25D Linux Foundation Course

02 – Working with the Linux Shell

Overview

- ❑ How the Linux shell works
- ❑ Using the shell prompt
- ❑ Getting help for Linux shell commands
- ❑ Working with environment variables
- ❑ Working with aliases
- ❑ Using redirection and piping

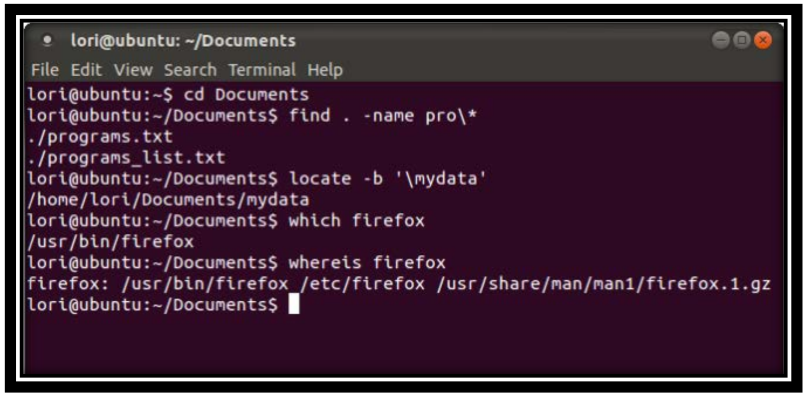
How the Linux Shell Works

- In a standard Linux installation both graphical and text based displays
  - Many offer up to 8 text based consoles and 1 graphic
  - Only one console can be displayed at a time
  - Each console has it's own settings
  - Switching between the console is done via CTR-ALT-F(1-9) buttons (functionality may vary between distributions)
- Command line (text-based) takes longer to learn but has benefits:
  - Speed and flexibility over graphic displays
  - Often graphic tools are not available (especially in remote)

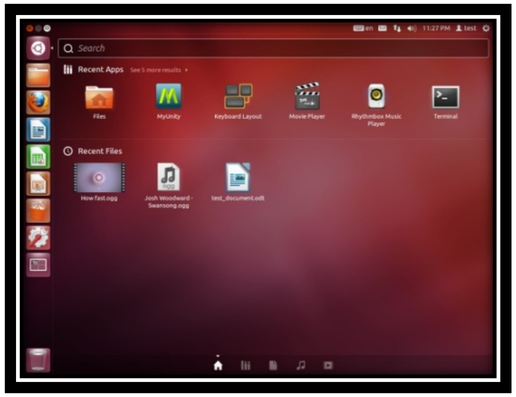
How the Linux Shell Works

❑ **Commands are given to the OS by the user via user interfaces**

- **Command-line interface (CLI)**
  - **Tool used by the user to communicate with the OS**
    - **Done by typing commands in the correct syntax at the command prompt**
- **Graphical user interface (GUI)**
  - **Configuration that allows for interaction via a mouse to graphics and icons instead of commands at the command prompt**



```
lori@ubuntu: ~/Documents
File Edit View Search Terminal Help
lori@ubuntu:~$ cd Documents
lori@ubuntu:~/Documents$ find . -name pro\*
./programs.txt
./programs_list.txt
lori@ubuntu:~/Documents$ locate -b '\mydata'
/home/lori/Documents/mydata
lori@ubuntu:~/Documents$ which firefox
/usr/bin/firefox
lori@ubuntu:~/Documents$ whereis firefox
firefox: /usr/bin/firefox /etc/firefox /usr/share/man/man1/firefox.1.gz
lori@ubuntu:~/Documents$ 
```



How the Linux Shell Works

❑ Linux shells

– Are command line interpreters that allow a user to input a command via a keyboard and send it to the kernel

- sh (Bourne Shell)
- bash (Bourne-Again Shell)
- csh (C Shell)
- tcsh
- zsh (Z Shell)

– Multiple command line sessions can be run at a time in either interface type (CLI or GUI) (CTRL-ALT-F*x*)

Exercise 2-1: Working with Linux Shells

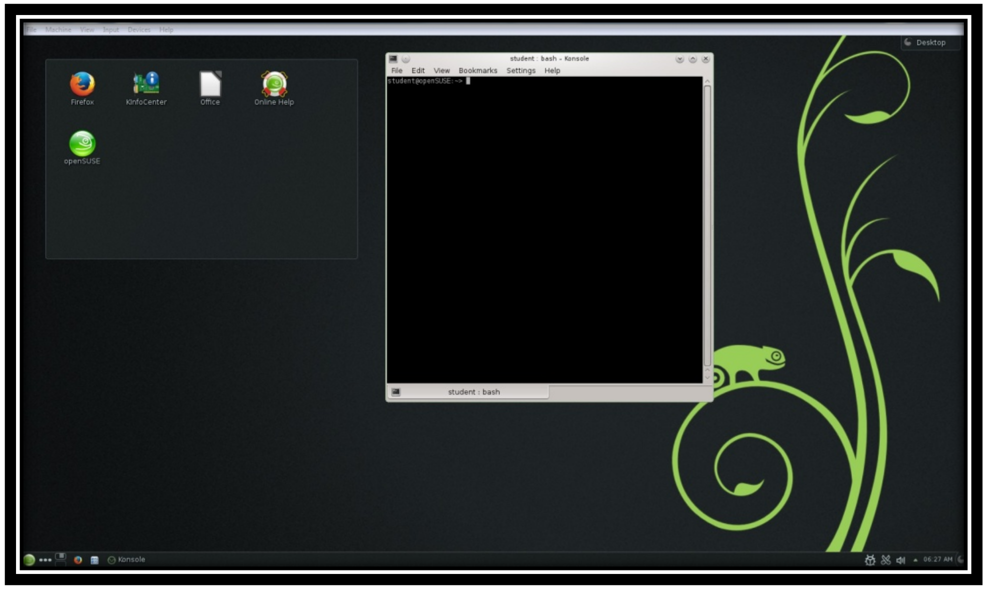
Please open your Practical Exercise book to Exercise 2-1.

Time to Complete: 5 Minutes

Shell Configuration Files

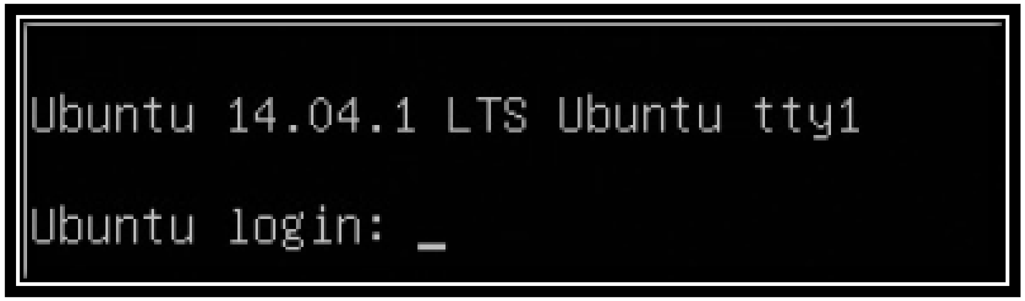
- **Login Shell vs. Non-Login Shell**
  - **Login Shell is the text based CLI the OS boots to**
  - **Non-Login Shell is the graphic version of the OS where you can open terminal windows within the desktop**
- **The shell you are using determines the configuration files run during start up**
  - **Configuration files are text-based shell scripts**
  - **Contain specific commands to run**
  - **Commands are run in order they appear in the file**
  - **The configuration file customizes the shell environment**

Shell Configuration Files



```
student@openSUSE:~>
```

Non-Login Shell



```
Ubuntu 14.04.1 LTS Ubuntu tty1

Ubuntu login: _
```

Login Shell

Shell Configuration Files

Bash Configuration File	Type of Shell	Function
/etc/bashrc or /etc/bash.bashrc	Non-login shells	Contains shell system-wide functions and aliases
~/.bashrc	Non-login shells (Although login shells on many distributions use this file as well. It is frequently called from one of the configuration files listed next.)	Stores user-specific functions and aliases
/etc/profile and the files in /etc/profile.d	Login shells	Contains system-wide shell environment configuration parameters
~/.bash_profile	Login shells	Stores user-specific shell preferences
~/.bash_login	Login shells	Stores user-specific shell preferences
~/.profile	Login shells	Stores user-specific shell preferences
~/.bash_logout	Login shells	Stores user-specific shell preferences

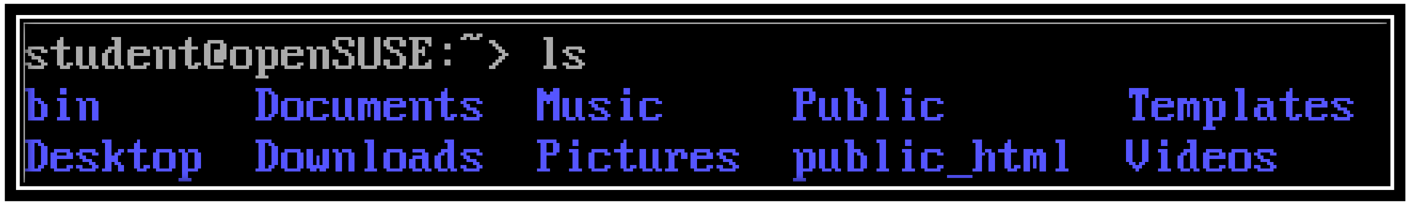
Shell Configuration Files

- **Non-Login Shell**
  - **Bash shell runs /etc/bashrc for system-wide functions and aliases first**
  - **Bash will then run the ~/.bashrc from the user's home directory for user specific customizations**
- **Login Shell**
  - **Bash will run the /etc/profile and applies configurations from that file**
  - **Depending on the distribution in use one of the following files is run next**

• **~/.bash_profile**	**~/.profile**
• **~/.bash_login**	**~/.bash_logout**

Entering Commands at the Shell Prompt

- Running a program or command from the shell prompt is easy
  - Type the command, script filename, or program filename at the prompt and press ENTER

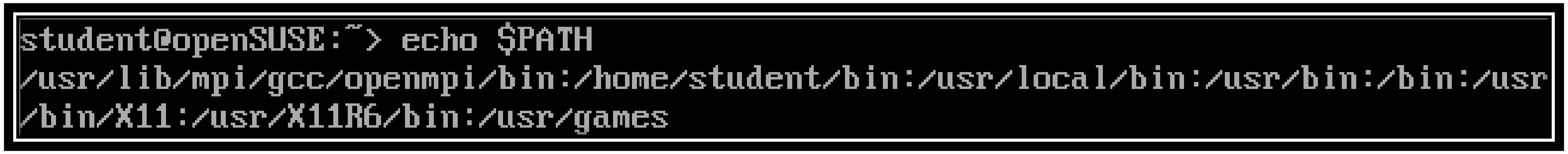


```
student@openSUSE:~> ls
bin       Documents  Music     Public       Templates
Desktop   Downloads  Pictures  public_html  Videos
```

  - In the above example the ls or list command (similar to DIR in Windows) has been entered
  - The files and directories within the current directory have been displayed

Entering Commands at the Shell Prompt

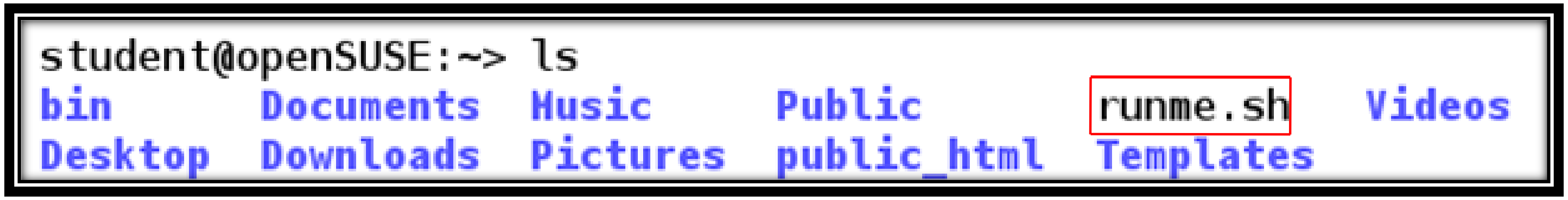
- Linux handles the path to the executable you want to run differently than other operating systems
  - employs a PATH environment variable
  - does not check the current directory
  - only searches for the file being run within the directories in the current users PATH



```
student@openSUSE:~> echo $PATH
/usr/lib/mpi/gcc/openmpi/bin:/home/student/bin:/usr/local/bin:/usr/bin:/bin:/usr
/bin/X11:/usr/X11R6/bin:/usr/games
```

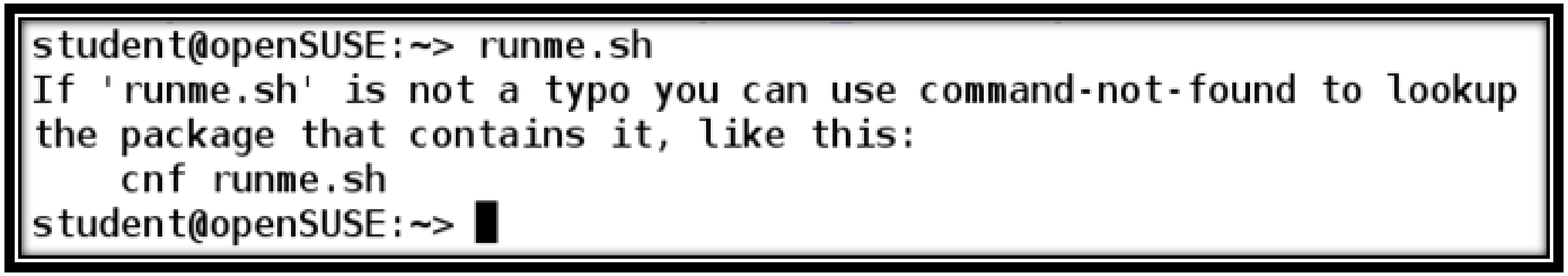
Entering Commands at the Shell Prompt

- In this example, an executable script file named runme.sh is located in the home directory of the student user:



```
student@openSUSE:~> ls
bin        Documents   Music       Public        runme.sh    Videos
Desktop    Downloads   Pictures    public_html   Templates
```

- When runme.sh is entered at the shell prompt, the shell cant find the file:



```
student@openSUSE:~> runme.sh
If 'runme.sh' is not a typo you can use command-not-found to lookup
the package that contains it, like this:
    cnf runme.sh
student@openSUSE:~> █
```

- The shell can’t find the file because student’s home directory is not listed within the PATH variable

Entering Commands at the Shell Prompt

- **There are ways to deal with issues seen in the last example:**
  - **You can enter the full path name: /home/student/runme.sh at the shell prompt to execute the file**
  - **You can switch to the directory where the executable file resides and add the "./" to the beginning of the command**
    - **First verify the you are in the current directory (/home/student in this example)**
    - **Once verified enter ./runme.sh at the shell prompt**
    - **The "./" characters specify the current directory, by using them you are telling the shell to look in that current directory**
  - **You could also add the directory of where the executable resides to the PATH environmental variable (covered later)**

Entering Commands at the Shell Prompt

- Be aware that that Linux filenames and directory names are case sensitive!
  - This means Linux commands are also case sensitive
- If the executable file you are trying to run is named runme.sh you must enter runme.sh not:
  - Runme.sh
  - RUNME.SH
  - Runme.SH
- The shell will interpret each of the above names as different files
  - This goes for directory names as well (student, Student and STUDENT for example)

Entering Commands at the Shell Prompt

- You can also use the “exec” command to run a program
  - For instance; if you had a Linux application named myapp you could execute it by entering “exec myapp” in the shell prompt (assuming the path to myapp is included in your PATH environmental variable)
  - The “exec” command is usually not used, the prior methods discussed in past slides are preferred
  - Using “exec” does have a useful feature over the other methods:
    - When not using the “exec” command the process created runs alongside the shell
    - When using “exec” the process created actually replaces the shell process from which it was launched

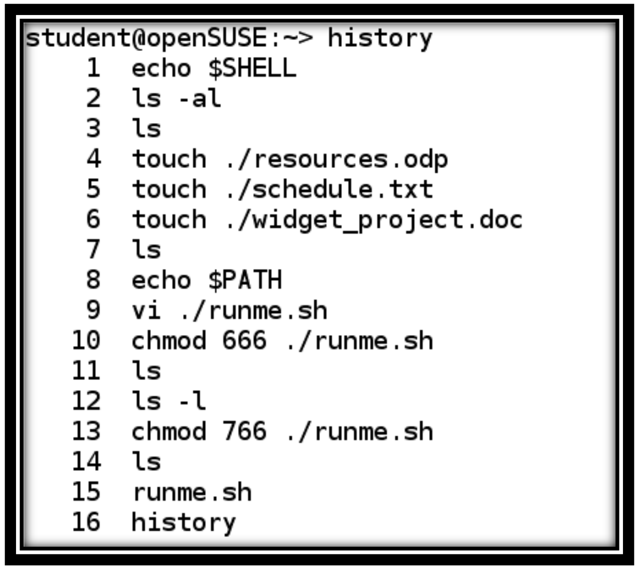
Exercise 2-2: Using Linux Commands

Please open your Practical Exercise book to Exercise 2-2.

Time to Complete: 5 Minutes

Using Command History

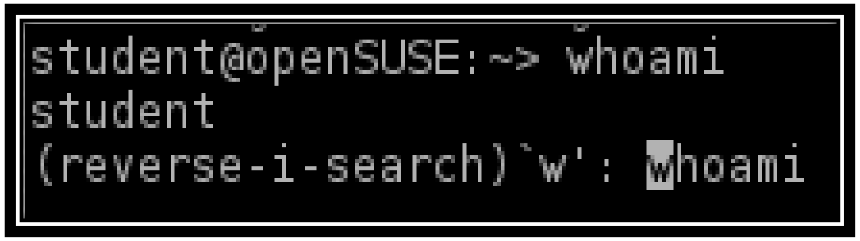
- The bash shell supports command history
  - Every time a command is entered at the shell prompt, that command is saved in the ~/.bash_history file in your home directory
  - The ~/.bash_history is a simple hidden text file, old commands are listed at the top and new commands at the bottom
  - The contents of the file can be displayed by entering "history" at the shell prompt:



```
student@openSUSE:~> history
    1  echo $SHELL
    2  ls -al
    3  ls
    4  touch ./resources.odp
    5  touch ./schedule.txt
    6  touch ./widget_project.doc
    7  ls
    8  echo $PATH
    9  vi ./runme.sh
   10  chmod 666 ./runme.sh
   11  ls
   12  ls -l
   13  chmod 766 ./runme.sh
   14  ls
   15  runme.sh
   16  history
```

Using Command History

- **One great thing about the history file is you can press the Up arrow and bash will read the file and display the last command**
  - **Pressing the Up arrow repeatedly will allow you to scroll through the last used commands**
    - **A great feature especially if very long, complex commands need to be entered again or corrected and entered again**
- **"CTRL-R" can also be used along with part of a command to find a past used command**
  - **Bash will search through the command history and match the most recent matching command**



```
student@openSUSE:~> whoami
student
(reverse-i-search)`w': whoami
```

Using Command History

- **Entries in the history file can be managed via the below environmental variables:**
  - **HISTSIZE or HISTFILESIZE- Configures the size of the history file which by default is usually 1,000 entries**
  - **HISTCONTROL- Controls how past command history is stored via four values:**
    - **ignoredups- tells the shell to ignore commands in the history list that are duplicates**
    - **ignorespace- tells the shell to ignore commands in the history list that start with spaces**
    - **ignoreboth- specifies ignoredups and ignorespace**
    - **erasedups- deletes/removes duplicates from the history file**

Exercise 2-3: Using Command History

Please open your Practical Exercise book to Exercise 2-3.

Time to Complete: 5 Minutes

Using Command Completion

- Extremely helpful when initially entering very long filenames
- Press TAB key while typing commands at shell prompt
- Bash shell guesses what you want to type in and automatically completes it for you
- Example:
  – We need to extract a tarball file to install an application contained within via the command: tar –zxvf /tmp/vmware-linux-tools.tar.gz
  – We could enter the first part of the command, tar –zxvf /tmp/vmw and hit the TAB key until the bash displays the desired command (tar –zxvf /tmp/vmware-linux-tools.tar.gz)

Exercise 2-4: Using Command Completion

Please open your Practical Exercise book to Exercise 2-4.

Time to Complete: 5 Minutes

Getting Help for Linux Shell Commands

- ❑ **Documentation and help information for Linux are abundantly available**
  - **In some cases depending on the vendor or distribution, the documentation may be minimal**
  - **With all of the different applications, distributions, etc. it is a difficult task for Linux vendors to include everything**
  - **Since Linux is an open source OS some programmers write their own documentation and the quality varies**
  - **Most documentation available now is not print based**
- ❑ **Getting help in Linux is easy using:**
  - **man pages**
  - **info**

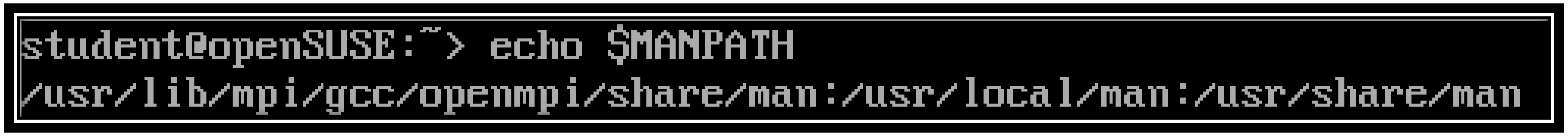
Getting Help for Linux Shell Commands

- ❑ **man pages**
  - – **Short for manual pages**
  - – **Contain documentation about the OS as well as installed applications**
  - – **Impossible to know all commands and syntax**
    - **Some commands are rarely used**
  - – **Use man to display the appropriate man page for a command or utility**
  - – **Whenever a package or application is installed the man pages containing the documentation for the software is also installed**

Getting Help for Linux Shell Commands

❑ man Directory Structure

- Actual pages displayed by man are in several directories beneath the MANPATH environmental variable
- Value of MANPATH can be verified by entering “echo $MANPATH” at a shell prompt
- This command will list the directories containing man pages like in this example:

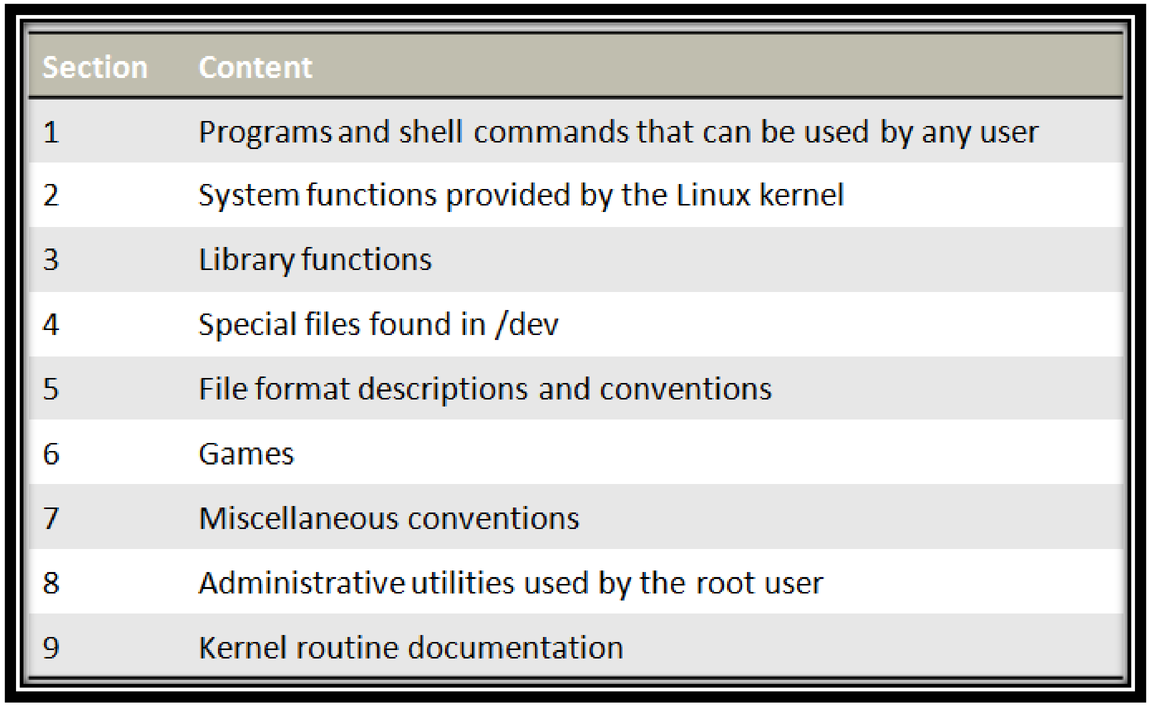


```
student@openSUSE:~> echo $MANPATH
/usr/lib/mpi/gcc/openmpi/share/man:/usr/local/man:/usr/share/man
```

- The man utility can be implemented in different ways on different Linux distributions
  - In Fedora the MANPATH environmental variable is not used by default and the MANPATH_MAP command is used

Getting Help for Linux Shell Commands

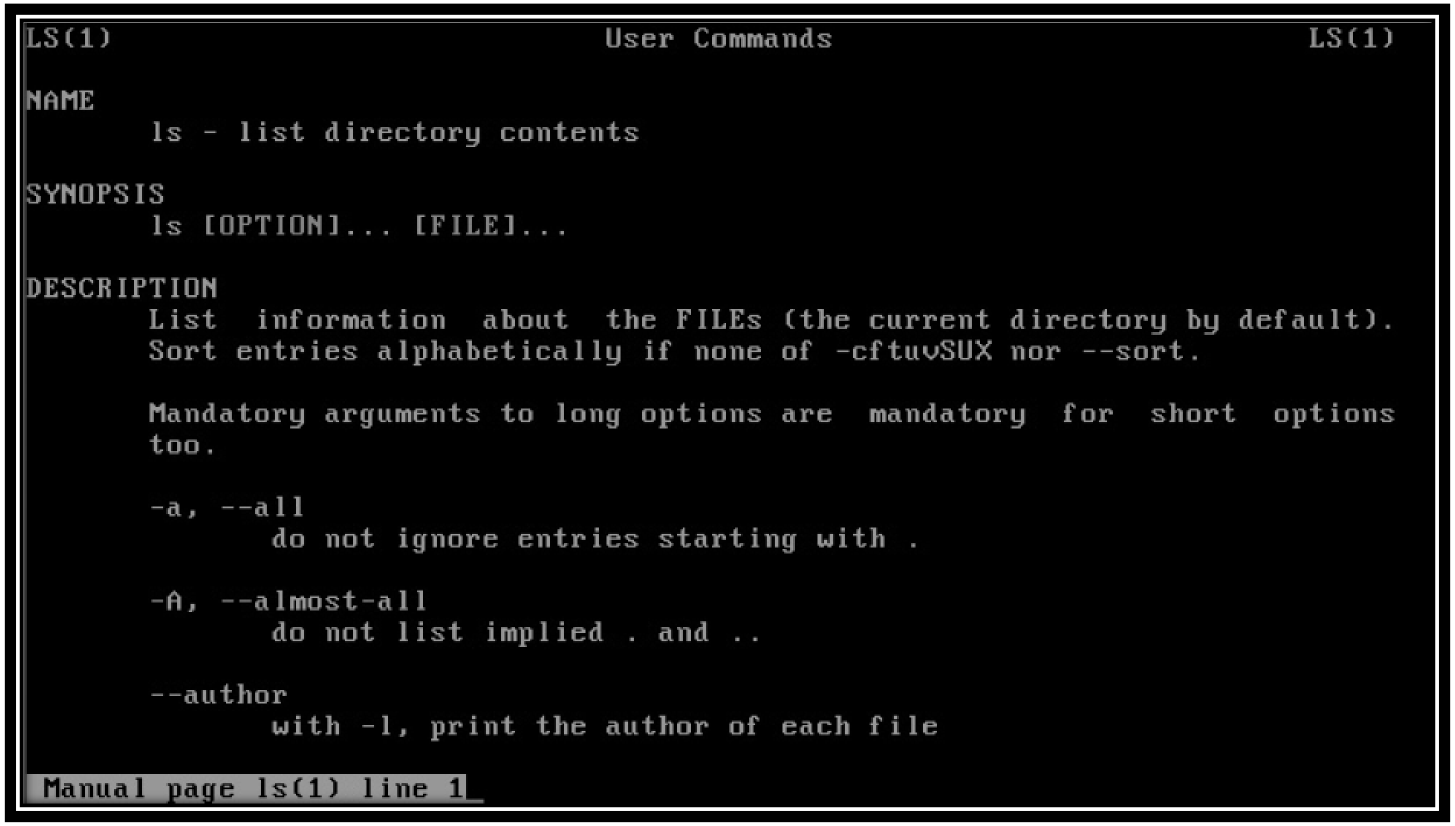
- **Manual sections**
  - **All of the man pages contained in various man directories compose the actual manual**
  - **Standards used by man page authors divide the manual into sections**



Section	Content
1	Programs and shell commands that can be used by any user
2	System functions provided by the Linux kernel
3	Library functions
4	Special files found in /dev
5	File format descriptions and conventions
6	Games
7	Miscellaneous conventions
8	Administrative utilities used by the root user
9	Kernel routine documentation

Getting Help for Linux Shell Commands

- Using man is simple, enter “man” then the name of the utility you need information about
  - man ls would display the following:



```
LS(1)                            User Commands                           LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).
       Sort entries alphabetically if none of -cftuvSUX nor --sort.

       Mandatory arguments to long options are  mandatory  for  short  options
       too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
              with -l, print the author of each file

 Manual page ls(1) line 1
```

Getting Help for Linux Shell Commands

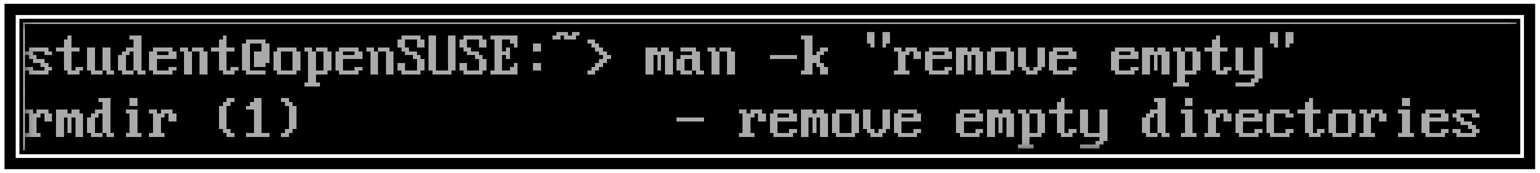
- In a man page all sections other than NAME are optional
- Authors can add new sections if required
- At the bottom of the man page there is a status indicator displaying:
  - the manual page displayed
  - top line currently displayed
  - percentage through the document at the line displayed

Getting Help for Linux Shell Commands

- ❑ **Keystrokes are used to navigate the man page**
  - **Down and Up arrows: Scroll up or down one line in a page**
  - **Page Up and Down: Scrolls up or down 14 lines in the page**
  - **Spacebar: Scrolls down 26 lines in the page**
  - **Home: Moves the user back to the beginning of the page**
  - **End: Moves the user to the end of the page**
  - **Q: exits the man utility**
- ❑ **Searching for specific words in a man page**
  - **While in the man page type / followed by the word you are searching for (/long)**
  - **Press n for the next instance**

Getting Help for Linux Shell Commands

- ❑ “man –k”
  - may need to use a utility but can’t remember it or the exact syntax
  - Use “man –k” to look for key words (man –k “remove empty”)
    - The shell will then list man pages with those key words



```
student@openSUSE:~> man -k "remove empty"
rmdir (1)                  - remove empty directories
```

    - In this example you can see that the rmdir command can be used to remove directories
    - Now do a man rmdir for specifics
- ❑ “apropos” command is similar to “man –k”
  - Example (apropos “remove empty”)

Exercise 2-5: Using man Pages

Please open your Practical Exercise book to Exercise 2-5.

Time to Complete: 5 Minutes

Using the info Utility

- **info Utility**
  - Serves a similar purpose as man
  - man pages have information but it is more syntax than informative
  - info is more of learning utility
  - Info nodes have similar info as man pages but more verbose
- **Launching info**
  - Simply enter “info” and then the command (info ls)
  - info divides information into nodes, man stores information on a single page
  - Keystrokes move you from node to node

Using the info Utility

- **Keystrokes for Navigating info Nodes**
  - **Down and Up Arrow: Scrolls up and down one line at a time**
  - **Page Up and Down: Scrolls up and down one page at a time**
  - **Spacebar: Scrolls down one page at a time**
  - **DEL or Backspace: Scrolls up one page at a time**
  - **Home: Moves you to the beginning of the node**
  - **End: Moves you to the end of the node**
  - **N: Moves to the next node**
  - **P: Takes you to the previous node**
  - **Q: Exits the info utility**

Exercise 2-6: Using info

Please open your Practical Exercise book to Exercise 2-6.

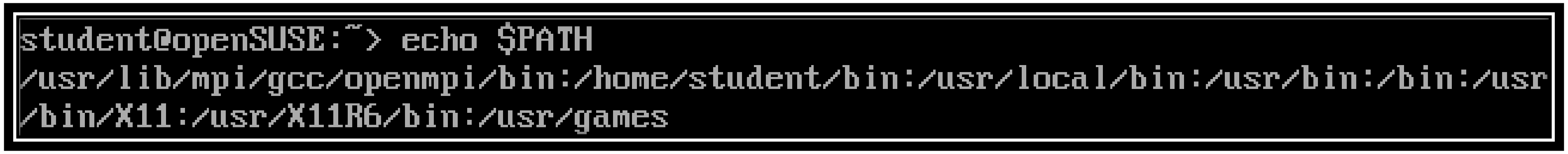
Time to Complete: 5 Minutes

What Are Environment Variables?

- **Variables**
  - **Area in system RAM, reserved to store values you put in**
- **Two different types**
  - **User–defined: User can create these themselves. This includes name and the contents**
  - **Environmental: Initially created, named and populated by the OS itself. Used to configure the systems computing environment**
- **Several Environmental Variables**
  - **BASH and SHELL**
  - **HISTFILE and HISTSIZE**
  - **PATH**

Viewing Variables and Their Values

- If the values of a single variable need to be viewed you can use the “echo” command:
  - echo $*variable* (example: echo $PATH)
  - $ is important as it identifies to echo to retrieve the contents of the variable

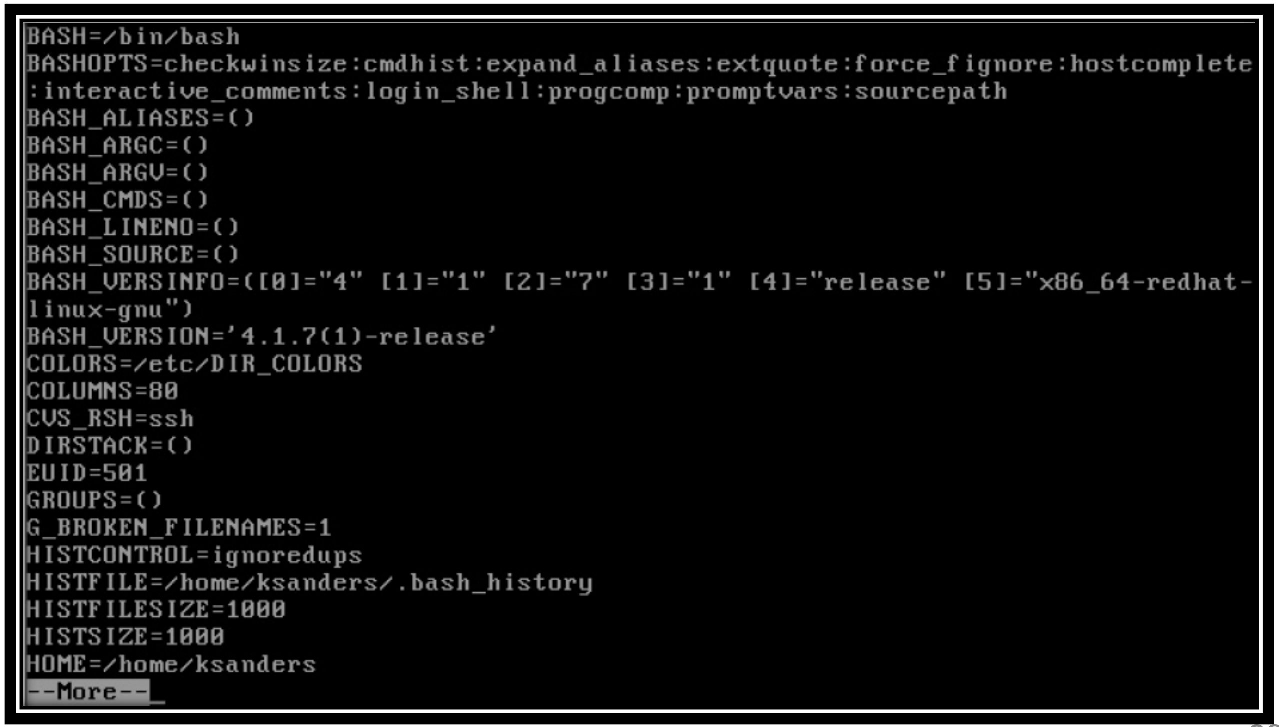


```
student@openSUSE:~> echo $PATH
/usr/lib/mpi/gcc/openmpi/bin:/home/student/bin:/usr/local/bin:/usr/bin:/bin:/usr
/bin/X11:/usr/X11R6/bin:/usr/games
```

- echo will display the contents of the variable which means you need to know the name of the variable, what if you don’t?

Viewing Variables and Their Values

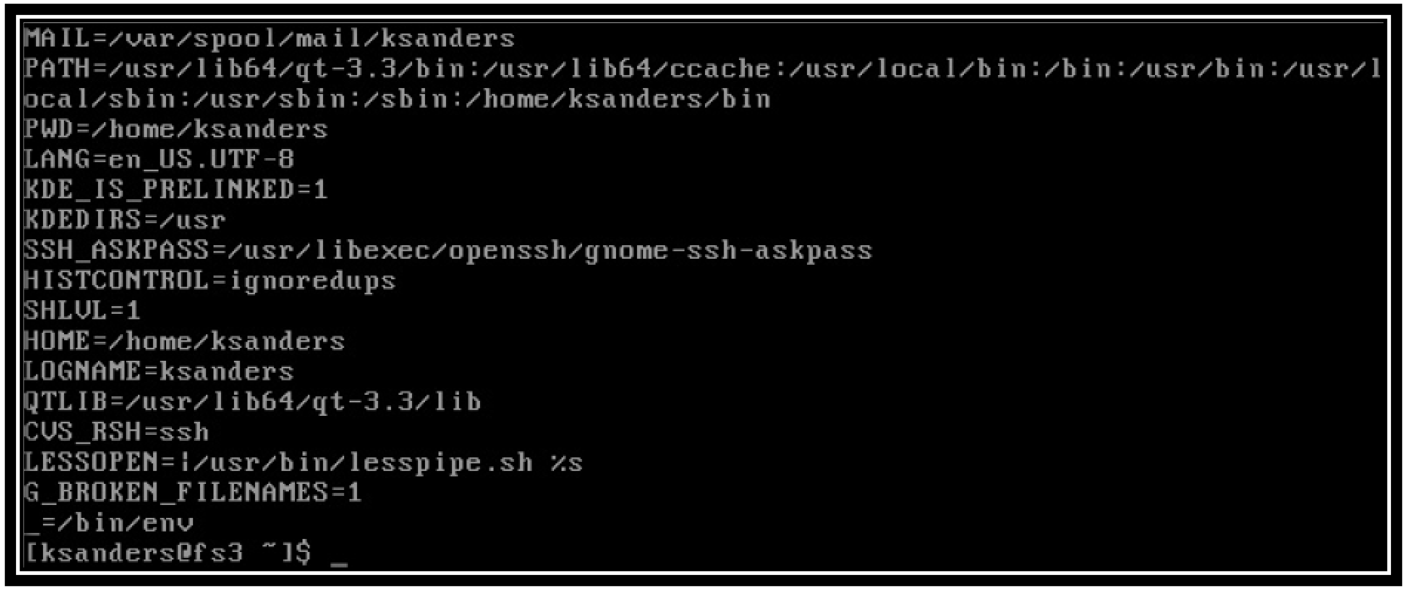
- To view all of the variables and their associated values use the “set” command at the shell prompt
  - This output can be very long so use the pipe (|) with “more” which will pause the output of the set command one page at a time
  - set | more



```
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:expand_aliases:extquote:force_fignore:hostcomplete
:interactive_comments:login_shell:progcomp:promptvars:sourcepath
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSINFO=([0]="4" [1]="1" [2]="7" [3]="1" [4]="release" [5]="x86_64-redhat-
linux-gnu")
BASH_VERSION='4.1.7(1)-release'
COLORS=/etc/DIR_COLORS
COLUMNS=80
CVS_RSH=ssh
DIRSTACK=()
EUID=501
GROUPS=()
G_BROKEN_FILENAMES=1
HISTCONTROL=ignoredups
HISTFILE=/home/ksanders/.bash_history
HISTFILESIZE=1000
HISTSIZE=1000
HOME=/home/ksanders
--More--
```

Viewing Variables and Their Values

- To view all of the variables and their associated values use the “env” command at the shell prompt
  - This output can be very long so use the pipe (|) with “more” which will pause the output of the env command one page at a time
  - env | more



```
MAIL=/var/spool/mail/ksanders
PATH=/usr/lib64/qt-3.3/bin:/usr/lib64/ccache:/usr/local/bin:/bin:/usr/bin:/usr/l
ocal/sbin:/usr/sbin:/sbin:/home/ksanders/bin
PWD=/home/ksanders
LANG=en_US.UTF-8
KDE_IS_PRELINKED=1
KDEDIRS=/usr
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
HISTCONTROL=ignoredups
SHLVL=1
HOME=/home/ksanders
LOGNAME=ksanders
QTLIB=/usr/lib64/qt-3.3/lib
CVS_RSH=ssh
LESSOPEN=|/usr/bin/lesspipe.sh %s
G_BROKEN_FILENAMES=1
_=/bin/env
[ksanders@fs3 ~]$ _
```

Setting the Value of a Variable

- Most of the variables on the system you will not change.
- What if you need to change a value in a variable to improve the user or your experience like:
  - Adding an additional directory to the PATH variable
  - Modifying the DISPLAY variable to send a display to a remote system
- Changing the values of an environmental variable is a relatively simple task but care must be taken!
- Let’s look at an example on the next slide.

Setting the Value of a Variable

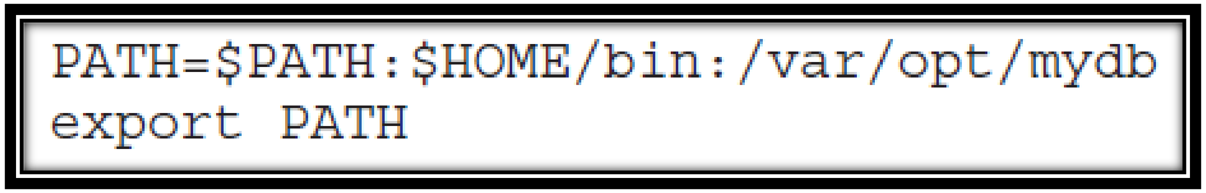
- You have installed an application named myapp in /var/opt/myapp
  - This path however does not appear in your PATH variable
  - You want to avoid typing the full path every time you use it
- The first section of the command would be the "variable=value" in this case: PATH=$PATH
  - $PATH must be specified to modify the variable
  - Without specifying $PATH you would erase and replace the old PATH variable which could cause system issues
- The second part of the command would be the path you want added :/var/opt/myapp
  - Complete command would be PATH=$PATH:/var/opt/myapp

Setting the Value of a Variable

- ❑ **So with the last slide the PATH variable has been modified with a new directory**
  - – **This change only applies to this shell session**
    - **Opening a new shell, the change would not be applied**
- ❑ **To make this change apply to all shells we must export the new value of the variable**
  - – **To do this enter "export *variable*" at the shell prompt**
  - – **In the example from the prior slide the command would be export PATH**
- ❑ **Once this command is entered the new value assigned to PATH is available to all shells**

Setting the Value of a Variable

- ❑ export does not make the change persistent
  - – New value to the variable will be removed on system reboot
- ❑ If the change needs to be persistent, modification to a bash configuration file will be required
  - – Adding the command to a global bash configuration (etc/profile) will apply the change to all users
  - – If the change only need to be applied to a single user, use the appropriate bash configuration file in the users home directory
- ❑ The following commands would modify the ~/.bash_profile file in the users home directory



```
PATH=$PATH:$HOME/bin:/var/opt/mydb
export PATH
```

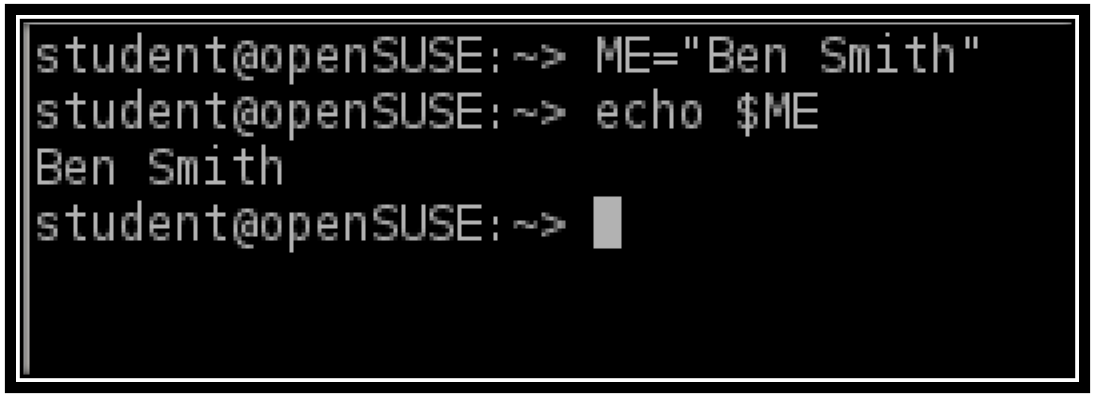
Exercise 2-7: Working with Environment Variables

Please open your Practical Exercise book to Exercise 2-7.

Time to Complete: 5 Minutes

Creating a User-Defined Variable

- User-Defined Variables
  - Your own, customized variables
- Commands are similar to commands used with Environmental Variables
  - variable=value



```
student@openSUSE:~> ME="Ben Smith"
student@openSUSE:~> echo $ME
Ben Smith
student@openSUSE:~> 
```

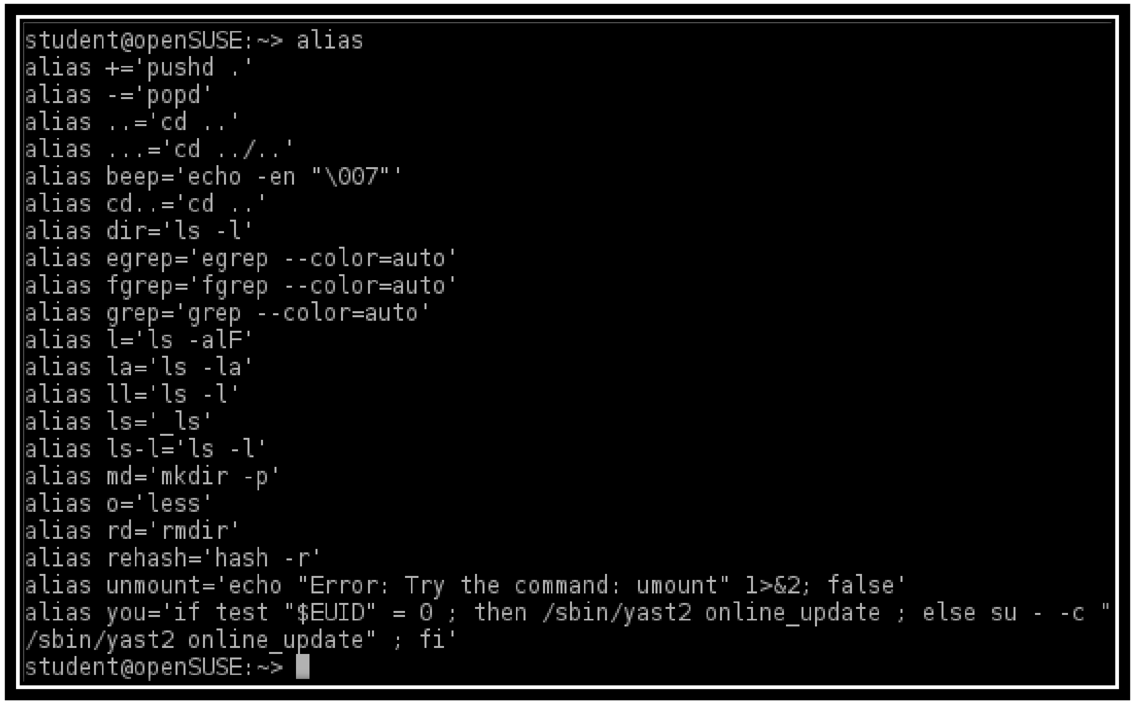
  - In this example a variable named ME was created with a value of Ben Smith

Creating a User-Defined Variable

- Like environment variables, user-defined variables are only available to the current shell instance
  - Need to export it to make it available to other shells
    - export ME for example
  - If the user-defined variable created needs to be persistent, it needs to be added to the appropriate bash configuration file
- Keep the following in mind when creating user-defined variables:
  - Names can contain letters or numbers, cannot begin with a number
  - Names can contain hyphens (-) or underscore characters (_)
  - Try to use all uppercase characters when naming variables

Working with Aliases

- An alias in Linux is a shortcut to a different file or command
- Upon bootup, a series of aliases is created for you
- Those aliases can be viewed by entering alias in the shell prompt



```
student@openSUSE:~> alias
alias +='pushd .'
alias -='popd'
alias ..='cd ..'
alias ...='cd ../..'
alias beep='echo -en "\007"'
alias cd..='cd ..'
alias dir='ls -l'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -alF'
alias la='ls -la'
alias ll='ls -l'
alias ls='_ls'
alias ls-l='ls -l'
alias md='mkdir -p'
alias o='less'
alias rd='rmdir'
alias rehash='hash -r'
alias unmount='echo "Error: Try the command: umount" 1>&2; false'
alias you='if test "$EUID" = 0 ; then /sbin/yast2 online_update ; else su - -c "
/sbin/yast2 online_update" ; fi'
student@openSUSE:~> 
```

Working with Aliases

- The commands in the screenshot on the prior slide are not commands but aliases that point to commands
- The 8th line down in the prior screenshot is:
  - alias dir='ls –l'
  - When you enter dir in the command shell you are actually doing a ls –l
- You can create your own aliases
  - This is done via the following syntax: alias *name*="*command*" entered in the command shell
    - alias ldir="ls -l" would create an alias named ldir that would create a long listing of your directory

Working with Aliases

- Multiple commands can be included in a single alias
  - Separate the commands with a semicolon (;)

```
alias mntdvd="mount -t iso9660 /dev/sr0 /media/dvd;ls -l /media/dvd"
```

$\underbrace{\texttt{alias mntdvd=}}_{\text{Alias}}\underbrace{\texttt{"mount -t iso9660 /dev/sr0 /media/dvd}}_{\text{Command 1}}\boxed{\texttt{;}}\underbrace{\texttt{ls -l /media/dvd"}}_{\text{Command 2}}$

  - In the above example an alias was created named “mntdvd” to mount a DVD in the optical drive (Command 1)
  - The second command in the alias will generate a long list of the files the DVD contains (note the semicolon)
- Like variables, aliases you define are not persistent by default
  - Aliases will need to be added to the appropriate bash configuration file to be made persistent

Exercise 2-8: Working with Aliases

Please open your Practical Exercise book to Exercise 2-8.

Time to Complete: 5 Minutes

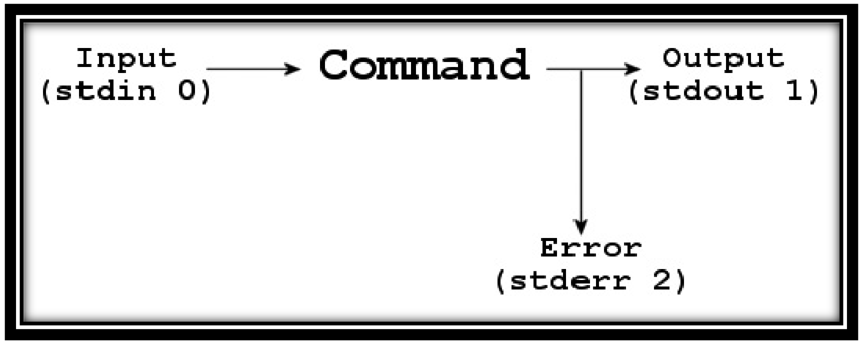
Using Redirecting and Piping

- **The bash shell is powerful and flexible**
  - **One reason for this is the ability to manipulate command input and output**
- **There are three areas that need to be discussed to use this ability**
  - **Standard bash File Descriptors**
  - **Redirecting Output and Input for Shell Commands**
  - **Piping Information**

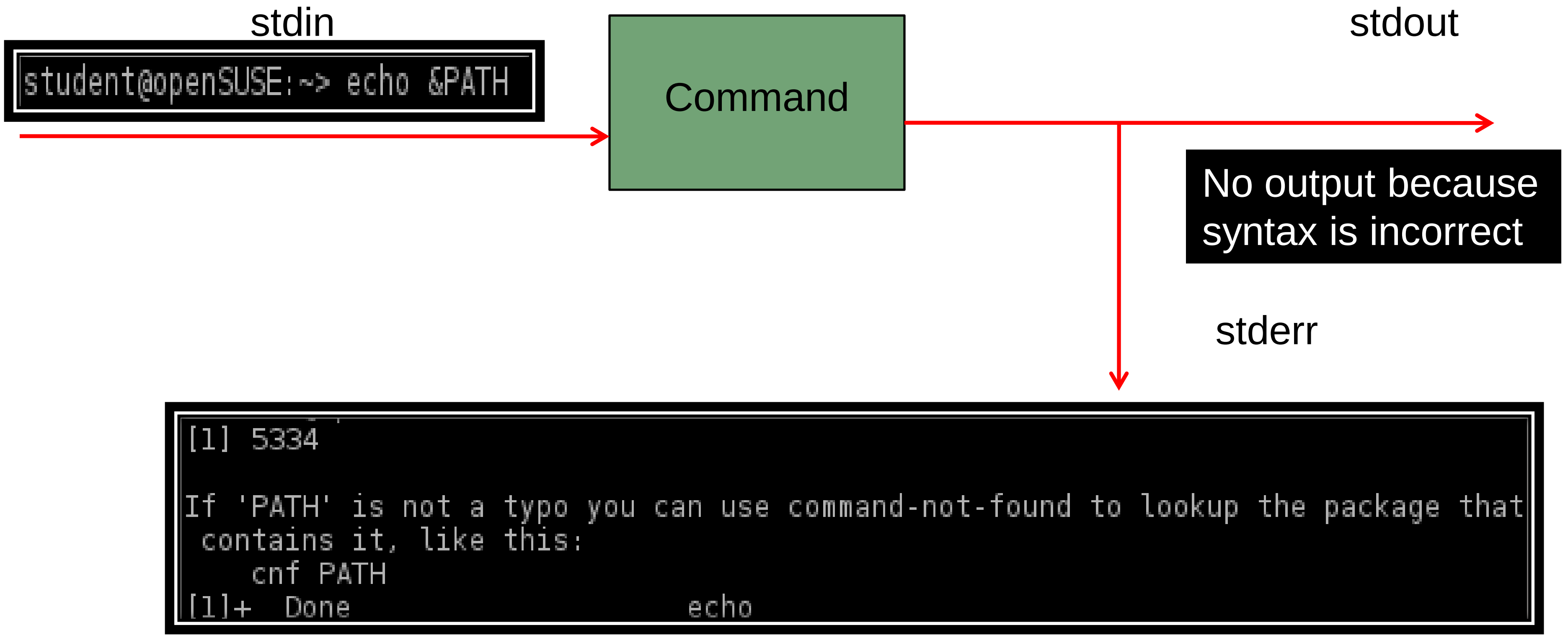
Using Redirecting and Piping

Standard bash File Descriptors

- Three file descriptors are available for every command entered at a shell prompt
  - stdin (Standard Input): input provided to a particular command to process, represented by the number 0
  - stdout (Standard Output): the output from a particular command, represented by the number 1
  - stderr (Standard Error): the error code generated, if any, by a command, represented by the number 2
- Not all commands use all three descriptors, but most do



Using Redirecting and Piping



```
student@openSUSE:~> echo &PATH
```

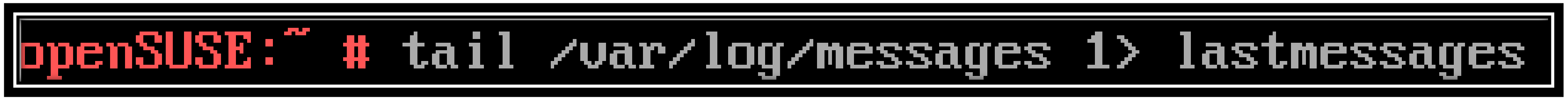
```
[1] 5334

If 'PATH' is not a typo you can use command-not-found to lookup the package that
 contains it, like this:
    cnf PATH
[1]+  Done                    echo
```

Using Redirecting and Piping

- Using the three descriptors we can manipulate where a command gets it’s input from and where it sends its output
- Redirecting output
  - By default the output of a command is displayed on the screen
  - You can redirect it from displaying on the screen to a text file
    - Especially a long output
- Redirection is done by use of the greater than (>) sign
  - *command output> filename_or_device*

Using Redirecting and Piping

- Let’s look at an example of redirecting:

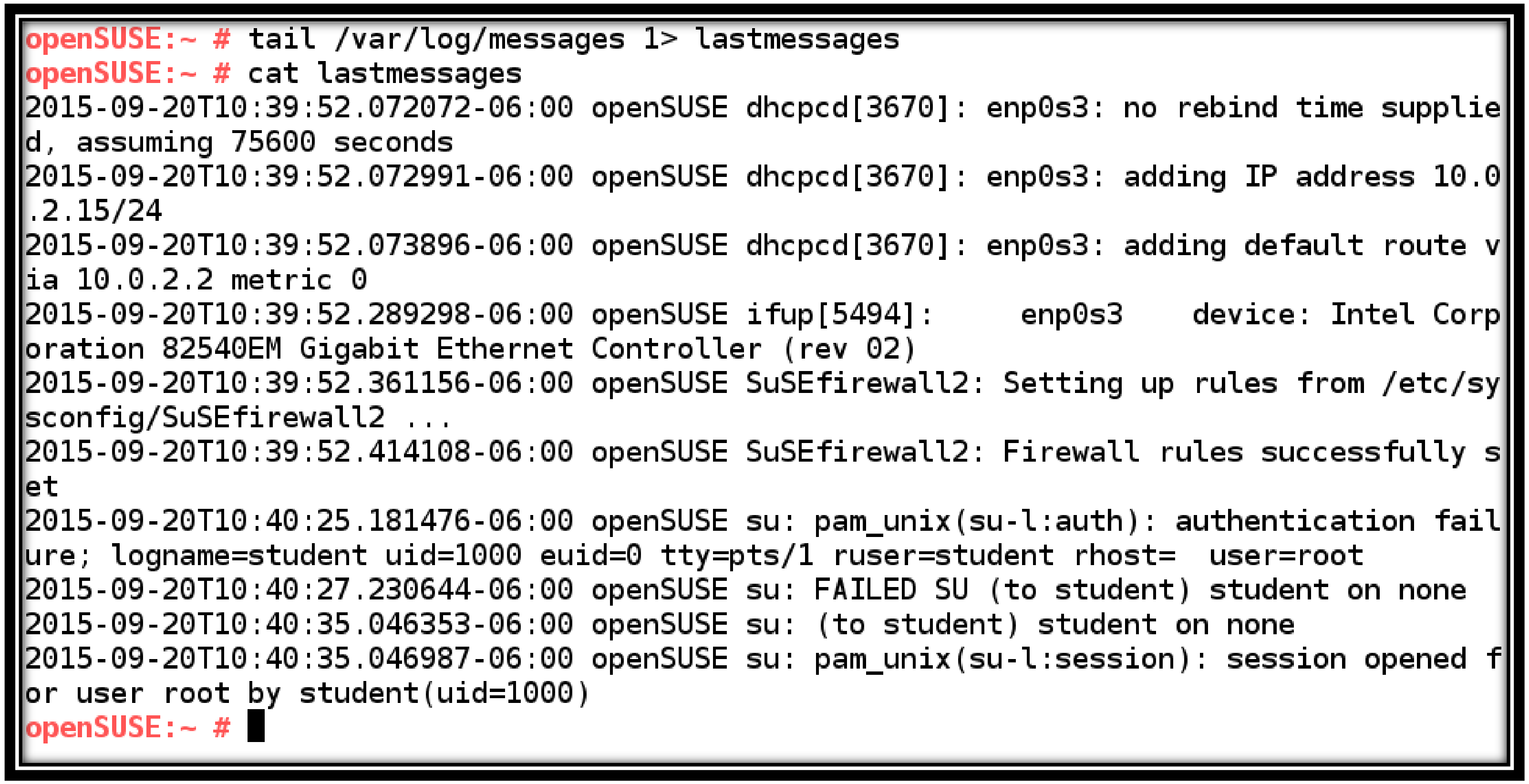


```
openSUSE:~ # tail /var/log/messages 1> lastmessages
```

- In this example the “tail” command is being used to display the last entries in the system log file
- The “1> lastmessage” is the redirect of the output to a text file titled “lastmessages”
  - To view the log entries now the lastmessages file would need to be opened
    - cat lastmessages
  - The log entries in that file would now be displayed
- The “1” could be left out, the shell defaults to stdout

Using Redirecting and Piping

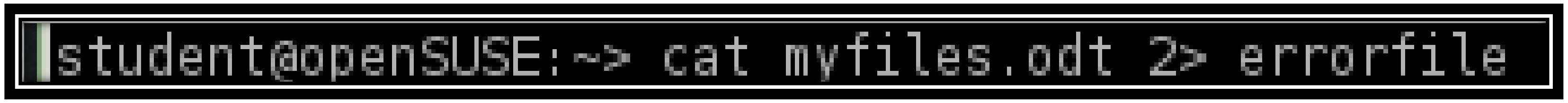
Output of the “cat lastmessages”



```
openSUSE:~ # tail /var/log/messages 1> lastmessages
openSUSE:~ # cat lastmessages
2015-09-20T10:39:52.072072-06:00 openSUSE dhcpcd[3670]: enp0s3: no rebind time supplie
d, assuming 75600 seconds
2015-09-20T10:39:52.072991-06:00 openSUSE dhcpcd[3670]: enp0s3: adding IP address 10.0
.2.15/24
2015-09-20T10:39:52.073896-06:00 openSUSE dhcpcd[3670]: enp0s3: adding default route v
ia 10.0.2.2 metric 0
2015-09-20T10:39:52.289298-06:00 openSUSE ifup[5494]:     enp0s3    device: Intel Corp
oration 82540EM Gigabit Ethernet Controller (rev 02)
2015-09-20T10:39:52.361156-06:00 openSUSE SuSEfirewall2: Setting up rules from /etc/sy
sconfig/SuSEfirewall2 ...
2015-09-20T10:39:52.414108-06:00 openSUSE SuSEfirewall2: Firewall rules successfully s
et
2015-09-20T10:40:25.181476-06:00 openSUSE su: pam_unix(su-l:auth): authentication fail
ure; logname=student uid=1000 euid=0 tty=pts/1 ruser=student rhost=  user=root
2015-09-20T10:40:27.230644-06:00 openSUSE su: FAILED SU (to student) student on none
2015-09-20T10:40:35.046353-06:00 openSUSE su: (to student) student on none
2015-09-20T10:40:35.046987-06:00 openSUSE su: pam_unix(su-l:session): session opened f
or user root by student(uid=1000)
openSUSE:~ # █
```

Using Redirecting and Piping

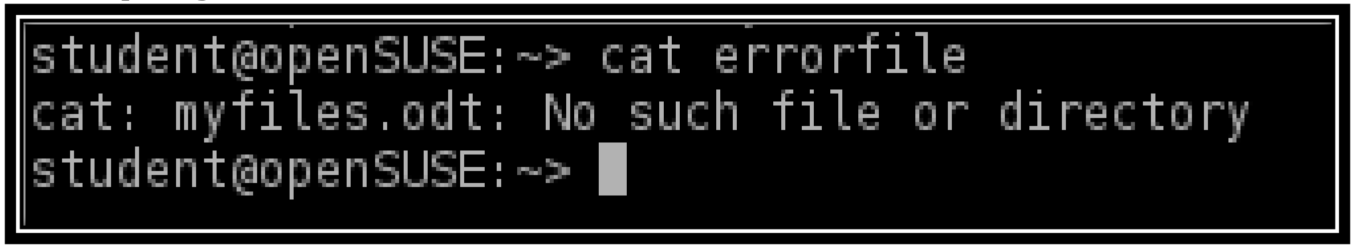
- The same technique can be used to redirect errors (stderr) from the screen to a file:



```
student@openSUSE:~> cat myfiles.odt 2> errorfile
```

  – In the above example errors from the "cat" commands will be redirected to a file titled "errorfile"

  – When opening the file with "cat" titled "errorfile" the following is displayed:



```
student@openSUSE:~> cat errorfile
cat: myfiles.odt: No such file or directory
student@openSUSE:~>
```

  – An error was generated when the "cat" command was executed to open a file that does not exist, myfiles.odt

Using Redirecting and Piping

- **Redirecting outputs or inputs to a file will cause the file to be:**
  - **Created if new**
  - **Erased if existing and new file created**
- **There is a way to append the output or errors to an existing file without replacing it**
  - **To redirect and append the stdout to an existing file use the “>>”**
  - **Example: *tail /var/log/messages 1>> lastmessages***

Using Redirecting and Piping

- **The stdout and stderr can be redirected to separate text files simultaneously:**
  - **Syntax:** ***command 1> stdout_filename 2> stderr_filename***
  - **Example:** ***mount 1> mntok 2> mnterr***
  - **In the above example the output of mount would be placed in a file titled “mntok” and errors would go to the file “mnterr”**
- **The stdout and stderr can be redirected to the same text file:**
  - **Syntax:** ***command 1> filename 2> &1***
  - **Example:** ***mount 1> mntall 2> &1***
  - **In the above example the stdout will be written to the “mntall” file first the stderr will then be sent to the stdout output**

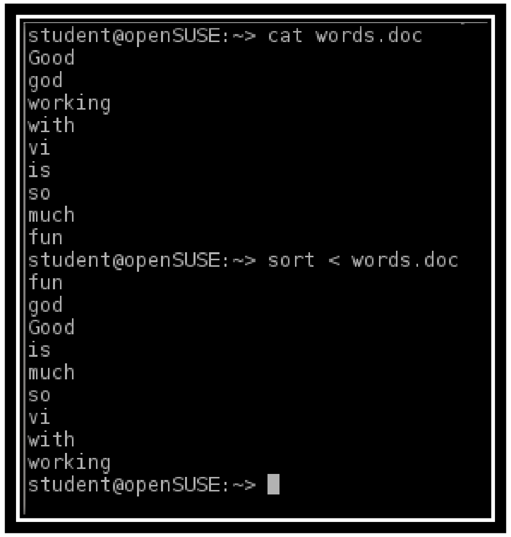
Using Redirecting and Piping

Redirecting input

– Allows you to send a lot of text to a command expecting it

– Reverse the characters used in the stdout and stderr

- Syntax: *command* < *input_text_or_file*
- sort < words.doc
- In this example the list of words in the words.doc would be sorted via the sort command and look like this:



```
student@openSUSE:~> cat words.doc
Good
god
working
with
vi
is
so
much
fun
student@openSUSE:~> sort < words.doc
fun
god
Good
is
much
so
vi
with
working
student@openSUSE:~> 
```

Exercise 2-9: Redirecting Input and Output

Please open your Practical Exercise book to Exercise 2-9.

Time to Complete: 5 Minutes

Piping Information

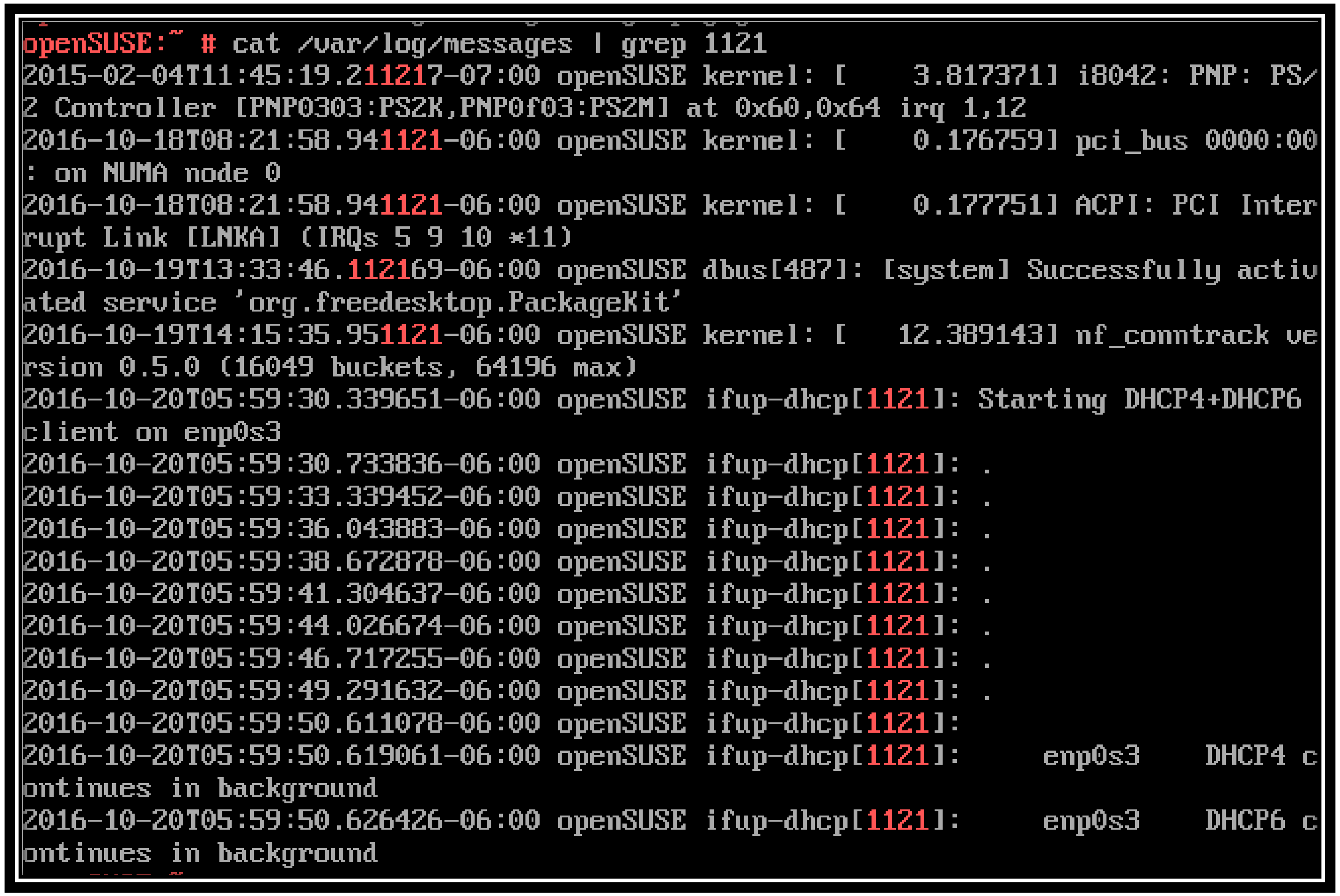
- **Piping**
  - **allows us to take the output from the first command and use it as the input for another**
  - **Accomplished using the pipe character (|)**
  - **Example: *cat /var/log/messages* | *more***
  - **In the above example the cat command would normally read the contents of the "/var/log/messages" and display it**
  - **With the pipe however the shell will not display it but send it to the next command, "more"**
  - **The more command will take the stdout from cat and display the output one line at a time**
  - **Only the last command in the pipe will display an output on the screen**

Piping Information

- Pipes can be used with any command that produces an output and accept an input
- grep is a great command to use with pipes
  - grep is used to search input files for lines containing a match to a given pattern list
  - Syntax: *command* | grep *expression*
  - Example: cat /var/log/messages | grep 1121
  - In the above example the output of the cat command to display system log will be piped to the grep command and all instances of “1121” will be displayed

Piping Information

❑ **Results of the cat /var/log/messages | grep 1121 command:**



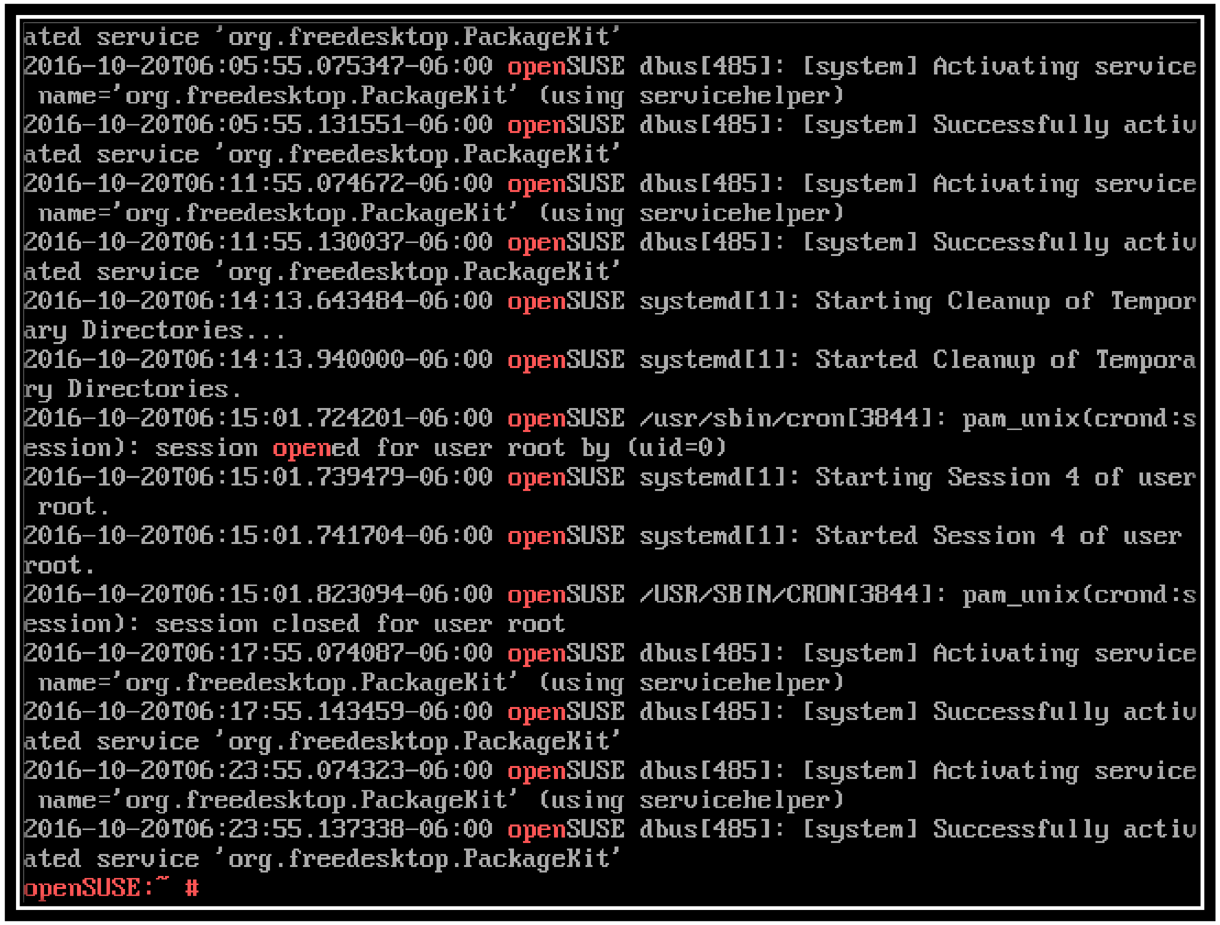
```
openSUSE:~ # cat /var/log/messages | grep 1121
2015-02-04T11:45:19.211217-07:00 openSUSE kernel: [    3.817371] i8042: PNP: PS/
2 Controller [PNP0303:PS2K,PNP0f03:PS2M] at 0x60,0x64 irq 1,12
2016-10-18T08:21:58.941121-06:00 openSUSE kernel: [    0.176759] pci_bus 0000:00
: on NUMA node 0
2016-10-18T08:21:58.941121-06:00 openSUSE kernel: [    0.177751] ACPI: PCI Inter
rupt Link [LNKA] (IRQs 5 9 10 *11)
2016-10-19T13:33:46.112169-06:00 openSUSE dbus[487]: [system] Successfully activ
ated service 'org.freedesktop.PackageKit'
2016-10-19T14:15:35.951121-06:00 openSUSE kernel: [   12.389143] nf_conntrack ve
rsion 0.5.0 (16049 buckets, 64196 max)
2016-10-20T05:59:30.339651-06:00 openSUSE ifup-dhcp[1121]: Starting DHCP4+DHCP6
client on enp0s3
2016-10-20T05:59:30.733836-06:00 openSUSE ifup-dhcp[1121]: .
2016-10-20T05:59:33.339452-06:00 openSUSE ifup-dhcp[1121]: .
2016-10-20T05:59:36.043883-06:00 openSUSE ifup-dhcp[1121]: .
2016-10-20T05:59:38.672878-06:00 openSUSE ifup-dhcp[1121]: .
2016-10-20T05:59:41.304637-06:00 openSUSE ifup-dhcp[1121]: .
2016-10-20T05:59:44.026674-06:00 openSUSE ifup-dhcp[1121]: .
2016-10-20T05:59:46.717255-06:00 openSUSE ifup-dhcp[1121]: .
2016-10-20T05:59:49.291632-06:00 openSUSE ifup-dhcp[1121]: .
2016-10-20T05:59:50.611078-06:00 openSUSE ifup-dhcp[1121]:
2016-10-20T05:59:50.619061-06:00 openSUSE ifup-dhcp[1121]:     enp0s3    DHCP4 c
ontinues in background
2016-10-20T05:59:50.626426-06:00 openSUSE ifup-dhcp[1121]:     enp0s3    DHCP6 c
ontinues in background
```

Piping Information

- **What if the command used displayed a large amount of information?**
  - **Example: cat /var/log/messages | grep open**
  - **The "more" command could be used allowing the screen to be paused one page at a time**
    - **Example: cat /var/log/messages | grep open | more**
- **We may need the output of commands to be displayed and written to a file**
  - **The "tee" command can accomplish this**
  - **Syntax: *command* | tee *file_name***
  - **Example: ls –l | tee listout.txt**
  - **Example: cat /var/log/messages | grep pid | tee ~/info.txt**

Piping Information

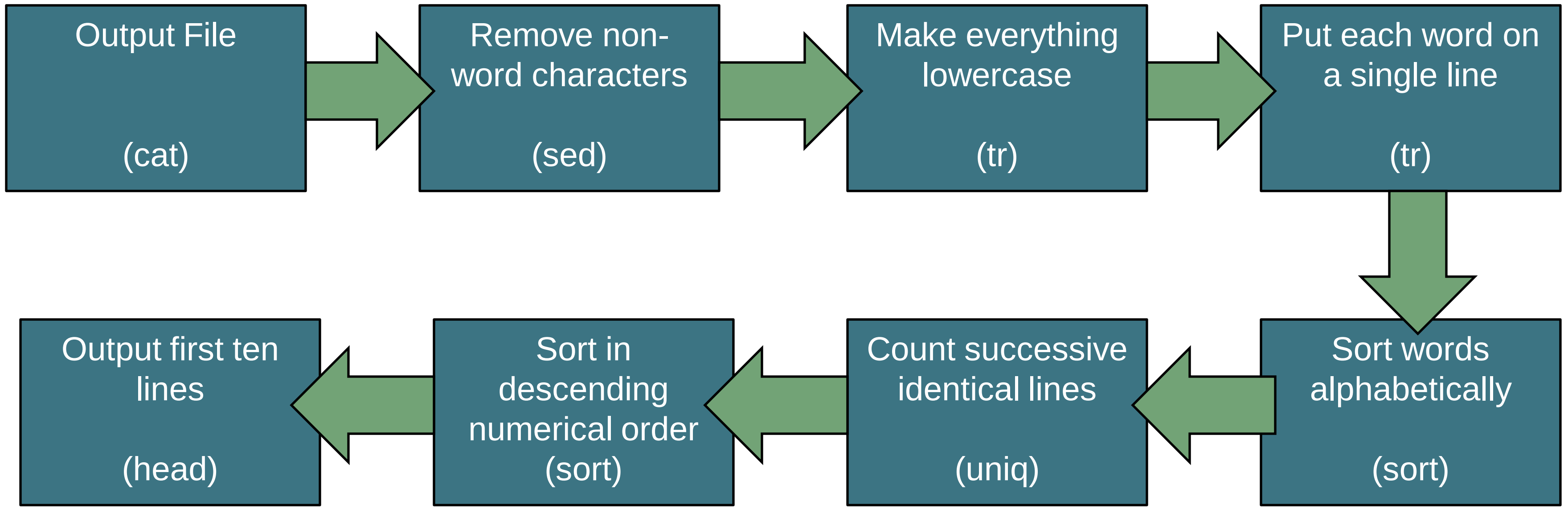
❑ Results of the cat /var/log/messages | open command:



```
ated service 'org.freedesktop.PackageKit'
2016-10-20T06:05:55.075347-06:00 openSUSE dbus[485]: [system] Activating service
 name='org.freedesktop.PackageKit' (using servicehelper)
2016-10-20T06:05:55.131551-06:00 openSUSE dbus[485]: [system] Successfully activ
ated service 'org.freedesktop.PackageKit'
2016-10-20T06:11:55.074672-06:00 openSUSE dbus[485]: [system] Activating service
 name='org.freedesktop.PackageKit' (using servicehelper)
2016-10-20T06:11:55.130037-06:00 openSUSE dbus[485]: [system] Successfully activ
ated service 'org.freedesktop.PackageKit'
2016-10-20T06:14:13.643484-06:00 openSUSE systemd[1]: Starting Cleanup of Tempor
ary Directories...
2016-10-20T06:14:13.940000-06:00 openSUSE systemd[1]: Started Cleanup of Tempora
ry Directories.
2016-10-20T06:15:01.724201-06:00 openSUSE /usr/sbin/cron[3844]: pam_unix(crond:s
ession): session opened for user root by (uid=0)
2016-10-20T06:15:01.739479-06:00 openSUSE systemd[1]: Starting Session 4 of user
 root.
2016-10-20T06:15:01.741704-06:00 openSUSE systemd[1]: Started Session 4 of user
root.
2016-10-20T06:15:01.823094-06:00 openSUSE /USR/SBIN/CRON[3844]: pam_unix(crond:s
ession): session closed for user root
2016-10-20T06:17:55.074087-06:00 openSUSE dbus[485]: [system] Activating service
 name='org.freedesktop.PackageKit' (using servicehelper)
2016-10-20T06:17:55.143459-06:00 openSUSE dbus[485]: [system] Successfully activ
ated service 'org.freedesktop.PackageKit'
2016-10-20T06:23:55.074323-06:00 openSUSE dbus[485]: [system] Activating service
 name='org.freedesktop.PackageKit' (using servicehelper)
2016-10-20T06:23:55.137338-06:00 openSUSE dbus[485]: [system] Successfully activ
ated service 'org.freedesktop.PackageKit'
openSUSE:~ #
```

❑ Pipelines

– **Data flows through the commands like oil through a pipeline:**


Output File (cat)
Remove non-word characters (sed)
Make everything lowercase (tr)
Put each word on a single line (tr)
Sort words alphabetically (sort)
Count successive identical lines (uniq)
Sort in descending numerical order (sort)
Output first ten lines (head)

```
cat textfile | sed ‘s/[^[:alpha:][:space:]]/ /g’ | \
    tr ‘[:upper:]’ ‘[:lower:]’ | tr -s ‘ \t’ ‘  ’ | sort | uniq -c \
    sort -rn | head -10
```

Exercise 2-10: Using Pipes

Please open your Practical Exercise book to Exercise 2-10.

Time to Complete: 5 Minutes

Summary

- ❑ How the Linux shell works
- ❑ Using the shell prompt
- ❑ Getting help for Linux shell commands
- ❑ Working with environment variables
- ❑ Working with aliases
- ❑ Using redirection and piping

Questions

Questions?

Check on Learning

Question 1

Which shell is the default shell for most Linux distributions?

A. sh

B. csh

C. bash

D. zsh

Check on Learning

Question 2

Your organization uses openSUSE Linux in a bash shell CLI-only environment. You have a shell prompt running a program. You now need to run another program but you do not want to halt the current program to do so. What can you do to launch the second program?

A. Nothing. You must halt the first program and access the first shell.

B. Press CTRL-Pause pausing the running program and allowing you to access the shell.

C. Press ALT-F2 to open a new shell session and run the program.

D. Press CTRL-SHIFT-F6 which will open a new shell.

Check on Learning

Question 3

You have copied an executable from shared drive named newupdate.sh to the /tmp directory on your Linux system. You change directory to the /tmp directory via the shell prompt. From the shell prompt you then enter newupdate.sh but the shell indicates it cannot find the file. What can you do to troubleshoot and get this file run?

A. Capitalize all the letters in the command and try again.

B. Add ./ at the before the filename and run the command again.

C. Enter the filename again but remove the .sh extension.

D. Move the file to your home directory and then execute it.

Check on Learning

Question 4

You are having an issue and need to identify what directories in the Linux file system are in path. What command listed could you use to get that information? (Select two.)

- **A. env**
- **B. show $PATH**
- **C. man path**
- **D. echo $PATH**
- **E. writeln PATH**

Check on Learning

Question 5

You have copied an executable file name update1.sh to /tmp on your Linux. You open a shell and change directory to /tmp. At the prompt you enter ./Update1.sh but the shell indicates that the file cannot be found. What can you do to run this executable?

A. **Enter the filename in all lowercase letters.**

B. **Add .\ before the filename and enter the command again.**

C. **Remove the extension of .sh and attempt to run again.**

D. **Move the file to the home directory and run it from there.**

Check on Learning

Question 6

Which Linux utility can be used to change to a different user account at the shell prompt?

A. user

B. chuser

C. swuser

D. su

Check on Learning

Question 7

Which utility is used to view manual pages?

A. man

B. manual

C. gman

D. kwrite

Check on Learning

Question 8

You want to add the ~/temp directory to your system’s PATH environment variable. You do not want to overwrite the existing directories in your current path. You enter PATH=PATH:~/temp at the shell prompt. Did you carry out the task correctly with that command?

A. Yes, this command worked correctly.

B. No, you must first export the variable before setting it.

C. No, the $ must be used before each PATH variable.

D. No, the $ must be used before the second PATH variable

Check on Learning

Question 9

Which file descriptor refers to the text displayed on the screen after a command has been executed?

A. stdin

B. stdout

C. stdisplay

D. stdoutput

E. stderr

Check on Learning

Question 10

You want to send the standard output (stdout) and the standard error (stderr) from the tail /var/log/ firewall command to a file named lastevents in the current directory you are in. Which command will complete that task?

A. tail /var/log/firewall 1> lastevents 2> lastevents

B. tail /var/log/firewall > lastevents

C. tail /var/log/firewall > lastevents 2> &1

D. tail /var/log/firewall 1&2> lastevents