

CYBR 4423

Linux/Unix Administration

Linux Shell Basics

Overview

Shell

- Shell builtin commands
- Command line scripting

Bash scripting basics

- Input/output
- Piping
- Variables and data types

Shell

A Unix/Linux shell is a piece of software (command-line interpreter) that provides an interface for users to access the services of the OS kernel

Major shells

Bourne shell compatibles

Bourne shell: `/bin/sh`

Bash (Bourne-Again shell): `/bin/bash`

C Shell (csh)

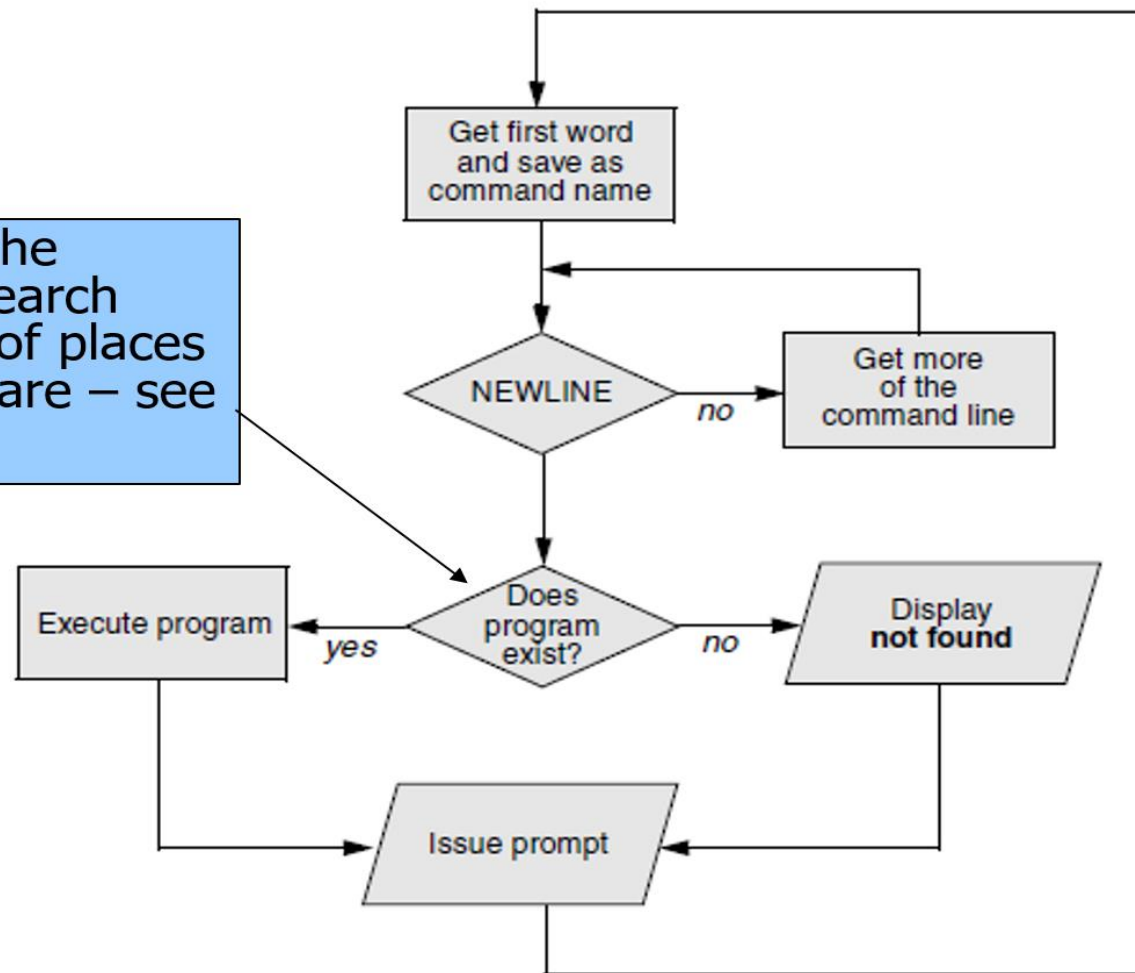
See the `/etc/shells` file for valid shells (not installed shells)

Ubuntu (and most Linux distros) uses bash as the default shell

You can switch to a different shell at any time use the "chsh" command

Command Execution

Shell will look for the command from "Search Path" – a number of places where commands are – see slides later



Get Command Reference

man

View command manuals (press "q" to exit)

type

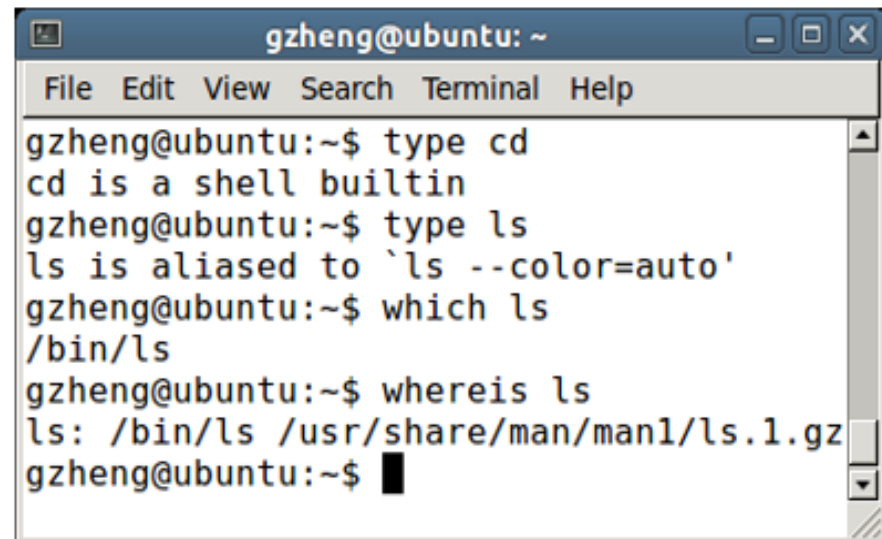
Show the type of a command

which

Show the file location of a command.

whereis

Locate binary, source, manual, configuration, and other files related to a command.

A terminal window titled 'gzheng@ubuntu: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and outputs:

```
gzheng@ubuntu:~$ type cd
cd is a shell builtin
gzheng@ubuntu:~$ type ls
ls is aliased to `ls --color=auto'
gzheng@ubuntu:~$ which ls
/bin/ls
gzheng@ubuntu:~$ whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.gz
gzheng@ubuntu:~$
```

Basic Command Operations

Auto completion: use tab key

No need to type all characters!

This applies to commands and directory/file names

Command history

Use up/down arrow key to navigate through commands entered before.

Use "history" command and "!" operator

[Linux C and history command](#)

[Linux Command Line History](#)

Command editing

Use left/right arrow keys, del, backspace to edit a command

Scripting

Scripting vs. Programming

Scripting language in Linux

Shell (Bash)

Perl

Python

Two scripting modes

Command line scripting

Script file execution

Command Line Scripting

Write scripts directly in the shell at the command prompt

Command editing

Use up arrow key to get previous commands

Ctrl + A to go to the beginning of the command

Ctrl + E to go to the end of the command

Multi line commands

Use “\” to indicate a soft return and continue on the next line

Multiple commands on one line

Use “;” to separate commands

Shell Built-in Commands

Shell built-ins are commands interpreted by the shell directly (no separate executable files). Examples:

- cd
- pwd
- type
- echo
- alias

Use "type" command to see if a command is a builtin

[Reference](#)

Command Alias

An alias is a (usually short) name that the shell translates into another (usually longer) name or (complex) command.

Aliases allow you to define new commands.

Example

Enter "alias" without any argument to check the current aliases defined

```
#> alias ls='ls -l'  
#> alias cp='cp -i'  
#> alias dir='ls -l'
```

When an alias has a space in its name

Use "" (quotation marks)

and

avoid calling an alias

Remove alias

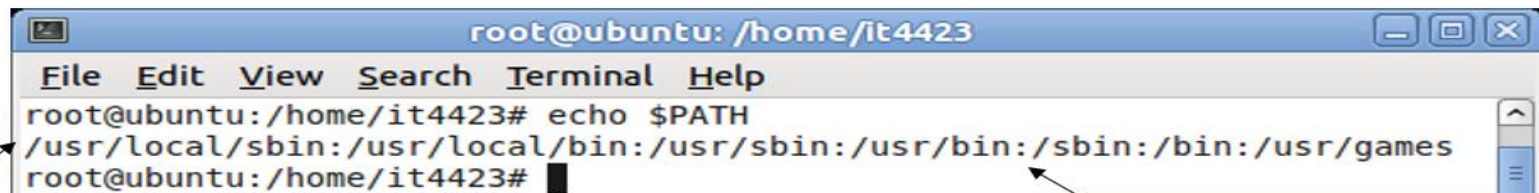
Use the "unalias" command followed by the alias name



Command Search Path

Search path

These are the directories to search when a command is entered without its path.
Paths are stored in the \$PATH environment variable

A terminal window titled 'root@ubuntu: /home/it4423' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the command 'echo \$PATH' being executed, resulting in the output: '/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games'. The prompt 'root@ubuntu:/home/it4423#' is visible at the bottom.

```
root@ubuntu: /home/it4423
File Edit View Search Terminal Help
root@ubuntu:/home/it4423# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
root@ubuntu:/home/it4423#
```

Note that the current directory (represented by a dot .) is not in search path. Why?

Each directory is separated by :

```
#>PATH=$PATH:/dir/path; export PATH
```

This is a second command to make PATH global.

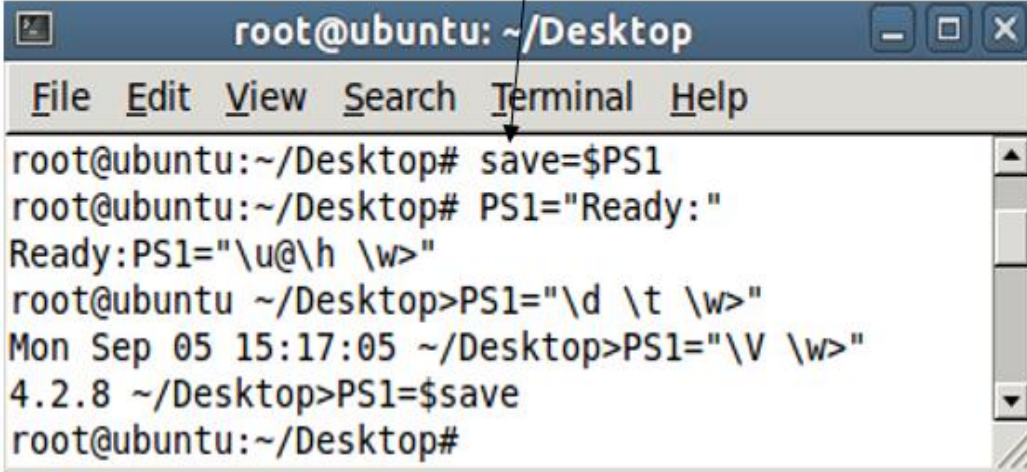
Changing Command Prompt

Command prompt is usually used to display useful environment information such as

Current user, directory, date, etc.

Change the “PS1” variable to change command prompt

Save the original PS1 to a temporary variable "save"



```
root@ubuntu: ~/Desktop
File Edit View Search Terminal Help
root@ubuntu:~/Desktop# save=$PS1
root@ubuntu:~/Desktop# PS1="Ready:"
Ready:PS1="\u@\h \w>"
root@ubuntu ~/Desktop>PS1="\d \t \w>"
Mon Sep 05 15:17:05 ~/Desktop>PS1="\V \w>"
4.2.8 ~/Desktop>PS1=$save
root@ubuntu:~/Desktop#
```

[Reference](#)

Other Environmental Variables

Prompt statement

\$PS1 - command prompt

\$PS2 - command continuation prompt

Other commonly used ones

\$SHELL - current shell

\$HOME - home directory

\$PWD - current working directory

\$PATH - command search path

Use "printenv" command to show all environmental variables

Input and Output

Every process has at least 3 communication channels available

"standard input"	(STDIN)
"standard output"	(STDOUT)
"standard error"	(STDERR)

These channels are setup by the kernel, so the process itself doesn't necessarily know them

Most commands accept their input from STDIN and write their output to STDOUT. They write error messages to STDERR

In the context of an interactive terminal window

STDIN normally reads from the keyboard

STDOUT and STDERR write their output to the screen

Reroute from/to Files

Use “>” to redirect screen output to files

Use “>>” to append to a file rather than to overwrite it

```
#>echo "hello, world" > file1  
#>echo "hello, world, again" >> file1
```

Use “<” to get input from a file

```
#>read variable1 < file1
```

Use “2>” to redirect errors to a file

Use “/dev/null” if errors should be ignored

```
#>badcommand 2> file1  
#>badcommand 2> /dev/null
```

Pipe

Pipe operator: |

To connect the STDOUT of one command to the STDIN of another

```
#>ls -lat | head -2
```

Filter

Any well-behaved command that reads STDIN and writes STDOUT can be used as a filter (that is, a component of a pipeline) to process data.

Common filter commands:

grep, wc, head, tail, tee, cut, sort, tr, cat

[Unix Filter](#)

More Pipe Examples

If file list is too long, use "less" to see them in pages.

```
#>ls -l | less
```

Count how many files

```
#>ls -l | wc -l
```

Look for subdirectories – the first letter in each file is "d" for directories

```
#>ls -l | grep ^d
```

Provide a value to a program reads from user put

grep

grep is used to find text within files

grep [options] PATTERN [FILE...]

Options

- i ignore case
- w whole word
- v reverse results
- o resulted phrase only

Pattern

Strings or regular expressions

More grep Examples

File name ending with ".jpg"

```
#>ls -l | grep .jpg$
```

Get only the match part, not the whole line (-o option)

```
#>ls -l | [a-z0-9-]*.jpg -o
```

[More examples](#)

[grep reference](#)

[Complete Regex reference](#)

Variables

Variable naming

Variable names are case sensitive

All-caps names typically suggest environment variables or variables read from global configuration files

Local variables are all-lowercase with components separated by underscores

Assignment

All variables are of the string data type when assigned

Referencing variables

Use the "\$"+variable name

Use {} around variable name optionally

```
#> var1="today"
```

no space around the = symbol

```
#>var1=1000; printf "User input: $var1\n"  
User input: 1000
```

I/O Command: echo

Use echo command to send results to “STDOUT”

```
#>echo "hello, world"  
hello, world
```

One argument

Multiple arguments followed

```
#>echo "hello," "world"  
hello, world
```

Two arguments

I/O Command: printf

Similar to “echo”, but with some formatting

\t tab
\n new line

Format controls reference: use man or visit

[The printf command](#)

```
#>printf "first name\tlast name\njack\t\tzheng\n"
```

```
first name    last name
```

```
jack          zheng
```

I/O Command: read

Accept user input from the keyboard and save it to a variable

Use -p option for input prompt

```
#> read -p "Enter something:" var1; echo "User input: $var1"  
Enter something: 1000  
User input: 1000
```

; to separate two commands.

User input will be saved to this variable

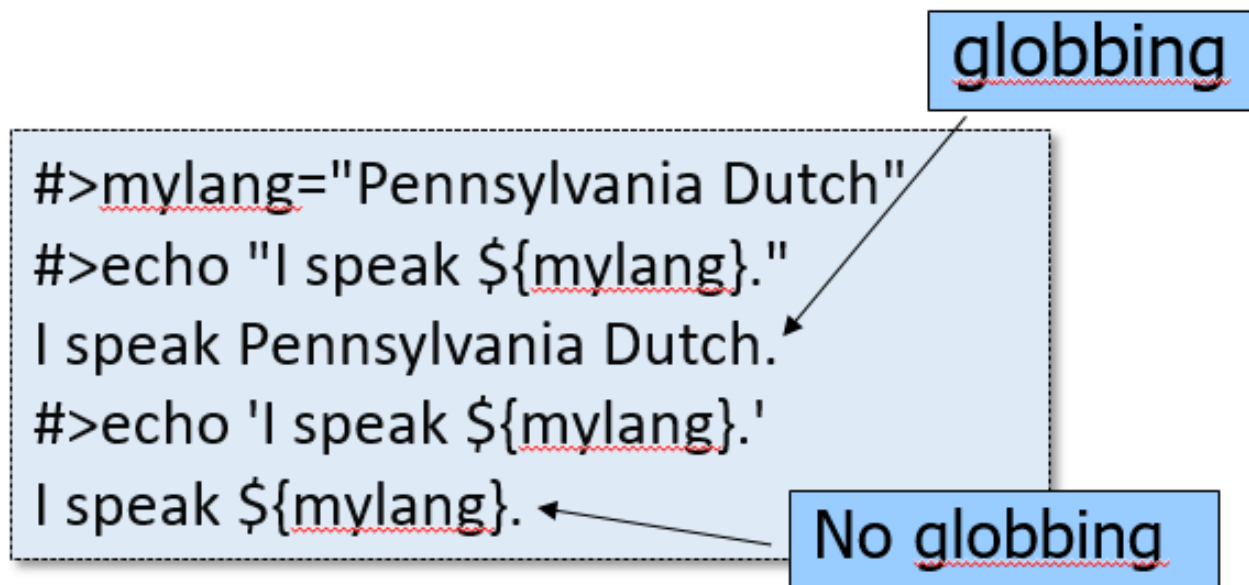
Double and Single Quotes

The shell treats strings enclosed in single and double quotes similarly, except that double-quoted strings are subject to globbing – a pattern matching behavior like:

The expansion of filename-matching metacharacters such as * and ?

Variable expansion \$

Command history !



Capture Command Output

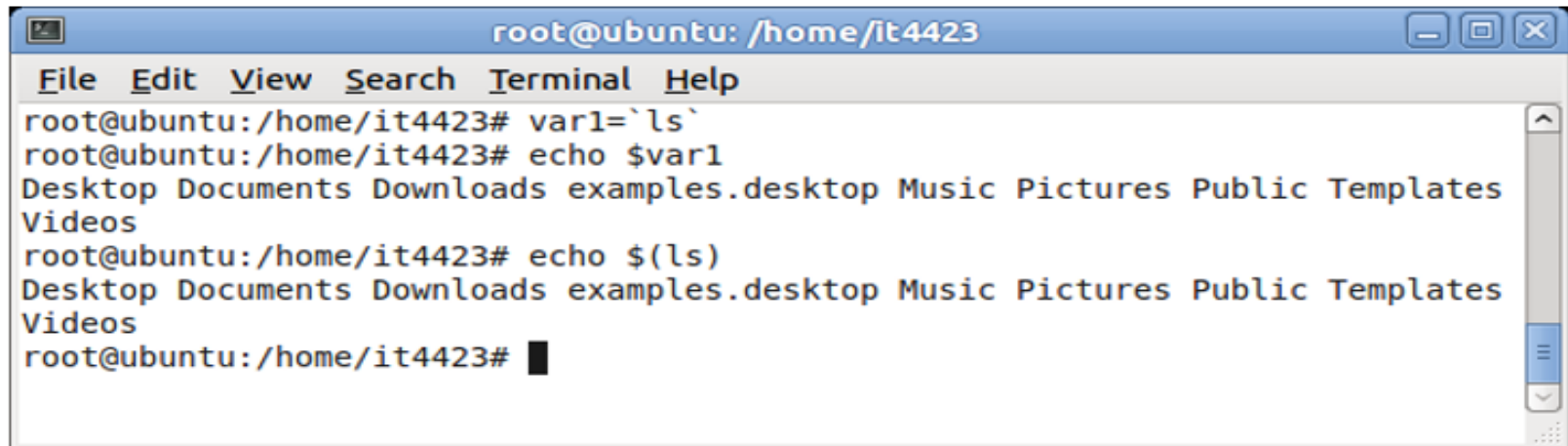
``

Back quotes (or back-ticks), are similar to double quotes, but they have the additional effect of executing the contents of the string as a shell command and replacing the string

```
#>echo "There are `wc -l /etc/passwd` lines in the passwd file."  
There are 28 lines in the passwd file.
```

\$()

Another form of command substitution



```
root@ubuntu: /home/it4423  
File Edit View Search Terminal Help  
root@ubuntu:/home/it4423# var1=`ls`  
root@ubuntu:/home/it4423# echo $var1  
Desktop Documents Downloads examples.desktop Music Pictures Public Templates  
Videos  
root@ubuntu:/home/it4423# echo $(ls)  
Desktop Documents Downloads examples.desktop Music Pictures Public Templates  
Videos  
root@ubuntu:/home/it4423#
```

Summary

Key concepts

Command-line scripting

I/O channels: stdin, stdout, stderr

Pipe

Environmental variable

Key skills: write simple command-line shell scripts utilizing the following elements

Channel operators: > < >> 2> |

Variables

Shell builtin commands

echo, printf, read, bash, alias

Commands: man, type, which, whereis

Good Readings and Resources

[Bash Guide for Beginners](#)

[15 Useful Bash Shell Built-in Commands](#)

Regular Expression for Searching

Regular Expression (Regex)

Looking for text patterns

Commonly used for string search

Use with “grep”

R`#> grep "http.*com" index.html`

Regex Quick Guide

Metacharacter

Meaning

[^]

^, \$

The ^ (circumflex or caret) **outside square brackets** means look only at the beginning of the target string.

The \$ (dollar) means look only at the end of the target string, for example, fox\$ will find a match in 'silver **fox**' since it appears at the end of the string but not in 'the fox jumped over the moon'.

.

The . (period) means any character(s) in this position, for example, **ton.** will find **tons**, **tone** and tonneau but not **wanton** because it has no following character.

[], a-z, 0-9, ^

range

|

* ? +

