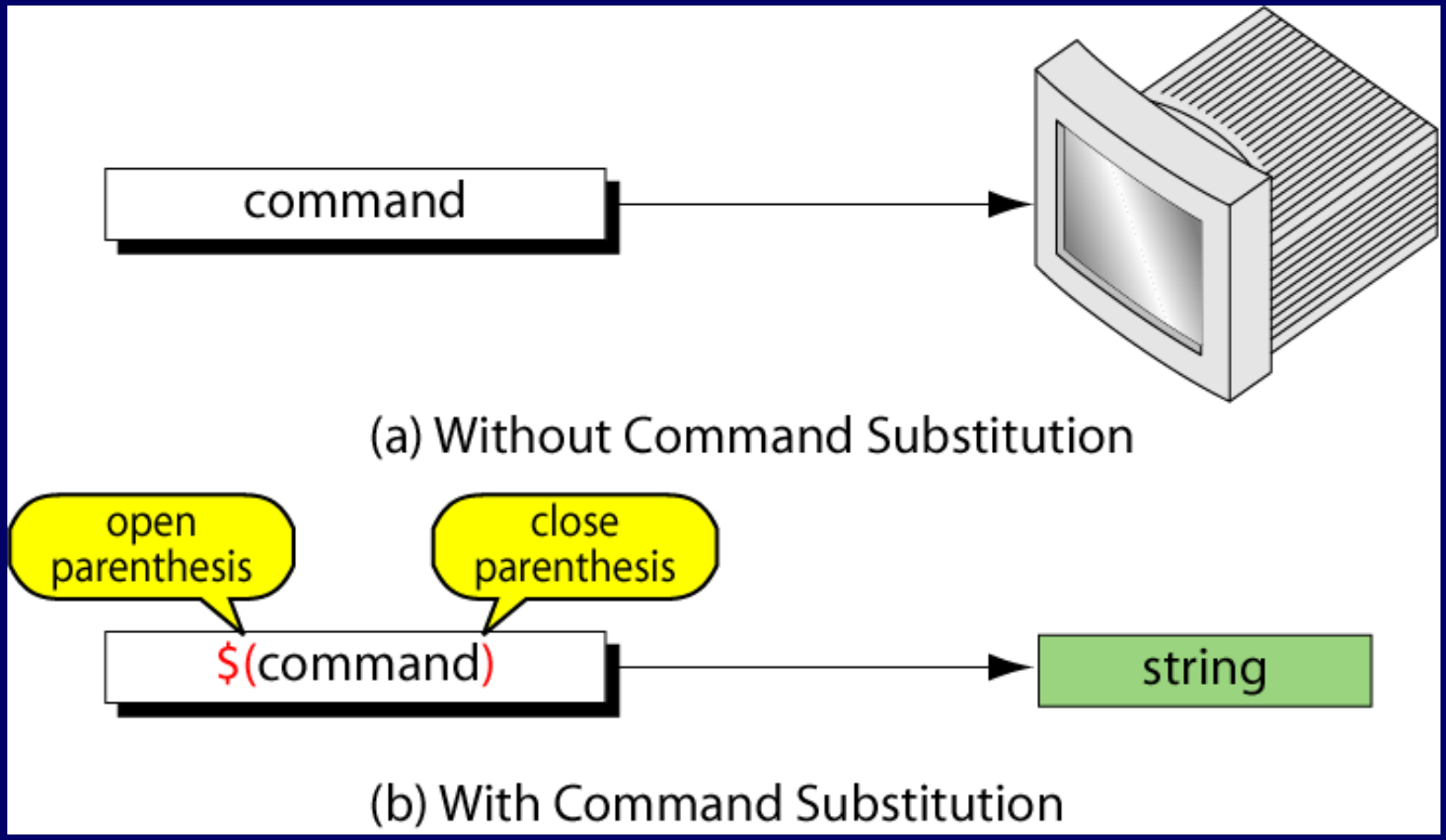
Use quotes inside the quotes

echo ‘Use quotes “inside” the quotes’

Use quotes “inside” the quotes

Command Substitution

Command substitution provides the capability to convert the result of a command to a string.



Using Command Substitution

```
echo The date and time are: date
     The date and time are: date


echo  The date and time are: $(date)
    The date and time are: Mon Sep 11 09:48:04 PDT 2000
          for Korn and Bash shells
echo  The date and time are: `date`
    The date and time are: Mon Sep 11 09:48:04 PDT 2000
          for C shell
```

Job Control

A job in UNIX is a command or set of commands entered on one command line.

UNIX is a multitasking operating system, therefore we can ran more than one job at a time.

UNIX defines two types of jobs: foreground and background.

Foreground Jobs

A foreground job is any job run under the active supervision of the user.

While it is running no other job may be started.

To start a foreground job, we simply enter a command and key Return.

All commands we have run so far has been run as foreground jobs.

To suspend a foreground job, key ctrl+z.

To resume it, use the foreground fg command.

To terminate (kill) a foreground job, use ctrl+c.

Background Jobs

When we know a job will take a long time, we may want to run it in the background.

Jobs run in the background free the keyboard and monitor so that we may use them for other tasks.

The foreground and background jobs share the keyboard and monitor.

Any messages sent to the monitor by the background job will be mingled with messages from the foreground job.

It recommended to redirect input and output for background jobs.

command& starts a background job

Background Jobs (cont)

stop %job_number suspends a background job.

bg %job_number restarts it.

kill %job_number terminates the background job.

fg %job_number moves it to foreground.

We can move a foreground job to background, we suspend it, and then use bg command, no job number is needed.

jobs Command

To list the current jobs and their status, we use the jobs command.

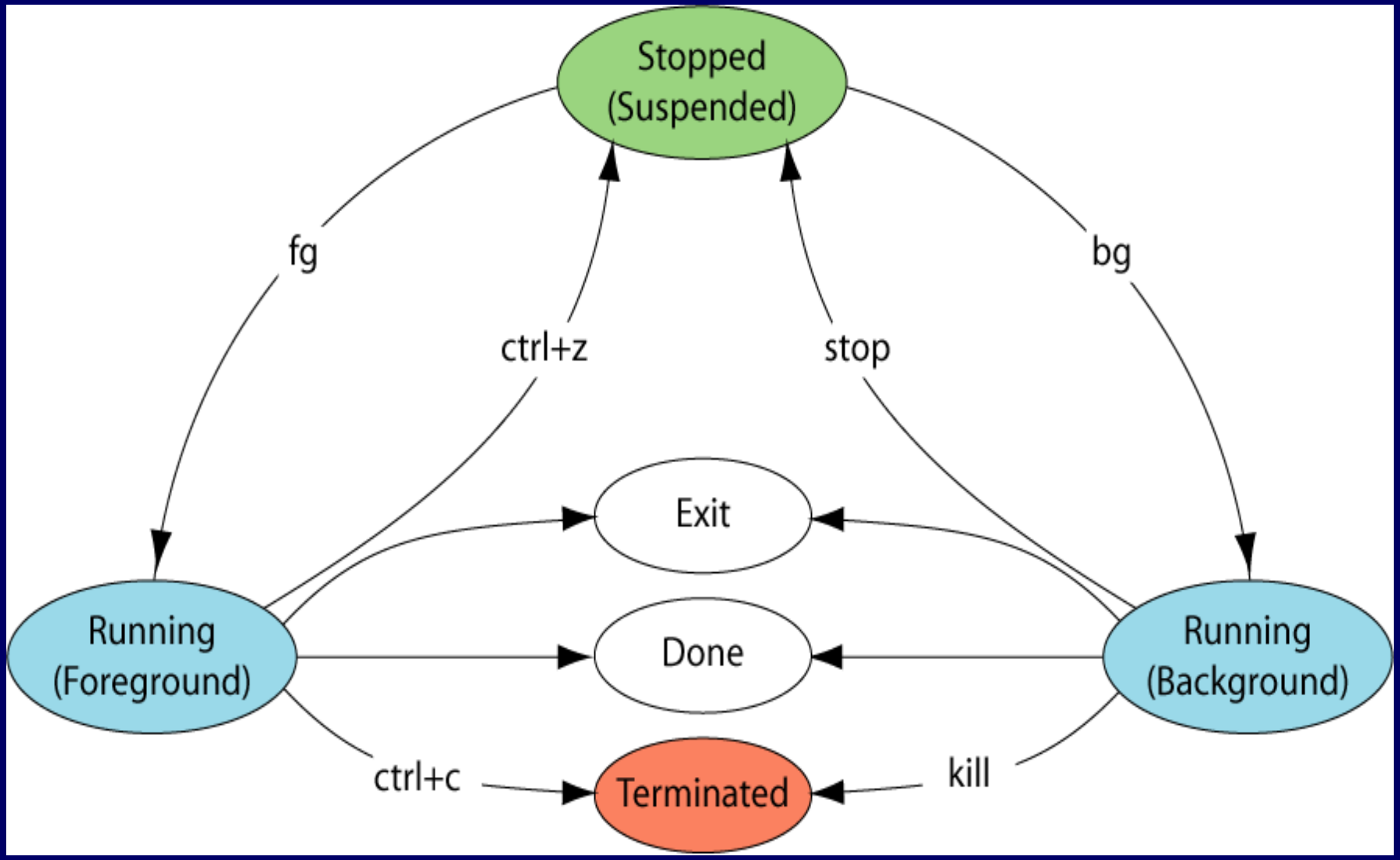
It shows the job number, currency, and status (running/stopped).

Jobs numbers are unique to a user session. They are not global.

UNIX assigns another identification, which is global, PID.

The ps command displays the current PIDs associated with the terminal.

Job States


Stopped
(Suspended)
fg
ctrl+z
stop
bg
Exit
Running
(Foreground)
Done
Running
(Background)
ctrl+c
Terminated
kill

Aliases in the Korn and Bash Shells

An alias provides a means of creating customized commands by assigning a name to a command.

alias name=command-definition

Examples:

alias dir=ls	Renaming a command
alias dir='ls –l'	Command with option
alias dir='ls –l | more'	Using multiple commands
alias lndir='dir –l | more'	Using an alias in a definition

Arguments are added after the command.

Sometimes it is ambiguous.

Aliases in the Korn and Bash Shells (cont)

To list all the aliases, we use alias command with no arguments.

To list a specific command, we use alias command with the name of the command.

Aliases are removed by using the unalias command.

unalias –a deletes all aliases.

It deletes even aliases defined by a system administrator.

Aliases in the C Shell

alias name definition

Example: alias dir “echo my directory list; ls –l | more”

C shell allows to control the positioning of arguments.

\!* is a designator for position of the only argument.

Example:

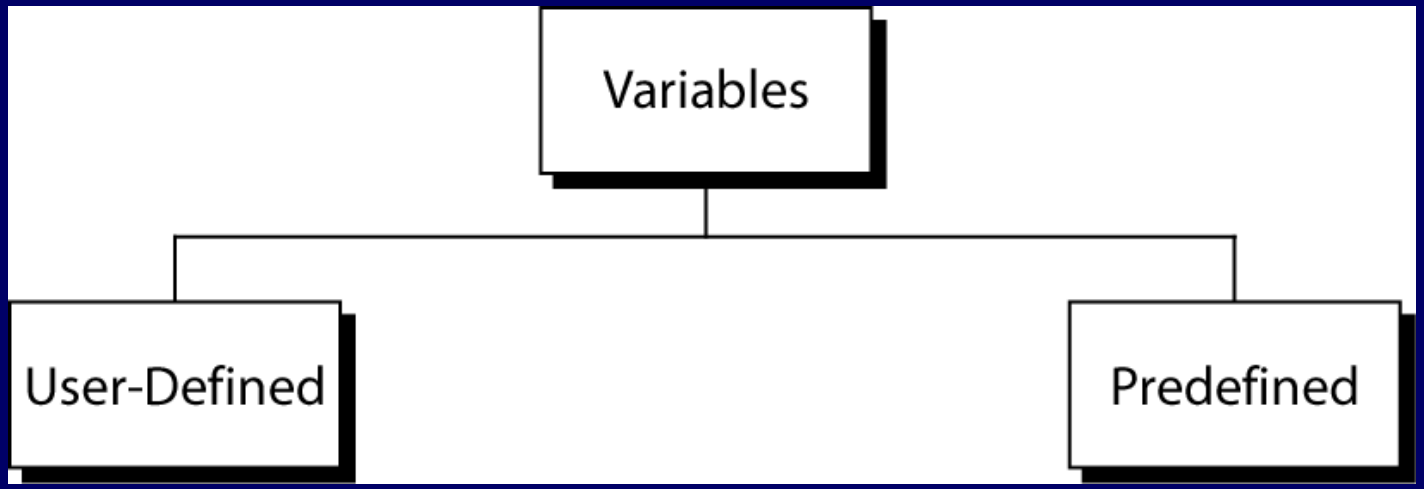
alias dir ‘ls –l \!* | more’

Listing and removing aliases are the same as for the Korn and Bash shells.

Variables

A variable is a location in memory where values can be stored.

Each variable has a name, which starts with an alphabetic or _ character followed by alphanumeric or _ characters.



Not defined in UNIX, defined by a user

Defined in UNIX, used to configure a shell environment

Storing Data in Variables

	Korn and Bash	C
Assignment	variable=value	set variable=value
Reference	$variable	$variable

```
x=23
echo $x
23
```

```
set x=23
echo $x
23
```

PATH

The PATH variable is used to search for a command directory.

The entries in PATH must be separated by colons.

PATH=/bin:/usr/bin::

The current working directory is listed last.

HOME

The HOME variable contains the PATH to your home directory.

When you use cd command with no arguments, the command uses the value of the HOME variable as the argument.

echo $HOME

/mnt/diska/staff/gilberg

CDPATH

The CDPATH variable contains a list of pathnames separated by colons.

:$HOME:/bin/usr/files

It starts with the current working directory, followed by pathname of the home directory and others.

The contents of CDPATH are used by the cd command to locate directories.

Primary Prompt

The primary prompt is set in the variable PS1 for the Korn and Bash shells and prompt for the C shell.

The shell uses the primary prompt when it expects a command.

The default is $ for the Korn and Bash shells and % for the C shell.

We can change the primary prompt by

PS1="KSH> " for the Korn and Bash shells

set prompt = 'CSH % ' for the C shell

SHELL

The SHELL variable holds the path to your login shell.

TERM variable

The TERM variable holds the description for the terminal you are using.

For the Korn and Bash shells,

TERM=vt100 sets the TERM variable

unset TERM unsets the TERM variable.

echo $TERM displays the value of the TERM variable

Shell/Environment Customization

UNIX allow us to customize the shells and the environment we use.

Temporary Customization

Temporary customization lasts only for the current session.

Permanent Customization

Permanent customization is achieved through startup and shutdown files by adding customization commands to them.

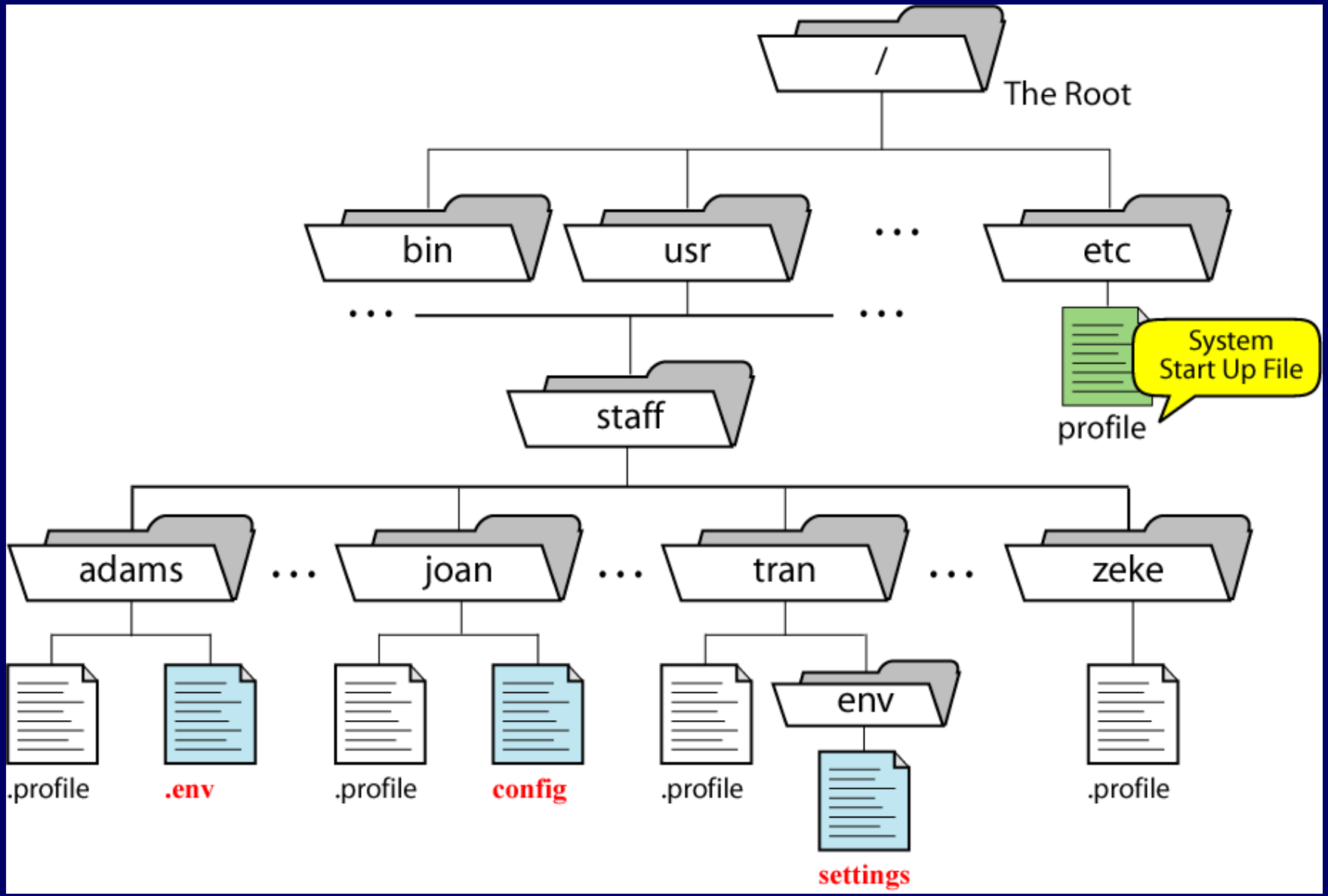
Korn Shell

System Profile File profile is stored under /etc directory and maintained by the system administrator. It contains general commands and variable settings that are applied to every user of the system at login time. It is read-only file.

Personal Profile File ~/.profile contains commands that are used to customize the startup shell. If you make changes to it, it is recommended to make a backup copy first.

Environmental File contains environmental variables, It does not have a predetermined name. Usually it is located at the home directory.

Korn's Environmental File



Bash Shell

/etc/profile is used for the system profile file.

For personal profile file, one of the three files is used:

~/.bash_profile **~/.bash_login** **~/.profile**

The environmental filename is stored under the BASH_ENV variable.

There is a logout file **~/.bash_logout**

C Shell

~/.login is the equivalent of user profile file

~/.cshrc is the environmental file

~/.logout is run when we log out of the C shell

Other system files

/etc/csh.cshrc

/etc/csh.login

/etc/csh.logout

Setting and Unsetting in C Shell

Predefined variables are divided into to categories: shell variables and environmental variables.

To set/unset a shell variable, we use set/unset command

set prompt = ‘CSH % ’

unset prompt

To set/unset an environmental variable, we use setenv/unsetenv command.

setenv HOME /mnt/diska/staff/gilberg

Note: there is no assignment operator.