

PenTest Class

Tutorial

Table of Contents

Table of Contents	2
Copyright	11
Introduction	12
Characters Meta Characters, Strings and Words	13
Chapter Overview	13
Chapter Prerequisites	13
Characters and Meta Characters	14
Topic Overview	14
Topic Objectives	14
Symbols and Characters	16
Symbol	16
Digits and Numbers	16
Character	17
Letter	17
Alphabet	17
Alpha Characters	17
Alphanumeric Characters	18
Case Sensitive	19
Uppercase Letters	19
Lowercase Letters	19
Storing Characters	20
Code Unit	20
Encoding	20
Code Point	20
Encoding Formats	21
Character Set	21
ASCII, Extended ASCII and EBCDIC	21

UTF	22
echo \$LANG	23
Meta Character	24
Literal Character	24
\$ as a Literal Character	24
Meta Character	24
\$ as a Bash Meta Character	24
\$ as a Regular Expression Meta Character	25
\	25
Strings, Delimiters and Words	26
Topic Overview	26
Topic Objectives	26
String	27
Delimiter	27
White Space Characters	28
IFS	28
Word	29
Command Line	29
Keyword	30
Reserved Word	30
Single Word Commands	31
Topic Overview	31
Topic Objectives	31
Command Word Examples	32
pwd	32
Absolute Path	32
cd	33
ls	34
cat	34

Read Files with cat	34
Concatenate Files with cat	35
Create Files with cat	35
echo	36
ps	37
date	37
uname	38
tty	38
Commands	39
Command Line Interpreter	39
Topic Overview	39
Topic Objectives	39
Shell	41
Shells as a User Interface	44
Shells as a Command Line Interpreter	44
Options, Format Controls and Arguments	45
Topic Overview	45
Topic Objectives	45
Options	46
- (short form option)	46
-- (long form option)	46
<space> (BSD option)	47
Applying Options	47
ls Command Options.	48

ls -l	48
ls -d	48
ls -i	48
ls -a	49
ls -n	49
ls -R	50
cat Command Options	50
cat -n	51
cat -s	51
echo	52
ps Command Options	53
ps -e	54
ps -a	54
ps a	54
ps -t	56
ps -f	56
ps -l	57
date Command Options	58
date	58
date -s	58
date -u	59
date -r	59
date math	59
uname Command Options	60

uname -s	60
uname -r	60
uname -v	60
uname -o	61
uname -p	61
uname -i	61
uname -n	61
Format Descriptors and Specifiers	62
Format Specifiers	62
ps	62
Format Descriptors	63
ps	63
date	63
Format Specifiers, Descriptors and Literal Characters	64
Arguments	65
Quoting and Command Substitution	66
Topic Overview	66
Topic Objectives	66
echo	68
Escape Sequences	68
Quoting	69
Using Single Quotes	69
Using Double Quotes	69
Embedding	70
Redirection	72
Lesson Overview	72
Lesson Objectives	72
stdin, stdout, stderr	74
stdin	74
stdout	74

stderr	74
File Descriptors	74
Viewing a Process File Descriptor Table	75
ls -l /proc/<pid>/fd	75
lsof -p	75
stdin	76
<	76
<<	77
stdout	78
stderr	79
2>&1	80
1>&2	81
>&	81
pipe	82
named pipe	82
unnamed pipe	82
tee	83
Regular Expressions	84
Lesson Overview	84
Lesson Objectives	84
Basic Regular Expression	85
Extended Regular Expressions	85
grep	86
grep -i	88
grep -v	89
grep -c	89
grep -n	89
grep -w	90

grep -f <filename>	90
grep -r or -R	91
grep -A #	91
egrep	92
fgrep	92
Regular Expression Metacharacters	94
Quantifiers	94
?	95
*	96
+	96
{n}	97
{n,}	97
{n,m}	98
Anchors	98
^<string>	99
<string>\$	99
. Period	100
Character Set	100
POSIX Character Sets	102
Vi	103
Lesson Overview	103
Lesson Objectives	103
Web References	103
Starting Vim	104
Syntax	104
Modes of Operation	105

Command Mode	105
Command line	105
Edit Mode	105
Insert	106
Append	106
Open	107
Cursor Movement	108
Cursor Word Movement	108
Cursor Line Movement	109
Manipulating Text	110
Deleting Text	110
Replacing Text	110
Copying Text	111
Moving Text	111
Undoing commands	112
Screen Positioning	112
Command Line Mode	113
Opening Documents from inside Vi	113
Saving and Exiting	113
Permissions	115
Symbolic Notation	116
Octal Notation	117
cp	118
Syntax	118
mv	120
Syntax	120
chmod	121

Symoblic Method	121
<permission set(s)>	121
<operator>	121
<permissions>	122
Octal Method	123
chown	124
Syntax	124
Glossary	126
Index	132

Copyright

This document is protected by both United States and International copyright law.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or any means electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's personal use without the written permission of Linux Courseware, LLC.

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in FAR 52.227-19 (Commercial Computer Software Restricted Rights) and DFAR 252.227-7013(c)(1)(ii) (Rights in Technical Data and Computer Software), and any other applicable FAR or DFAR regulation.

Introduction

This is a brief introduction to Linux terms and commands you may find helpful during the class.

This was not designed to be a Linux course and was put together in three days. I have removed information and rewritten some to make the tutorial smaller. If you find any mistakes I have missed please let me know.

I will post a Bash command line editor tutorial on Wednesday July, 6. I will give Lee the URL.

If you wish any other help or other course materials or have any questions or comments please contact me at sstrohmay@linuxcourseware.com.

You may download a CentOS 7 vdi file from the link below.

<https://ln.sync.com/dl/fd1354710#vzkeahbs-2fcp837p-8hb4zyck-s7isyz22>

Some Oracle Virtual Box tutorials are available here.

<https://linuxcourseware.com/support?view=kb&prodid=&catid=1>

Some of the tutorials are a little old , but still relevant.

I am creating a flash card set, and mind maps for the course so email if you wish a copy. They will be posted on my website and LMS when complete.

Please let me know if I can help further.

Sandor

Characters Meta Characters, Strings and Words

Chapter Overview

This chapter will explain:

- character types
- strings
- words

In addition we will introduce you to some basic commands.

Chapter Prerequisites

None

Characters and Meta Characters

Topic Overview

This topic will review:

- Character Types
- Case Sensitivity
- Character Encoding



The material in this topic will provide a reference to terms and concepts you will use during your study and administration of the Linux operating system.

Some of the terms we use to explain other concepts and terms, but have no direct relationship to Linux are provided for clarity.

Topic Objectives

- **Terms**
 - symbol
 - character
 - character types
 - alpha
 - numeric
 - digit
 - number
 - integer (data type)
 - alphanumeric
 - case sensitivity
 - upper case
 - lower case
 - encoding
 - meta characters

- **Commands**

- echo

- **Skills**





- Distinguish between a character, and meta character.
- View the current character set and encoding method.

Symbols and Characters

Symbol

A symbol is a written mark which represents an idea.

Table: Symbols

	International traffic symbol.
	Stop Sign
	No....
	No smoking.
\$	The dollar symbol may represents a unit of money
£	This symbol represents another unit of money (pound sterling)
a	This symbol represents a character which is present in many alphabets.

Digits and Numbers

Table: Number Types

Type	Definition
Digit	A digit refers to the symbols 0 through 9. The digits 1 and 0 are used to write the number 10.
Number	A number references a quantity. Whole numbers are the set of natural numbers (0,1,2,3...). Natural numbers do not include fractions.

Table: Number Types (continued)

Type	Definition
	Integers are the set of whole numbers and their negative values (-3,-2,-1, 0, 1, 2 3).
	Rational numbers are numbers that may be created by dividing two numbers (2/1, 3,4)
	Irrational numbers are numbers which cannot be expressed as a ratio (fraction). Pi, 3.1415926535..., is an irrational number.
	Real numbers are the set of whole numbers, rational numbers, irrational numbers, and their negative values.
Integer (Data Type)	A number may be stored as a character or an integer. Numbers stored as an integer are used in a mathematical expression.

Character

A character is a symbol which is part of a written language.

Letter

A letter is a character which represents a basic sound in an alphabet.

Alphabet

An alphabet is a set of letters, or symbols used to write a language.

Alpha Characters

Alpha characters are the set of:

- Upper and lower case characters
- Symbols
- Punctuation Marks

Alphanumeric Characters

Alphanumeric characters are the set of :

- Upper and lower case characters
- Numbers
- Symbols
- Punctuation Marks

Case Sensitive

Linux distinguishes between upper and lower case characters.

Uppercase Letters

Upper case characters are also referred to as capital letters.

Upper case characters are printed using the format A, B, C.

An example of a word printed in uppercase is TRUCK.

Lowercase Letters

Lower case characters are formatted a, b, c.

An example of a word printed in lower case is truck.



The terms upper case and lower case refer to a time when typesetting was done by hand, and letters were stored in angled stands. Capital letters (upper case) were stored above lower case letters.

Source: Oxford English Dictionary

The file names *filea*, and *FILEA* contain the same characters in the same positions, but in different case. Linux interprets the file names *filea* and *FILEA* as two different files.

Storing Characters

Computers store data as a binary code.

A unit of information defines capacity.

A bit (binary digit) is the smallest unit of information. A bit stores a value of 0 or 1.

An 8 bit unit of information may be called a byte or octet.

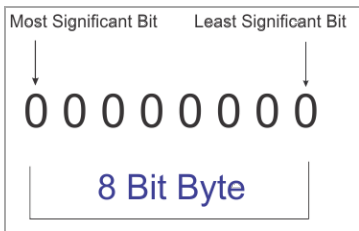


A byte is assumed to be an 8 bit unit of information.

The size of a byte may be hardware dependent.

For our purposes we will assume a byte is an eight bit unit of information.

The first digit on the left of a byte is called the most significant bit. The first digit on the right of a byte is called the least significant bit.



Code Unit

A code unit is the number of bits used to store a character.

The number of bits in a code unit is determined by the encoding format.

Encoding

Encoding maps a character to a unique number.

Code Point

A code point is the numerical value assigned (mapped) to a character.

Encoding Formats

Encoding formats determine how a character is stored.

Character Set

A character set is a defined list of characters.

The English alphabet character set contains 27 letters, 10 digits, and fourteen punctuation marks.

The ASCII character set contains 95 printable characters, 32 control codes and 1 space character.



We have included the ampersand (&) as a letter

ASCII, Extended ASCII and EBCDIC

Table: ASCII and EBCDIC Encoding Formats

Encoding Format	Description	Code Point (uppercase A)
ASCII	<p>The ASCII, American Standard Code for Information Interchange, character set uses a 7 bit encoding format.</p> <p>The most significant bit of the 8 bit byte is not used</p> <p>ASCII can support 128 characters.</p> <p> Extended ASCII uses 8 bit character encoding Extended ASCII can support 256 characters</p> <p>https://en.wikipedia.org/wiki/ASCII https://en.wikipedia.org/wiki/Extended_ASCII http://www.theasciicode.com.ar</p>	100 0001
www		
EBCDIC	<p>The EBCDIC, Extended Binary Coded Decimal Interchange Code, character set is an 8 bit (one octet or one byte) encoding format developed by IBM.</p>	1100 0001

Table: ASCII and EBCDIC Encoding Formats (continued)

Encoding Format	Description	Code Point (uppercase A)
www	EBCDIC can support 256 characters. https://en.wikipedia.org/wiki/EBCDIC	

UTF

ASCII and EBCDIC's code unit sizes cannot support the thousands of characters and symbols required by the world's alphabets.

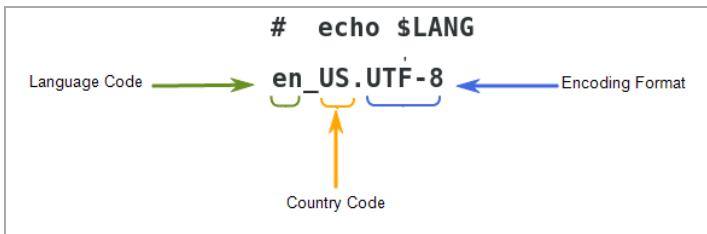
To create a code point for all characters and symbols, including Klingon, (<http://www.evertype.com/standards/csur/klingon.html>) an encoding format would have to support more than a one octet (byte) code unit. UTF, Unicode Transformation Format is designed to support one to four 8 bit code units

Table: UTF Encoding Formats

UTF	Supports one to four 8 bit code units.	
UTF-8	One 8 bit code unit.	0100 0001
UTF -16	One or two 16 bit code units	
UTF -32	One 32 bit code unit.	
www	http://unicode.org	

echo \$LANG

The command **echo \$LANG** displays the current language code, country code and encoding format.



A language code may contain a dialect code. (<language code> - <dialect code>)

pt is the language code for Portuguese

pt-PT language code Portuguese - dialect code Portugal

pt-BR language code Portuguese - dialect code Brazil

Meta Character

Literal Character

A literal character is interpreted based on its' assigned (code point) value.

\$ as a Literal Character

The dollar symbol may represent a unit of money.

The characters \$300.00 represents a monetary unit (three hundred dollars).

Meta Character

A meta character is a literal character which has a special interpretation applied.

An application, command line interpreter or regular expression engine interprets a meta character based on the context in which the meta character is applied.

\$ as a Bash Meta Character

If a dollar symbol (\$) precedes a character or series of characters) the Bash command line interpreter applies variable expansion.



Variable expansion is the process of substituting the value of a variable with a word (or word list) value.

The variable SHELL stores the absolute path to the default logon shell assigned to a user.

In the example below the meta character \$ instructs the command line interpreter to apply variable expansion and display the value of the variable SHELL.

```
# echo $SHELL  
/bin/bash
```


\$ as a Regular Expression Meta Character

A dollar symbol (\$) following a string of characters instructs the basic or extended regular expression engine to search for the preceding string of characters at the end of a line.

The command **grep bash\$ /etc/passwd** will search for the string bash at the end of each line in the file */etc/passwd*.

```
# grep bash$ /etc/passwd
root:x:0:0:root:/root:/bin/bash
student1:x:1000:1000:Student1:/home/student1:/bin/bash
```

\

A backslash (\) is a Bash meta character which instructs the command line interpreter to ignore (escape) the special meaning of the next character.

The output of the command **echo \$SHELL** below illustrates Bash interpreting the string SHELL as a variable name and replaces the variable name with the value of the variable.

```
# echo $SHELL
/bin/bash
```

The output of the command **echo \SHELL** below illustrates Bash ignoring the special interpretation of the dollar symbol. Notice the output displays the literal character (\$).

```
# echo \SHELL
$SHELL
```

Strings, Delimiters and Words

Topic Overview

This topic will introduce you to:

- strings
- words

Topic Objectives

- **Terms**
 - string
 - delimiter
 - IFS
 - word
 - reserved word
 - keyword
 - command
- **Commands**
 - echo
- **Skills**
 - Distinguish between a string and word.

String

A string is a series of characters.

```
string123xaA-~1
```

The sentence below contains two instances of the string break.

```
Our break for breakfast is at 07:00
```

The `grep` command is used to search for strings. The command `grep break stringtest` below searches for the string of characters break in the file *stringtest*.

```
# grep break stringtest
Our break for breakfast is at 07:00
```

Delimiter

A delimiter is a character used to establish a separation (data boundary) between strings or independent data fields.

The colon (:) and comma (,) are commonly used as data boundaries between fields in a flat file database.



A flat file database is a database stored in a text file.

Each line of the database file is a record.

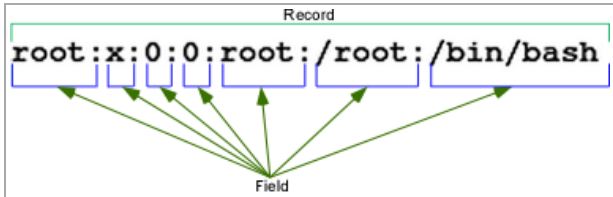
A record is a unit of information called a which may contain one or more associated data fields

A field is a single unit of information associated with a database record

The file `/etc/passwd` is a flat file database which contains user records.

Each user record is divided into seven fields.

The illustration below displays the record for the user `root` in the file `/etc/passwd`.



White Space Characters

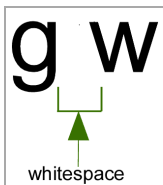
A white space character occupies space but is not visible.

White space characters are used as command line delimiters.

Examples of whitespace characters are:

- space character (ASCII 32)
- tab character (ASCII 9)
- newline character (ASCII 10)

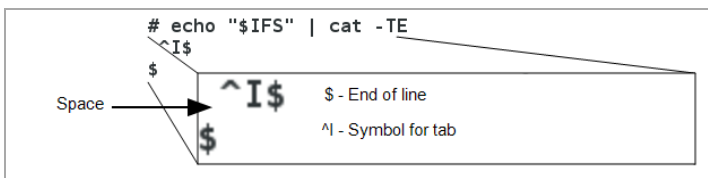
In the illustration below a space, ASCII 32, is used as a delimiter between two characters.



IFS

IFS, Internal Field Separator is an environmental variable which stores the character or characters used as delimiters.

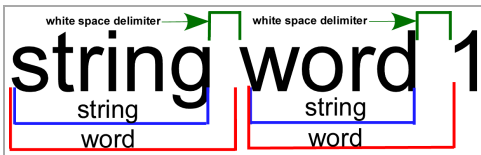
The command `echo "$IFS" | cat -TE` will display the current delimiters.



Word

To process a command line the Bash command line interpreter breaks the command line into word (tokens) and determines how each word must be applied.

A word is a printable character or string terminated by a delimiter.



The command **grep -w break stringtest** below will search for the word break in the file *stringtest*.

```
# grep -w break stringtest
Our break for breakfast is at 07:00.
```

Command Line

A command line may consist of multiple words delimited by white space characters.

ls is a command which lists the names of files in the current working directory. The word **ls** command is the first word in the command line. This is the command.

The second word, **-ld**, contains two options. The **l** option displays a file's properties, and the **d** option limits the output to files which are directories.

The third word, */etc*, is an argument.

```
# ls_-ld_/etc
drwxr-xr-x 99 root root 12288 Nov  6 10:12 /etc
```

The command **ls -ld /etc** will display the properties of the directory */etc*.

Keyword

A keyword is a word which has special meaning to a program or operating system, which may be used as an identifier (e.g. variable name).

Reserved Word

A reserved word has special meaning to a program or operating system, which may not be used as an identifier.

Single Word Commands

Topic Overview

This topic will introduce you to a few Linux command words.

Topic Objectives

- **Terms**

- absolute path
- node

- **Commands**

- pwd
- cd
- ls
- cat
- echo
- ps
- date
- uname
- tty

- **Skills**

- learn the application of the commands listed above.

Command Word Examples

pwd

The command `pwd`, print working directory, displays the absolute path to the current working directory.

Command Type	Builtin and external
Permissions	Available to all users
Documentation	man and info pages, help pwd



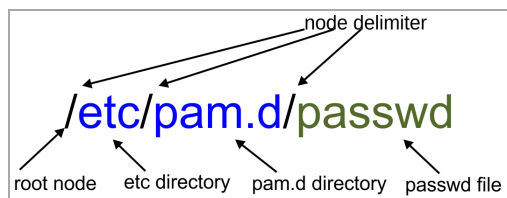
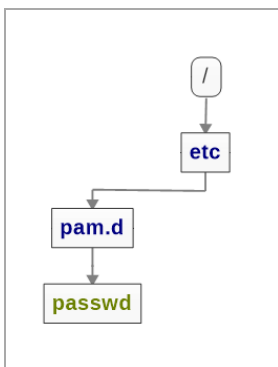
The current working directory is the directory which the user is working in.

Absolute Path

The absolute path specifies the path to a file from the system's root (`/`) directory.

The diagrams below show the absolute path from the root directory to the file `/etc/pam.d/passwd`.

A node is a link or connection point.



cd

Command Type	Builtin and external
Permissions	Available to all users
Documentation	man and info pages, help cd

The **cd**, change directory, command is used to change the current working directory.

If the change directory command, **cd**, is executed by itself, the current working directory is changed to the user's home directory.



To verify you have changed to the correct directory execute the **pwd** command after the **cd** command.

In the illustration below the current working directory is */etc* (line 2), and the user executing the commands is root.

The user root's home directory is */root*.

Notice when user root executes the **cd** command (line 4) the current working directory changes from */etc* to */root* (line 7).

The user root's home directory is */root*.

```
1 # pwd
2 /etc
3 #
4 # cd
5 #
6 # pwd
7 /root
```

ls

Command Type	External
Permissions	There are multiple default ls command aliases. To view them execute the command alias grep ls . Available to all users Users must have read and execute permissions for a directory to view the property of files in the directory
Documentation	man and info pages

The **ls** command will list the names of all the files (except hidden files) in the current directory.



A hidden file is any file whose file name is prefixed by a period (.) e.g. `.bashrc`.

Files are hidden to protect them from accidental modification by a user.

```
# ls
filea fileb filec
```

cat

The **cat** command is used to read, concatenate or create files.

Command Type	External
Permissions	Available to all users Users must have permissions to the files they are accessing. The cat command may also be affected by the shell parameter noclobber .
Documentation	man and info pages

Read Files with cat

The command **cat /etc/hosts** will read and display the contents of the file `/etc/hosts`.

```
# cat /etc/hosts
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6
```

Concatenate Files with cat

Concatenate means to link together.

The command **cat filea fileb** (line 7) will read and display the contents of *filea* and *fileb*.

```
1 # cat filea
2 this is filea
3 #
4 # cat fileb
5 this is fileb
6 #
7 # cat filea fileb
8 this is filea
9 this is fileb
```

Create Files with cat

The command **cat > <file name>** will create a file.

The command **cat > filea** followed by the key sequence Ctrl +d will create the file *filea*.

```
# cat > filea
█
Ctrl + d
```



The key sequence Ctrl +d issues an EOF and exits the process.

EOF, end of file, tells the operating system no more data is available from the data source.

echo

Command Type	Builtin and External
Permissions	Available to all users
Documentation	man and info pages , help echo

The echo command displays the argument or arguments to the right of the command to the standard output device.

The command **echo hello world** will display the output hello world.

```
# echo hello world
hello world
```

The variable \$HOME store the user's home directory. The command echo \$HOME will expand the variable \$HOME and print its' value.

```
# echo $HOME
/root
```

ps

Command Type	External
Permissions	Available to all users
Documentation	man and info pages


The **ps**, process status, command displays the status of processes executing in the current terminal at the time the ps command is executed.

```
# ps
  PID TTY          TIME CMD
 2979 pts/1        00:00:00 bash
 3206 pts/1        00:00:00 ps
```

Table: ps Output Columns

Column	Definition
PID	Process id
TTY	Terminal process is executing on.
TIME	Cummalitve run time - hours:minutes:seconds
CMD	Executing command (options and arguments are not displayed)

date

Command Type	External
Permissions	Available to all users
Documentation	man and info pages  Info pages contain more information than man pages.

The **date** command displays the current day, month, date, time (based on a 24-hour clock), time zone, and year.

```
# date
Thu Nov  6 23:02:00 EST 2014
```

uname

Command Type	External
Permissions	Available to all users
Documentation	man and info pages

The **uname** command will display the name of the operating system.

```
$ uname
Linux
```

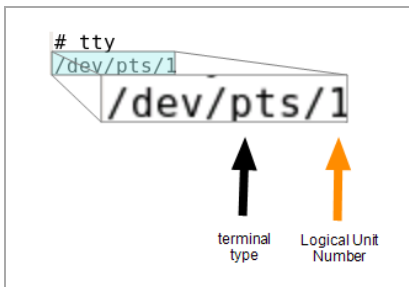
tty

Command Type	External
Permissions	Available to all users
Documentation	man and info pages

The **tty** command displays the device name of the current terminal.

The output below is for a pseudo terminal.

```
# tty
/dev/pts/1
/dev/pts/1
```



terminal type Logical Unit Number

Commands

A command line interface, CLI, is a text based interface used to communicate with the operating system via a terminal device.

A shell accepts and interprets user or application input for the operating system, and receives and processes output from the operating system. A shell may also be called a command line interpreter or command line processor.

Each shell has a set of grammatical rules called syntax. Syntax rules determine the structure of a command line. Syntax rules for a command may be found in the command's man page.

Semantics are a set of rules which determine what operations a shell may perform, and the order in which they are performed.

The following chapter will discuss:

- command components
- basic syntax rules.
- command precedence
- command execution
- redirection

Command Line Interpreter

Topic Overview

A shell functions as a user and application interface as well as a command line interpreter.

This topic will briefly define these concepts.

Topic Objectives

- **Terms**
 - command line interpreter
 - shell
 - syntax

- semantics
- */etc/shells*
- **Commands**

- **Skills**
 - Create a command
 - Troubleshoot a command line.

Shell

A shell is a program which provides a user or application interface with the operating system.

Table: Shells








Shell	Description
Tsh	The Thompson shell was the original UNIX shell. It was designed as a user interface and had no scripting capabilities.
Bourne	<p>The Bourne shell is a command line interpreter and scripting language developed by Stephen Bourne to replace the Thompson shell as the default shell for UNIX version 7.</p> <p>The Bourne Shell functioned as a command line interpreter and shell scripting language .</p> <p>The command, /bin/sh opens a Bourne shell.</p> <p> In Linux environments, the Bourne shell operates as a subset of the Bash shell. /bin/sh is a symbolic link to /bin/bash.</p> <p> https://en.wikipedia.org/wiki/Bourne</p>
C	<p>The C shell is a command line interpreter and scripting language written by Bill Joy of the University of California at Berkeley.</p> <p>The C shell appeals to C language programmers because its' syntax is similar to the C programming language. It also offers command line history and recall, job control and aliases.</p> <p> https://en.wikipedia.org/wiki/C_shell</p>
Korn	David Korn developed the Korn Shell

Table: Shells (continued)

Shell	Description
	<p>The Korn shell combined the attributes of both the Bourne and C shells as well as adding additional capabilities but was only available through commercial license until the development of alternate open source versions like the Public Domain Korn Shell, pdksh, and MirBSD Korn Shell, mksh. mksh is the most current version.</p>
	<p>http://www.kornshell.com https://en.wikipedia.org/wiki/KornShell</p> <p>http://www.mirbsd.org/mksh.htm</p> <p>web.cs.mun.ca/~michael/pdksh/</p>
Tcsh	<p>The TENEX C Shell, tcsh, is an enhanced compatible version of the Berkeley UNIX C shell.</p>
	<p>The original website, tcsh.org, is no longer available, but an archive exists here: https://web.archive.org/web/20170609182511/http://www.tcsh.org:80/Home https://en.wikipedia.org/wiki/Tcsh</p>
Bash	<p>The Bash, Bourne Again Shell, (http://www.gnu.org/software/bash/) is a super set of the Bourne shell written by Brian Fox as a replacement for the Bourne shell.</p>
	<p>https://www.gnu.org/software/bash/ https://www.gnu.org/software/bash/manual/ https://en.wikipedia.org/wiki/Bash_(Unix_shell)</p>
Dash	<p>Dash, Debian Almquist shell, is the replacement shell for the Ash, Almquist shell, on both Debian and Ubuntu. Dash requires less memory, less disk space, fewer libraries, and is faster than the Bash shell.</p>
Z	<p>The Z shell is an extension of the Bourne shell which contains many more capabilities (including an FTP client and TCP controls).</p>
	<p>http://zsh.sourceforge.net</p>



The file `/etc/shells` contains a list of login shells.

The command **chsh -l** will list the contents of */etc/shells* (login shells).

The commands **chsh -s <absolute_path_to_shell> <user name>** may be used to permanently change a user's login shell.

```
# chsh -s /bin/tcsh user1
Changing shell for user1.
Shell changed.
```

Shells as a User Interface

A shell provides a user's work environment.

The operation of a shell is controlled via generic and user specific configuration files. Generic files may be found in the `/etc` directory and use specific files may be found in the user's home directory. The number of configuration files is shell dependent.

Two main configuration files are user environment and run control. The user environment configuration file uses variables to configure the tools available to the user. The run control file is used to alter the operation of the shell.

Table: : Environment and Run Control Shell Configuration Files (User)

Shell	Environment	Run Control
Bash	<code>..bash_profile</code>	<code>.bashrc</code>
Tcsh	<code>.login</code>	<code>..tschrc</code>
Zsh	<code>.zshenv</code> , <code>..zprofile</code>	<code>.zshrc</code>
Korn	<code>.profile</code>	<code>.kshrc</code>

Shells as a Command Line Interpreter

A command line interpreter, CLI, is a program which evaluates and executes commands.

As a command line interpreter the shell reads input from the keyboard, file or device and parses the input based on the shell's syntax rules.



Syntax rules evaluate the elements of a command line to determine if they are valid and presented in the proper format. and order.

Once the command line is parsed the shell executes the command, and presents the output to the user's terminal, file or device.

The Bash shell is the default command line interpreter and login shell for most Linux distributions.

Options, Format Controls and Arguments

Topic Overview

A command line contains

Topic Objectives

- **Terms**

- option
- argument
- format specifier
- quoting
- embedding

- **Commands**

- ls
- cat
- ps
- date
- uname

Options

A command option controls how a command operates.

- (short form option)

Historically UNIX preceded options with a tac symbol (-).



Original versions of UNIX did not use a tac (-) symbol for options.

A tac symbol (-) preceding an option in Linux directs the command line interpreter to process one character at a time.

```
# ls -lh
total 8.0K
-rw-r--r--. 1 root root 158 May 17 09:55 hosts
-rw-r--r--. 1 root root 2.5K May 17 09:55 passwd
```

-- (long form option)

The double tac (--) facilitates a GNU naming convention designed to make option designations more understandable.

A double tac preceding an option directs the command line interpreter to process a word at a time.

```
# ls -l --human-readable
total 8.0K
-rw-r--r--. 1 root root 158 May 17 09:55 hosts
-rw-r--r--. 1 root root 2.5K May 17 09:55 passwd
```

The GNU option **--help** will provide a brief help document for most commands.

```
$ shutdown --help
shutdown [OPTIONS...] [TIME] [WALL...]

Shut down the system.

  --help      Show this help
  -H --halt   Halt the machine
  -P --poweroff Power-off the machine
  -r --reboot  Reboot the machine
  -h          Equivalent to --poweroff, overridden by --halt
  -k          Don't halt/power-off/reboot, just send warnings
  --no-wall   Don't send wall message before halt/power-off/reboot
  -c         Cancel a pending shutdown
```

<space> (BSD option)

Original UNIX versions did not precede an option with a tac symbol.

When the University of Berkley developd BSD UNIX they decided to maintain the same format.

In order to provide backwards compatibility Linux supports both UNIX and BSD option types.

Both FreeBSD and OpenBSD support the short form option.

Notice the difference in output between the **ps a** (line 1) and **ps -a** (line 8) commands.

```

1 ps a
2 PID TTY          STAT TIME COMMAND
3 1203 tty1      SsL+ 0:11 /usr/bin/X :0 -background none -noreset -audit 4 -verbose -auth /run/gdm/auth-for-gdm-9AKFXw/database -seat seat0 -nolisten tcp vt1
4 4999 pts/0      Ss    0:00 bash
5 5040 pts/0      R+    0:00 ps a
6 #
7 # ps -a
8 # PID TTY          STAT TIME CMD
9 5061 pts/0      00:00 ps

```

Applying Options

A single option may be applied

```
# uname -r
3.10.0-327.22.2.el7.x86_64
```

Multiple options may be specified as separate words.

```
# uname -n -r
localhost.localdomain 2.6.32-431.el6.i686
```

Multiple options may be or grouped as one word.

```
# uname -nr
localhost.localdomain 2.6.32-431.20.3.el6.i686
```

ls Command Options.

The ls command lists the files and directories in the current working directory.

Command Type	External
Permissions	There are multiple ls command aliases. To view them execute the command alias grep ls . Available to all users Users must have read and execute permissions for a directory to view the property of files in the directory
Documentation	man and info pages, help pwd

ls -l

The option **-l** will list the names and properties of all files (except hidden files) in the current directory.

2250	-rw-r--r--	2	root	root	15	Sep 18 08:10	filea
↓	↓	↓	↓	↓	↓	↓	↓
inode number	file type permissions	# hard links	owner	group owner	file size	modification time and date	filename

```
# ls -l
total 68
-rw----- 1 root root 1322 Mar  5 14:07 anaconda-ks.cfg
drwxr-xr-x 2 root root 4096 Mar  5 14:13 Desktop
-rw-r--r-- 1 root root 34305 Mar  5 14:07 install.log
-rw-r--r-- 1 root root 5401 Mar  5 14:05 install.log.syslog
```

ls -d

The **-d** option limits the output to directories.

```
# ls -ld
drwxr-x--- 14 root root 4096 Mar 24 12:24 .
```

ls -i

The **-i** option will display a file's inode number and file name.


```
# ls -i /etc/passwd
148184 /etc/passwd
```



An inode is a file system data structure which stores a file's ownership, time stamp, and data location information.

ls -a

The **-a** option will display all files (including hidden files) in the current directory.



A hidden file is any file whose file name is prefixed by a period (.) e.g. .bashrc.

Files are hidden to protect them from accidental modification by a user.

```
# ls -a
.          Desktop      .gnome2_private  .nautilus
..         .dmrc       .gstreamer-0.10  .redhat
anaconda-ks.cfg .eggccups   .gtkrc-1.2-gnome2 .tcshrc
.bash_history .esd_auth   .history          .Trash
.bash_logout .gconf      .ICEauthority     .xsession-errors
.bash_profile .gconfd     install.log
.bashrc      .gnome      install.log.syslog
.cshrc       .gnome2     .metacity
```

ls -n

The **-n** option will display the file owner's user id rather than the user name and group id rather than the group name .

The command **ls -ln** produced the output below.

```
-rw-----, 1 0 0 1997 Apr 19 21:51 anaconda-ks.cfg
-rw-----, 1 0 0 2045 Apr 19 22:31 initial-setup-ks.cfg
```

ls -R

The **-R** option displays all of the files in the directory specified in the argument, and descends into any sub-directories and displays all the files in the directorates. This is called recursive.

```
# ls -lR dirtest
dirtest:
total 4
drwxr-xr-x. 2 root root 1024 Jul  5 11:40 dirtest2
-rw-r--r--. 1 root root   0 Jul  5 11:39 filea
-rw-r--r--. 1 root root   0 Jul  5 11:39 fileb

dirtest/dirtest2:
total 2
-rw-r--r--. 1 root root 0 Jul  5 11:40 file1
-rw-r--r--. 1 root root 0 Jul  5 11:40 file2
```

cat Command Options

The **cat** command reads each line of a file from the top of the file to the bottom of the file, and outputs the contents to the standard output device. By default the standard output device is the user's terminal.

Command Type	External
Permissions	Available to all users Users must have permissions to the files they are accessing. The cat command may also be affected by the shell parameter noclobber .
Documentation	man and info pages

The illustration below uses the **cat** command to display the contents of the file **cattest**.

```
1 # cat cattest
2 this   line   is       tabbed
3
4
5
6 this line is spaced
```

cat -n

The **-n** option numbers the lines.

```

1 # cat -n catest
2   1 this line is tabbed
3   2
4   3
5   4
6   5 this line is spaced

```

cat -s

The **-s** option removes (squeezes) multiple blank lines.

```

1 # cat -n catest
2   1 this line is tabbed
3   2
4   3
5   4
6   5 this line is spaced

```

```

1 # cat -ns catest
2   1 this line is tabbed
3   2
4   3 this line is spaced

```

The **cat** command may contain multiple arguments.

In the illustration below the **cat** command will concatenate (link) *filea* and *fileb* and send the output to the standard output device.

```

1 # cat filea
2 this is filea
3 #
4 # cat fileb
5 this is fileb
6 #
7 # cat filea fileb
8 this is filea
9 this is fileb

```

The **tac** command displays the lines of a file in sequence from the bottom of a file to the top of a file.

```

1 # tac catest
2
3 this line is spaced
4
5
6
7 this line is tabbed

```

echo

The **echo** command prints the command's argument to the standard output device.

Command Type	Builtin and External
Permissions	Available to all users
Documentation	man and info pages , help echo

Some options may not be available in all shells. Please refer to the shell man pages.

<u>Option</u>	<u>Argument</u>	<u>Description</u>
-e		enable backslash escape characters
	\n	new line
	\r	carriage return
	\e	escape
-E		disable backslash escape characters (default)
-n		do not output new line

The example below illustrates the **echo** command displaying its' argument to the standard output device.

```
# echo Hello
Hello
```

The escape sequence **\t** inserts a tab into the output.

```
# echo -e "The default shell is: \t $SHELL"
The default shell is:  /bin/bash
```

ps Command Options

The **ps** command will display the status of processes executing in the current terminal at the time the ps command is executed.

Command Type	External
Permissions	Available to all users
Documentation	man and info pages

```
# ps
  PID TTY          TIME CMD
 2979 pts/1        00:00:00 bash
 3206 pts/1        00:00:00 ps
```

Command Type	External
Permissions	Available to all users
Documentation Column	man and info pages Definition
PID	Process id
TTY	Process is executing on this terminal A question mark (?) in this column indicates a process which is not attached to a terminal (system process).
TIME	Cumulative run time - hours:minutes:seconds
CMD	Executing command (options and arguments are not displayed)

ps -e

The command **ps -e** displays every process

The illustration below displays the edited results from the **ps -e** command.

```

1 3985 ?          00:00:00 login
2 4023 tty2      00:00:00 bash
3 4081 tty2      00:00:00 vim
4 4084 ?          00:00:00 login
5 4114 tty3      00:00:00 bash
6 6804 ?          00:00:01 gnome-terminal-
7 6807 ?          00:00:00 gnome-pty-helpe
8 6808 pts/0      00:00:00 bash
9 6849 pts/1      00:00:00 bash

```

Output Edited

ps -a

The **-a** option will display all executing processes which are attached (executing from) a terminal in the current login session.

```

# ps -a
  PID TTY          TIME CMD
 9323 pts/2      00:00:00 vi
 9327 pts/1      00:00:00 ps

```

ps a

The command **ps a** displays all processes executed by the user in the current login session.



The **--sort=tty** option sorts processes by terminal.

```

$ ps a --sort=tty
  PID TTY          STAT      TIME COMMAND
 2381 tty1      Ss+      0:39 /usr/bin/Xorg :0 -background none -verbose -auth /run/gdm/auth-for-gdm-Y5
12369 tty2      Ss        0:00 -bash
12585 tty2      S+        0:00 vim
12465 tty3      Ss        0:00 -bash
12610 tty3      S+        0:00 sleep 10000
 3339 pts/0      Ss        0:00 /bin/bash
11712 pts/0      S+        0:00 ssh localhost -l root
12870 pts/1      Ss        0:00 /bin/bash
12920 pts/1      S          0:00 su -
12927 pts/1      S          0:00 -bash
12957 pts/1      S+        0:00 vim
 9809 pts/2      Ss        0:00 /bin/bash
13209 pts/2      R+        0:00 ps a --sort=tty
11719 pts/5      Ss+      0:00 -bash

```

Table: ps STAT command column

Column	Entry	Definition
STAT or S		Displays the state of a process
	R	Process is running or is able to run (active)  An active process is ready to run, but not running. The entry for this type of process may be the letter A
	S	Interruptable Sleep Waiting for a process to complete. Example waiting for a child process to complete.
	D	Uninterruptable Sleep Example - Waiting for I/O
	T	Stopped Process stopped by a control signal or trace. Example - Using Ctrl+z to suspend a job.
	Z	Zombie Child process which has terminated but still contains an entry in the process table.
	+	In foreground process group
	s	Session leader  Processes are organized into sets called sessions. Each session has a session ID. The session ID is the process id number of the process which created the session. That process is called the session leader.
CMD		Command being executed
	-bash	bash is being executed as a login shell

ps -t

The command **ps -t <device_name>** displays the properties of all processes attached to a specific terminal.

```
# ps -t /dev/pts/1
  PID TTY          TIME CMD
 3846 pts/1        00:00:00 bash
 3876 pts/1        00:00:00 vim
```

ps -f

Adding the **-f** or **-l** options to the **ps** command displays additional process properties.

```
UID      PID  PPID  C  STIME TTY          TIME CMD
student  3043 3042  0  18:51 pts/3        00:00:00 -bash
student  3075 3043  0  18:52 pts/3        00:00:00 vim
```

Table: ps UID, PPID, C and STIME command column

Column	Entry	Definition
UID		User executing process
PPID		Parent Process ID
C		CPU usage
STIME		Start time of process

```
UID      PID  PPID  C  STIME TTY          TIME CMD
student  3043 3042  0  18:51 pts/3        00:00:00 -bash
student  3075 3043  0  18:52 pts/3        00:00:00 vim
```


ps -l

Table: ps F, PRI, AND NI columns

Column	Entry	Definition
F		Flag
	0	No flags applied
	1	Process has forked but has not executed (active).
	4	Running as root
	5	Both flags 1 and 4 applied
PRI		Process Priority
NI		Nice value assigned to process.
ADDR		Process memory address.
SZ		Virtual Memory size.
WCHAN		The kernel address in which the current task is waiting for a resource (keyboard, mouse), data or processing time.
CMD		Command being executed.


```

$ ps -l
F S  UID  PID  PPID  C  PRI  NI  ADDR  SZ  WCHAN  TTY  TIME  CMD
0 S  1000  3339  3335  0  80   0  - 29032  wait  pts/0  00:00:00  bash
0 R  1000  9914  3339  0  80   0  - 30315  -    pts/0  00:00:00  ps

```

date Command Options

The date command displays the current day, month date, time (based on a 24 hour clock), timezone and year.

Command Type	External
Permissions	Available to all users
Documentation	man and info pages  Info page contains more information than man pages.

```
# date
Thu Nov 6 23:02:00 EST 2014
```

date -s

A user may set the current date by executing the command **date -s "<date-string>"**. The date string must be enclosed in quotes and contain:

Day Month Date HH:MM:SS TZ Year



The time portion of the string is expressed using a 24 hour clock.

HH - hour **MM** - minute **SS** - second **TZ** - Time Zone

In the example below the current date is displayed on line 2. The date is changed using the command **date -s "Mon June 27 09:00:00 EDT 2016"** on line 4.

```
1 # date
2 Wed Jun 22 14:56:06 EDT 2016
3 #
4 # date -s "Mon June 27 09:00:00 EDT 2016"
5 Mon Jun 27 09:00:00 EDT 2016
6 #
7 # date
8 Mon Jun 27 09:00:03 EDT 2016
```

date -u

The command **date -u** will display the Universal Coordinated time.

```
Mon Jun 27 13:14:01 UTC 2016
```

date -r

The **date -r <file_name>** command will display the time the contents of the file specified by the argument **<file_name>** were modified

In the illustration below line 2 displays the properties of the file */etc/passwd*. The time displayed is the last time the contents of the file was changed (modification time).

The **touch** command on line 4 changes the modification time of the file */etc/passwd* to the current time. The new modification time is displayed by the **ls -l** command on line 6. The **date -r** command on line 8 only displays a file's last modification date and time,

```
1 # ls -l /etc/passwd
2 -rw-r--r--. 1 root root 2382 Jun  7 12:15 /etc/passwd
3 #
4 # touch /etc/passwd
5 # ls -l /etc/passwd
6 -rw-r--r--. 1 root root 2382 Jun 27 09:11 /etc/passwd
7 #
8 # date -r /etc/passwd
9 Mon Jun 27 09:11:55 EDT 2016
```

date math

The command **date --date <time unit>** displays a date in the past or future. This command will not change the current date.

The command **date --date="2 days ago"** will display date information.

```
1 # date
2 Mon Jun 27 09:07:18 EDT 2016
3 #
4 # date --date="2 days ago"
5 Sat Jun 25 09:07:25 EDT 2016
6 #
7 # date
8 Mon Jun 27 09:07:27 EDT 2016
```

uname Command Options

The `uname` command prints system information.

```
$ uname
Linux
```

Command Type	External
Permissions	Available to all users
Documentation	man and info pages

uname -s

`uname -s` prints the kernel name

```
# uname -s
Linux
```

uname -r

The `uname -r` command will print the kernel version, operating system release number, and architecture.

```
# uname -r
3.10.0-327.22.2.el7.x86_64
```

uname -v

The command `uname -v` will display the type of kernel and the date and time the kernel was compiled.

```
$ uname -v
#1 SMP Thu May 12 11:03:55 UTC 2016
```

`uname -o`

`uname -o` prints the operating system name

```
# uname -o
GNU/Linux
```

`uname -p`

`uname -p` prints the processor type.

```
# uname -p
x86_64
```

`uname -i`

`uname -i` prints the hardware platform type.

```
# uname -i
x86_64
```

`uname -n`

The command `uname -n` displays the system node name (host name) and domain name.

```
$ uname -n
localhost.localdomain
```

Format Descriptors and Specifiers

A format descriptor or format specifier allows a user to determine what data is displayed, and how the data appears in a command's output.

- Not all commands support format specifiers or format descriptors.
- Most commands precede the format specifiers or descriptors by an option.
- A format descriptor is preceded by a percentage symbol (%).

Format Specifiers

In the examples below we use format descriptors to define which data output.

ps

Table: ps format specifier examples

Specifier	Definition
uid	User id
cmd	Command
tty	Terminal
pid	Process id
parent process id	Parent process id

```
# ps -U student1 -o uid,cmd,TTY,pid,ppid
  UID  CMD          TTY          PID  PPID
 1000 -bash          pts/1        8990  8989
 1000 vim          pts/1        9026  8990
```

Format Descriptors

In the example below we use format descriptors to define which data output

ps

Table: ps format descriptor examples

Descriptor	Definition
%U	Process executing as real or effective user id
%c	Command
%y	Terminal
%p	Process id
%P	Parent process id

```
# ps -U student1 -o %U%c%y%p%P
USER      COMMAND      TTY          PID    PPID
student1  bash         pts/1       8990   8989
student1  vim         pts/1       9026   8990
```

date

The **date** command uses format descriptors to control the output.

The date command requires a plus symbol (+) precede output formatting characters.

Table: date format descriptor examples

Descriptor	Definition
%m	Process executing as real or effective user id
%y	Command

```
1 # date
2 Mon Aug 15 12:50:53 EDT 2016
3 #
4 # date +%m%d%y
5 081516
```

Format Specifiers, Descriptors and Literal Characters

The example below combines format descriptor characters (m, d, and y), and literal character colon (:) to form a colon delimited output.

```
$ date +%m:%d:%y  
09:16:15
```

Changing the literal character from a colon (:) to a forward slash (/) changes the look of the output.

```
$ date +%m/%d/%y  
01/30/16
```

Below we have created an output which contains text and format specifiers.

```
$ date +"The date is %d %B %Y"  
The date is 28 January 2016
```


Arguments

An argument is an optional command line component.

An argument is a value passed to a command as a data source.

The command `ls -ld /etc` below will display the properties of the directory specified by the command's argument, */etc*.

```
# ls -ld /etc
drwxr-xr-x. 141 root root 9216 Aug 15 11:10 /etc
```



The data source is the inode for the directory */etc*.

The command `echo` in the illustration below uses the value of the shell variable `?` (exit status code) as its argument to display the exit status code of the previous command.

```
# echo $?
0
```

Commands may contain multiple arguments.

```
# ls -ld /etc /usr
drwxr-xr-x. 141 root root 9216 Aug 15 11:10 /etc
drwxr-xr-x. 14 root root 4096 Jun 26 03:49 /usr
```

In the illustration below the `cat` command on line 7 concatenates the files *filea* and the file *fileb*.

```
1 # cat filea
2 this is filea
3 #
4 # cat fileb
5 this is fileb
6 #
7 # cat filea fileb
8 this is filea
9 this is fileb
```

An argument may define the data source for a specific option.

In the command below the **-U** option refines the output of the **ps** command to display only those processes which are being executed by a user with the real user id of student.



The username entered as part of the login process is the real user id.

An effective user id is the username acquired after executing an su command.

```
# ps -U student
  PID TTY          TIME CMD
 3097 pts/3        00:00:00 bash
 3119 pts/3        00:00:00 vim
```

The illustration below shows how to have multiple options with an option requiring an argument.

```
1 # ps -ft /dev/tty1
2 UID      PID PPID  C STIME TTY          TIME CMD
3 root      1203 1191  0 12:03 tty1        00:00:04 /usr/bin/X :0 -background none -
4 #
5 # ps -f -t /dev/tty1
6 UID      PID PPID  C STIME TTY          TIME CMD
7 root      1203 1191  0 12:03 tty1        00:00:04 /usr/bin/X :0 -background none -
8 #
9 # ps -t /dev/tty1 -f
10 UID      PID PPID  C STIME TTY          TIME CMD
11 root      1203 1191  0 12:03 tty1        00:00:04 /usr/bin/X :0 -background none -
```

Quoting and Command Substitution

Topic Overview

An option is a command line word which modifies how a command executes.

Topic Objectives

- Terms
 - escape sequences
 - quoting
 - command substitution

- variable expansion
- embedded

echo

The echo command displays the command's arguments to the standard output device.

```
# echo Hello  
Hello
```

```
# echo hello world  
hello world
```

Escape Sequences

The bash shell has builtin escape sequences. Escape sequences are preceded by a backslash (\).

Table: echo command escape sequences

Escape Sequence	Output
<code>\n</code>	new line
<code>\r</code>	carriage return
<code>\t</code>	horizontal tab
<code>\v</code>	vertical tab

To use escape sequences the **echo** command requires the option **-e** and the line be enclosed in double quotes.

```
# echo -e "The default shell is: \t $SHELL"  
The default shell is: /bin/bash
```

Notice what happens if we execute the same command without quotes.

```
# echo -e The default shell is: \t $SHELL  
The default shell is: t /bin/bash
```

Quoting

Using Single Quotes

A line enclosed in single quotes is used to group words together. This prevents the shell from treating multiple words as multiple arguments. All characters between single quotes are treated as literal characters.

```
# echo 'The default shell is $SHELL'
The default shell is $SHELL
```

```
# echo 'The current working directory is $(pwd)'
The current working directory is $(pwd)
```

Using Double Quotes

A line enclosed in double quotes group the words between the double quotes together and treats them as a single unit. Double quotes permit the use of:

- Command Substitution
- Variable Expansion
- Arithmetic Expansion
 - In the examples below assume the variable 3 does not exist.
 - Line 1 executes the command **echo "I have \$300.00"**
 - The command line interpreter "sees" a dollar symbol (\$) before the 3 and attempts variable expansion.
 - Since the variable 3 does not exist the output on line 2 displays 00.00.

```
1 # echo "I have $300.00"
2 I have 00.00
```

- Line 4 below executes the command **echo "I have \\$300.00"**
- Notice the backslash (\) precedes the meta character \$.

- The output on line 5 illustrates how the command interpreter interpreted the metacharacter and printed the \$ as a literal character.

```
4 # echo "I have \$300.00"
5 I have $300.00
```

Table: Bash expansion

Expansion	Example
Command Substitution	Command substitution is also called embedding. Please reference embedding below.
Variable expansion	<code>\$<variable_name></code>
Arithmetic expansion	<code>"\${(2+1)}"</code>
Use of the backslash	<code>\\$SHELL, \t</code> <div style="border: 1px solid black; padding: 5px; width: fit-content;"> <pre># echo -e "The default shell is: \t \$SHELL" The default shell is: /bin/bash</pre> </div>

Embedding

Table: Bash expansion

Embedding	<code>`<command word>`</code> <code>\$(<command word>)</code>
------------------	--

Embedding is also called command substitution.

Embedding executes a command embedded in an existing command and replaces the embedded command with its' output.

Refer to the illustration below.

The **echo** command displays the string between the double quotes to the terminal.

The **pwd** command (`$(pwd)`) is embedded inside the **echo** command.

The `pwd` command executes as a new process.

The output of the command is `/root/Desktop`.

Notice the output has replaced **\$(pwd)** with the command output, `/root/Desktop`.

```
# echo "The current directory is $(pwd)"  
The current directory is /root/Desktop
```

The back tick (```) is normally found on the top left key of your keyboard. A command word enclosed by back ticks is executed by the shell. The illustration below is another example of embedding.

```
# echo "The current directory is `pwd`"
```

Redirection

Lesson Overview

A data stream is a sequence of digitally encoded data transmitted between two devices.

Data streaming is the process of moving a stream of data from one device to another.

Linux has three default data stream devices:

Name	File Descriptor	Default Device	Definition
stdin	fd0	Keyboard	standard input
stdout	fd1	Console	standard output
stderr	fd2	Console	standard error

Redirection is the process of altering the default data stream.

Redirection operators are evaluated left to right.

This lesson will explain key concepts associated with redirection, and demonstrate how to redirect input and output using control operators and commands.

Lesson Objectives

- **Terms**
 - stdout
 - stderr
 - file descriptor
 - named pipe
 - unnamed pipe

- **Skills**

- Redirect stdin, stdout and stderr
- Use the /proc directory, and **lsdf** command to view file descriptors for a specific process
- Use the **tee** command.

stdin, stdout, stderr

stdin

The standard input device, stdin, is the source of a data stream.. The default standard input device is the system keyboard.

stdout

The standard output device, stdout, is the destination for non error messages. The default standard output device is your terminal.

stderr

The standard error device is the destination of error messages. The default standard error device is your terminal.

File Descriptors

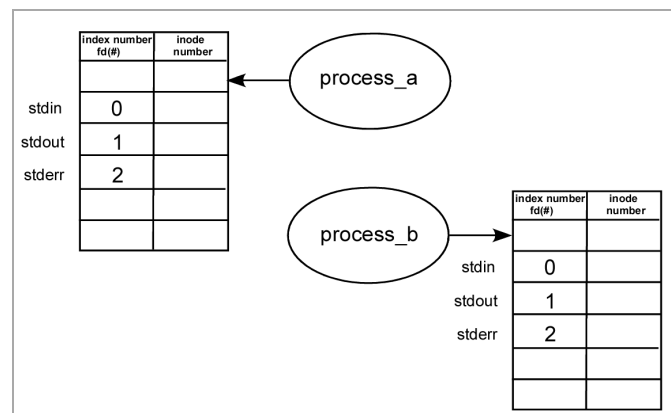
A file descriptor is a reference (also called a handle) used by the kernel to access a file.

A file descriptor table is created for most process. This table tracks all files opened by the process.

A unique index number (file descriptor number) is assigned to a file when it is opened by a process. This index number is linked to the file's inode. If a process opens the same file multiple times using different permissions (modes) additional file descriptor numbers are assigned.

When a process is created the file descriptor numbers 0, 1 and 2 are reserved for the stdin, stdout, and stderr devices.

- fd0 - standard input device, stdin.
- fd1 - standard output device, stdout.
- fd2 - standard error device. stderr.



Viewing a Process File Descriptor Table

ls -l /proc/<pid>/fd

/proc is a virtual filesystem which contains realtime system information.

Every process has a directory in */proc*. which contains process information. The directory name is the process id number.

Inside the process directory is the directory *fd*. This directory contains links between file descriptor numbers and a file opened by the process.

The command **ls -l /proc/<pid>/fd** will display all the file descriptors for a specific process, <pid>.

The output of the command **ls -l /proc/2180/fd** in the example below displays the file descriptors for process 2180. In this example, the standard input device (fd0), standard output device (fd1), and the standard error device (fd2) are linked to pseudo terminal 0, */dev/pts/0*.

```
$ ls -l /proc/2180/fd
total 0
lr-x-----. 1 student student 64 Apr  8 23:15 0 -> /dev/pts/0
lrwx-----. 1 student student 64 Apr  8 23:15 1 -> /dev/pts/0
lrwx-----. 1 student student 64 Apr  8 23:15 2 -> /dev/pts/0
lrwx-----. 1 student student 64 Apr  8 23:16 255 0 -> /dev/pts/0
                1 -> /dev/pts/0
                2 -> /dev/pts/0
```

lsdf -p

The **lsdf** command lists all files opened by all active processes.

The **-p** option restricts the output of the **lsdf** command to files opened by a specific process id, <pid>.

The command **lsdf -p 2180** will display all files opened by process 2180.

```
$ lsdf -p 2180
COMMAND PID  USER  FD  TYPE DEVICE SIZE/OFF  NODE NAME
bash    2180 student cwd   DIR   9,0   1024    27 /home/student/Desktop
bash    2180 student rtd   DIR   8,2   4096     2 /
bash    2180 student txt   REG   8,2  874184 2056 /bin/bash
bash    2180 student mem   REG   8,2  142480 1929 /lib/ld-2.12.so
bash    2180 student mem   REG   8,2 1876448 1930 /lib/libc-2.12.so
bash    2180 student mem   REG   8,2  19784 1128 /lib/libdl-2.12.so
bash    2180 student mem   REG   8,2  58704 5839 /lib/libnss_files-2.12.so
bash    2180 student mem   REG   8,2  99636 1229 /lib/libtinfo.so.5.7
bash    2180 student mem   REG 253,0 99158672 52577 /usr/lib/locale/locale-archive
bash    2180 student mem   REG 253,0 26048 13584 /usr/lib/gconv/gconv-modules.cache
bash    2180 student 0r   CHR 136,0    0t0    3 /dev/pts/0
bash    2180 student 1u   CHR 136,0    0t0    3 /dev/pts/0
bash    2180 student 2u   CHR 136,0    0t0    3 /dev/pts/0
bash    2180 student 255u CHR 136,0    0t0    3 /dev/pts/0
```

stdin

Input is received by a process through the standard input device.

The default standard input device is the console keyboard.

The default file descriptor number for the standard input device is 0. Therefore, the device name is `fd0`.

The less than symbol (`<`) and double less than symbol (`<<`) are control operators used to choose an alternate standard input device.

File Descriptor	Control Operator	Description
0	<code><</code>	Read the contents of the file specified by the argument following the less than control operator (<code><</code>) as input to the command.
0	<code><< <limit string></code>	Use the specified input source as stdin until the word, specified by the argument <code><limit word></code> is read.

`<`

In the example below the command `cat` uses the file `/etc/hosts` as its' input. (The command `cat /etc/hosts` would produce the same output).

```
$ cat < /etc/hosts
127.0.0.1    localhost.localdomain  localhost
::1        localhost6.localdomain6 localhost6
```



This form of redirection is called a here document or heretodoc. A here document allows us to access a data source to provide a list of data as standard input.

In the example below (line 1) **cat << EOF** indicates that each of the following lines will be used as input to the command **cat** until the limit string, EOF, is read (line 6).

The **cat** command will be treated as if it were the **echo** command.

Notice the output on lines 7-10.

```

1 # cat << EOF
2 > test
3 > the current shell is $SHELL
4 > the current exit statis is $?
5 > /etc/passwd
6 > EOF
7 test
8 the current shell is /bin/bash
9 the current exit statis is 0
10 /etc/passwd

```

In the example below (left) we have created a script, here document. Line 1 instructs the **grep** command to search through the input source for a string specified by the first argument of the command line. The input source begins on line 2 and ends when the word EOF (<< EOF) is read.

On the right we have executed the script *heredoc* (left) looking for the string Dumas.

```

1 grep $1 << EOF
2 Charles Dickens
3 Alexandar Dumas
4 Paulo Cohello
5 Don Miguel Ruiz
6 EOF

```

```

1 # heredoc Dumas
2 Alexandar Dumas

```

Another here-document example may be found in a Grub2 configuration script.

```

1 #!/bin/sh -e
2 cat << EOF
3 if [ -f \${prefix}/user.cfg ]; then
4     source \${prefix}/user.cfg
5     if [ -n "\${GRUB2_PASSWORD}" ]; then
6         set superusers="root"
7         export superusers
8         password_pbkdf2 root \${GRUB2_PASSWORD}
9     fi
10 fi
11 EOF

```

stdout

The output of a successful command is directed to the standard output device, stdout.

The default standard output, stdout, device is the console monitor.

The default file descriptor number for the standard output device is 1. Therefore, the device name is fd1.

The greater than (>) and double greater than symbols (>>) are control operators used to choose an alternate standard output file or device.

File Descriptor	Control Operator	Description
1	>	<p>Write all non error message to the file or device specified after the control operator 1></p> <p>If the file does not exist create the file</p> <p>If the file existst write over the data in the file</p>
1	>>	<p>Write all non error message to the file or device specified after the control operator 1>></p> <p>If the file exists append the output message to the end of the file.</p> <p>If the file does not exist create the file.</p>

In the example below the output of the command `ls -ld /etc /usr` is redirected to the file `stdoutfile` (`> stdoutfile`).

```

1 # ls -ld /etc /usr > stdoutfile
2 #
3 # cat stdoutfile
4 drwxr-xr-x. 141 root root 9216 Aug 11 12:08 /etc
5 drwxr-xr-x.  14 root root 4096 Jun 26 03:49 /usr

```



The shell parameter `noclobber` will not allow the `>` redirection to overwrite an existing file.

The command `set -o | grep noclobber` will verify if `noclobber` is set

The command `set -o noclobber` will turn the parameter off.

The command `cat /etc/hosts >> stdoutfile` appends the contents of the file `/etc/hosts` to the end of the file `stdoutfile`.

```

1 # cat /etc/hosts >> stdoutfile
2 #
3 # cat stdoutfile
4 drwxr-xr-x. 141 root root 9216 Aug 11 12:08 /etc
5 drwxr-xr-x. 14 root root 4096 Jun 26 03:49 /usr
6 127.0.0.1    localhost localhost.localdomain localhost4 localhost4.localdomain4
7 ::1        localhost localhost.localdomain localhost6 localhost6.localdomain6

```

stderr

Error messages are output to the standard error device, `stderr`.

The default standard error device, is the console monitor.

The default file descriptor number for the standard error device is 2. Therefore, the device name is `fd2`.

The operators `2>` and `2>>` are control operators used to chose an alternate standard error file or device.

File Descriptor	Control Operator	Description
2	2>	<p>If an execution error occurs write all error messages to the file specified by the argument after the control operator <code>2></code>,</p> <p>If the file does not exist create the file</p> <p>If the file does exist write over the data in the file</p>
2	2>>	<p>If an execution error occurs write all error messages to the file specified by the argument after the control operator <code>2></code>,</p> <p>If the file exists append the output message to the end of the file.</p> <p>If the file does not exist create the file</p>

The command `ls -ld /etc /roses` in the example below will attempt to list the properties of the directories `/etc` and `/roses`. The directory `/roses` does not exist.

The control operator `2>` will redirect the error messages to the file `errorfile`.

The properties of the directory `/etc` are displayed on the standard output device (line 2) .

The data stream for error messages has been redirected to the file `errorfile` (line 4).

```
1 # ls -ld /etc /roses 2> errorfile
2 drwxr-xr-x. 141 root root 9216 Aug 11 16:32 /etc
3 #
4 # cat errorfile
5 ls: cannot access /roses: No such file or directory
```

2>&1

The combined control operators `2>&1` redirect the standard error to the same location as the standard output.

- The control operator `2>` designates we are altering the data stream for error messages.
- The `&` is a Boolean and.
- The `1` is the file descriptor number for the standard output data stream,

In the example below the stdout is being redirected to the file `dirlist` (`> dirlist`), and the stderr is "anded" to the standard output (`2>&1`). Therefore, both the standard output and standard error data stream are being redirected to the file `dirlist`.

```
1 # ls -ld /etc /acorn > dirlist 2>&1
2 #
3 # cat dirlist
4 ls: cannot access /acorn: No such file or directory
5 drwxr-xr-x. 141 root root 9216 Aug 11 16:32 /etc
```


1>&2

The combined control operators 1>&2 redirects the standard output to the same location as the standard error.s.

- The 1> control operator redirects the standard out[put].
- The & is a Boolean and.
- The 2 is the file descriptor number for the standard error data stream.
- 1>&2 redirects stdout (1>) to the same device as stdout (&2).

In the example below the stderr is being redirected to the file `/root/errorfile` (`2> /root/errorfile`), and the stdout is "anded" to the standard output (`1>&2`). Therefore, both the standard error and standard out data stream are being redirected to the file `/root/errorfile`.

```
1 # ls -ld /turnip /etc 2> /root/errorfile 1>&2
2 #
3 # cat /root/errorfile
4 ls: cannot access /turnip: No such file or directory
5 drwxr-xr-x. 141 root root 9216 Aug 11 16:32 /etc
```

>&

The control operator >& redirects both the standard output and standard error to the same file.

```
1 # ls -ld /turnip /etc >& filea
2 #
3 # cat filea
4 ls: cannot access /turnip: No such file or directory
5 drwxr-xr-x. 141 root root 9216 Aug 11 16:32 /etc
```

pipe

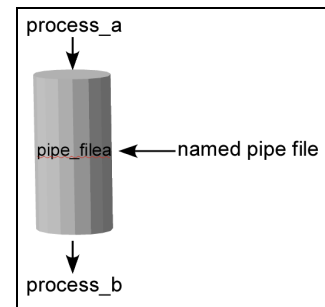
A pipe is a method of connecting the stdout of one process to the stdin of another process.

named pipe

A named pipe is a file which facilitates interprocess communication.

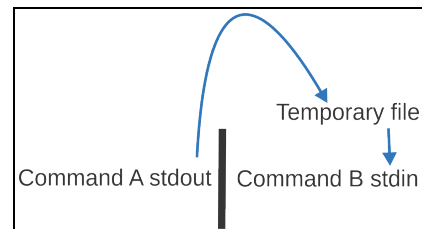
In the example to the right process_a writes data to the file *pipe_filea* and process_b reads data from *pipe_filea*.

The **mknod** or **mkfifo** commands create a named pipe file. Deleting the file removes the named pipe



unnamed pipe

An unnamed pipe is a temporary file (dynamically created) used to transfer the standard output data stream from the command to the right of the pipe control operation (|) to the standard input data stream of the command to the right of the pipe control operator. Once the transfer process completes the temporary file is deleted.

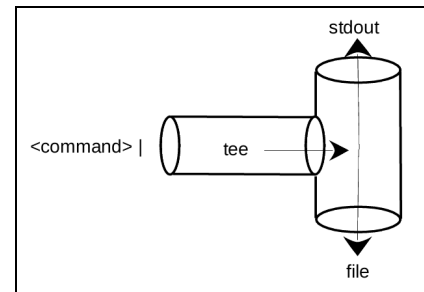


The unnamed pipe in the example to the right redirects the stdout of the command **cat file1** (this is file1) to the stdin of the command **cat > filea**.

```
1 # cat file1
2 this is file1
3 #
4 # cat file1 | cat > filea
5 #
6 # cat filea
7 this is file1
```

tee

The **tee** command redirects the standard input to both the stdout device and a file. If the file does not exist it will be created. If the file does exist it will be overwritten.



Option	Argument	Description
-a	<file_name>	Name of file to write output to. If the file specified by the argument <file_name> exists append the command's output to the file.

In the example below the standard output of the command **ls -ld /etc** becomes the standard input to the **tee** command.

The tee command writes the output to both the standard output device (line 2) and the file *teetest* (line 4).

```

1  -ld /etc | tee teetest
2  drwxr-xr-x. 141 root root 9216 Aug 11 16:32 /etc
3  #
4  # cat teetest
5  drwxr-xr-x. 141 root root 9216 Aug 11 16:32 /etc

```

Regular Expressions

Lesson Overview

A regular expression is a string which defines a pattern text to search for within a file or files.

A regular expression engine processes regular expressions. Many languages (Python, Perl, Java) have their own regular expression engine. These engines are not always compatible. Therefore, an expression which is valid on one engine may not be valid on another.

The commands **grep**, **egrep** and **fgrep** are used to search for patterns within text.



The commands **egrep** and **fgrep** have been deprecated, but are included in the operating system for backward compatibility,

The commands **grep -E** and **grep -F** have replaced the commands **egrep** and **fgrep**.

Our discussions will include basic, BRE, and extended, ERE regular expressions.

Lesson Objectives

- **Terms**

- anchors
- boundaries
- quantifiers
- class
- groups

- **Commands**

- **grep**
- **egrep**
- **fgrep**

Basic Regular Expression

The **grep** command uses BRE expressions.

A backslash (\) must precede most meta characters in a basic regular expression

Expressions that contain meta characters preceded by a backslash must be enclosed in single quotes (') to insure interpretation by the regex engine and not the command line interpreter.

The basic regular expression in line 1 below will find any string in the file */etc/passwd* that has the letter o appear two times consecutively (oo). The expression is enclosed in single quotes to avoid interference from the command interpreter.

```
1 # grep o'\{2\}' /etc/passwd
2 root:x:0:0:root:/root:/bin/bash
3 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
4 mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
5 operator:x:11:0:operator:/root:/sbin/nologin
6 setroubleshoot:x:992:989::/var/lib/setroubleshoot:/sbin/nologin
7 postfix:x:89:89::/var/spool/postfix:/sbin/nologin
```



The meta characters *, ^, \$, or [] do not need to be preceded by a backslash when used in a Basic

Extended Regular Expressions

UNIX developers created the **egrep** command to expanded the capabilities of the **grep** command.

The **grep** and **egrep** commands are not compatible, but the command **grep -E** interprets search patterns as an extended regular expression rather than basic regular expressions.

Extended Regular Expressions, ERE, do not require escaping the meta characters or enclosing the expression in single quotes.

```
1 # egrep o{2} /etc/passwd
2 root:x:0:0:root:/root:/bin/bash
3 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
4 mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
5 operator:x:11:0:operator:/root:/sbin/nologin
6 setroubleshoot:x:992:989::/var/lib/setroubleshoot:/sbin/nologin
7 postfix:x:89:89::/var/spool/postfix:/sbin/nologin
```

grep

The `grep`, Global Regular Expression Print, command searches for lines in a file or files which contains a specific pattern or patterns and prints the lines which contain matching patterns.

Command Type	Exttternal
Permissions	Available to all users
Documentation	man and info pages, grep info pages contains additional helpful information

By default `grep` searches for a string.

The `grep` command is not POSIX compatible.



POSIX, Portable Operating System Interface, defines the interface between programs.

In the example below the `grep` command will find the string `root` in the file `/etc/passwd`

```
# grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
```

The command `grep root /etc/passwd /etc/shadow /etc/group /etc/gshadow` will search the files `/etc/passwd` `/etc/shadow` `/etc/group` and `/etc/gshadow` for the string `root`.

Each line of the output will be prefaced by the name of the file where the string match was found.

```
# grep root /etc/passwd /etc/shadow /etc/group /etc/gshadow
/etc/passwd:root:x:0:0:root:/root:/bin/csh
/etc/passwd:operator:x:11:0:operator:/root:/sbin/nologin
/etc/shadow:root:$6$Sc6Je0hRsFBsUy9r$LkC7p.JFhFqw3guZfF1sx0SVdL/mjZ5dybNRH.8xV0Zrd3mRFYUcqEV/
zp7Rk15nXg36qofth2Mf5LjJ5TWdJ1::0:99999:7:::
/etc/group:root:x:0:
/etc/gshadow:root:::
```

To find a meta character used as a literal character in a BRE expression you must precede it with a backslash.

The file *grepbackslash* contains the following lines:

```
1 The \ in front of a meta character negates the meaning of the metacharcter
2 Let's look at the following example
3 The command echo \ $SHELL will print $SHELL
```

The command **grep -n '\'** *grepbackslash* will display the lines which contain a backslash.

```
# grep -n '\ ' grepbackslash
1:The \ in front of a meta character negates the meaning of the metacharcter
3:The command echo \ $SHELL will print $SHELL
```

Table: grep Options

Option	Argument	Description
-i		Ignore case.
-v		Output lines that <u>do not</u> match the expression pattern.
-c		Count and print the number of lines which match the expression pattern
-n		Print the number of the line the matched expression
-w		Search for a word (string terminated by a white space character)
-f	<filename>	Retrieve regular expression pattern from the file specified by the argument filename.
-R or -r		Search files in the current directory and subdirectories.
-A #		Display the line which matches the regular expression and the following <i>n</i> number of lines.

Table: grep Options (continued)

Option	Argument	Description
-B #		Display the line which matches the regular expression and <i>n</i> number of lines before.
-C #		Display the line which matches the regular expression and <i>n</i> number of lines before and after.
-E		Extended grep (egrep). Accepts the syntax of the egrep command
-F		Fixed string (fgrep), fgrep treats all characters based on their encoded value.

grep -i

The **-i** option ignores the case of the search pattern.

The file *grepi* (left) contains the word truck in three formats.

The command **grep truck grepi** (line 1 right illustration) searches for the string truck in the file *grepi*. This search requires **grep** to find an exact string match.

The command **grep -i truck** (line 4 right illustration) searches for the string truck regardless of character case.

```
1 My truck is a Ford.
2 The Truck is red.
3 The monster Truck fell over.
```

```
1 # grep truck grepi
2 My truck is a Ford.
3 #
4 # grep -i truck grepi
5 My truck is a Ford.
6 The Truck is red.
7 The monster Truck fell over.
```


grep -v

The **-v** option will display all lines which do not contain the search pattern.

```
# grep -v root /etc/passwd
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
```

Output Edited

grep -c

The **-c** option displays the number of lines which contain the string.

```
# grep -c root /etc/passwd
2
```

Adding the **-v** option will count the number of lines which do not contain the string root.

```
# grep -cv root /etc/passwd
45
```

grep -n

The **-n** option will add the line number the pattern is found on to the output.

```
# grep -n root /etc/passwd
1:root:x:0:0:root:/root:/bin/bash
10:operator:x:11:0:operator:/root:/sbin/nologin
```

grep -w

The `-w` option will search for a word rather than a string.

The file `grepw` contains the words `breakfast` (line 2) and `break` (line 3).

```
1 # cat grepw
2 The most important meal is breakfast.
3 My break is at 10:00AM
```

The command `grep break grepw` (line 1 left illustration below) searches for the string `break` in the file `grepw`.

The command `grep -w break grepw` (line 1 right illustration below) searches for the word `break` in the file `grepw`.

```
1 # grep break grepw
2 The most important meal is breakfast.
3 My break is at 10:00AM
```

```
1 # grep -w break grepw
2 My break is at 10:00AM
```

grep -f <filename>

The command `grep -f` retrieves a regular expression pattern or patterns from the file specified by the argument `filename`.

In the illustration below right the file `grepfile` contains the regular expression pattern `^en`. This pattern search for the string `en` at the beginning of a line.

The illustration below left filters the output of the `ifconfig` by searching for the pattern set in the file `grepfile` (`-f grepfile`).

```
# cat grepfile
^en
```

```
# ifconfig | grep -f grepfile
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
```

grep -r or -R

The options **-r** or **-R** perform a recursive search for a pattern. Recursive searches will examine all files in the current working directory and sub directories.

In the example below the current working directory is `/etc`. The command **grep -r linuxuser | more** will display all the files which contain linuxuser in the current working directory and sub directories.

Notice the search starts in the directory `/etc` and continues searching sub directories.

```

1 # pwd
2 /etc
3 #
4 # grep -r linuxuser | more
5 shadow-:linuxuser:$6$kcjxoz49ex8HSmzZ$jQ0FxFJ1iwXRvejcwP90XYKYfDrt2gw5YdH3oHbpVXtvpDQ8oz8TqH
6 zipm/zKgm5.ZTtTf5FGgp1sVVvmWVd/::0:99999:7:::
7 group-:linuxuser:x:1000:linuxuser
8 gshadow:linuxuser:!!:linuxuser
9 group:linuxuser:x:1000:linuxuser
10 selinux/targeted/booleans.subs dist:allow execheap selinuxuser execheap

```

grep -A

The **-A #** option will display the line which matches the regular expression and the following *n* number of lines.

The command below will find all network interface names which begin with `en` and display their IPV4 and IPV6 addresses.

```

[root@localhost ~]# ifconfig | grep -A 2 '^en'
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
        inet6 fe80::1a09:94a4:728f:4e19 prefixlen 64 scopeid 0x20<link>

```

If we had not include the **-A 2** option we would not have seen the additional `enp0s3` configuration information.

```

# ifconfig | grep -f grepfile
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

```

egrep

egrep, Extended Global Regular Expression Print, was designed to expand the capabilities of the grep command on UNIX systems. egrep has been deprecated, but the command is provided for backwards compatibility.

The command **grep -E** will provide the functionality of the egrep command.

The options which were illustrated for the grep command (page 86

fgrep

The fgrep, Fixed StringRegular Expression Print command interprets all characters based on their encoded value.

Although fgrep has been deprecated the command is provide for backwards compatibility.

The command **grep -F** will provide the functionality of the fgrep command.

The command **cat -e .bash_profile** command below prints the file with the \$ end of line character.

```
# cat -e .bash_profile
# .bash_profile$
#$
# Get the aliases and functions$
if [ -f ~/.bashrc ]; then$
    . ~/.bashrc$
fi$
#$
# User specific environment and startup programs$
#$
PATH=$PATH:$HOME/.local/bin:$HOME/bin$
#$
export PATH$
```



The **-e** option displays the end of line character, \$.

The command **grep \$./bash_profile** will search for lines in the file `./bash_profile` that contain the `$` character and meta character.

Notice all lines are printed.

```
# grep $ ./bash_profile | cat -e
# ./bash_profile$
$
# Get the aliases and functions$
if [ -f ~/.bashrc ]; then$
    . ~/.bashrc$
fi$
$
# User specific environment and startup programs$
$
PATH=$PATH:$HOME/bin$
$
export PATH$
```

The command **fgrep \$./bash_profile** interprets the `$` based on it's encoded value only.

```
# fgrep $ ./bash_profile
PATH=$PATH:$HOME/.local/bin:$HOME/bin
```

Regular Expression Metacharacters

The following will explain some of the meta characters used in regular expressions.



BRE indicates a meta character used with Basic Regular Expressions (grep).

ERE indicates a meta character used with Extended Regular Expressions (egrep).

If BRE or ERE are not indicated the metacharacters apply to both basic and extended regular expressions.

Quantifiers

Quantifiers (also called iterations) are meta characters used define repeating characters.

Table: Quantifiers

Quantifier Operator	Type	Definition
\? (Question Mark) ?	BRE ERE	The question mark (?) will match the preceding character 0 or 1 time.
* (Asterisk)		The asterisk (*) will match the preceding character 0 or more times.
\+ +	BRE ERE	Matches one or more occurrences of the character to the left
\{n} {n}	BRE ERE	Match <i>n</i> occurrences of the character to the left.
\{n,} {n,}	BRE ERE	Matches <i>n</i> or more occurrences off the characer too the left.
\{n,m} {n,m}	BRE ERE	Math a minimum of <i>n</i> and a maximum number of <i>m</i> occurrences of the character to the left.

?

Please reference the contents of the file greptesta for the following examples.

```
# cat greptesta
cart
car
cat
```

The pattern found by **grep** or **egrep** is in **red**.

The expression below searches for a string which begins with the letter c and then 0 or one occurrence of the letter a followed by the letter t.

```
# grep 'ca?t' greptesta
cat
```

```
# egrep ca?t greptesta
cat
```

The expression below searches for a string which begins with the letter c and then 0 or one occurrence of the letter a followed by the letter r.

```
# grep 'ca?r' greptesta
cart
car
```

```
# egrep ca?r greptesta
cart
car
```

The expressions below search for a string which begins with the letter c and then 0 or one occurrence of the letter a followed by the letter t.

```
# cat greptestb
cart
car
cat
ct
```

```
# grep 'ca?t' greptestb
cat
ct
```

```
# egrep ca?t greptestb
cat
ct
```

*
—

The asterisk (*) will match the character to the left 0 or more times.

```
# grep ro* /etc/passwd
root:x:0:0:root:/root:/bin/bash
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
systemd-bus-proxy:x:999:997:systemd Bus Proxy:/:/sbin/nologin
```

Output Edited

+
—

Matches one or more consecutive occurrences of the character to the left.

The expression below will search the file */etc/passwd* for a string which contains one or more consecutive occurrences of the number 9.

```
# grep '9\+' /etc/passwd
nobody:x:99:99:Nobody:/:/sbin/nologin
systemd-bus-proxy:x:999:997:systemd Bus Proxy:/:/sbin/nologin
systemd-network:x:192:192:systemd Network Management:/:/sbin/nologin
polkitd:x:998:996:User for polkitd:/:/sbin/nologin
colord:x:997:994:User for colord:/var/lib/colord:/sbin/nologin
unbound:x:996:993:Unbound DNS resolver:/etc/unbound:/sbin/nologin
tss:x:59:59:Account used by the trousers package to sandbox the tcsd daemon:/dev/null:/sbin/nologin
```

Output Edited

```
# egrep 9+ /etc/passwd
nobody:x:99:99:Nobody:/:/sbin/nologin
systemd-bus-proxy:x:999:997:systemd Bus Proxy:/:/sbin/nologin
systemd-network:x:192:192:systemd Network Management:/:/sbin/nologin
polkitd:x:998:996:User for polkitd:/:/sbin/nologin
colord:x:997:994:User for colord:/var/lib/colord:/sbin/nologin
unbound:x:996:993:Unbound DNS resolver:/etc/unbound:/sbin/nologin
tss:x:59:59:Account used by the trousers package to sandbox the tcsd daemon:/dev/null:/sbin/nologin
```

Output Edited

{n}

Match n occurrences of the character to the left.

The expressions below search the file `/etc/passwd` for a string which contains 2 consecutive occurrences of the character 0.

```
# grep '0\{2\}' /etc/passwd
games:x:12:100:games:/usr/games:/sbin/nologin
linuxuser:x:1000:1000:Linux User:/home/linuxuser:/bin/bash
```

```
# egrep 0{2} /etc/passwd
games:x:12:100:games:/usr/games:/sbin/nologin
linuxuser:x:1000:1000:Linux User:/home/linuxuser:/bin/bash
```

{n,}

Matches n or more consecutive occurrence off the characer to the left.

The expressions below searches the file `/etc/passwd` for a string which contains 2 or more consecutive occurrences of the character 0.

```
# grep '0\{2,\}' /etc/passwd
games:x:12:100:games:/usr/games:/sbin/nologin
linuxuser:x:1000:1000:Linux User:/home/linuxuser:/bin/bash
```

```
# egrep '0{2,}' /etc/passwd
games:x:12:100:games:/usr/games:/sbin/nologin
linuxuser:x:1000:1000:Linux User:/home/linuxuser:/bin/bash
```

{n,m}

Match a minimum number, *n*, and a maximum number, *m*, consecutive occurrences of the character to the left.


The expressions below will search the file `/etc/passwd` for a string which contains a minimum of 2 and maximum of 3 consecutive occurrences of the character `0`.

```
# grep '0\{2,3\}' /etc/passwd
games:x:12:100:games:/usr/games:/sbin/nologin
linuxuser:x:1000:1000:Linux User:/home/linuxuser:/bin/bash
```

Anchors

An anchor specifies where the match must take place.

Table: Anchors

Anchor Operator	Definition
<code>^<string></code>	The string must be found at the beginning of a line.
<code><string>\$</code>	The string must be found at the beginning of a line.
<code>.</code> (Period)	Holds a single character position for any character except a new line (line break) character.  A period (.) inside a class group [a.c] is interpreted using its literal value.

^<string>

The string must be found at the beginning of a line.

The ^ only applies to the string it prefixes.

The expression on line 5 searches the file */etc/passwd* for the string root at the beginning of a line.

```

1 # grep root /etc/passwd
2 root:x:0:0:root:/root:/bin/bash
3 operator:x:11:0:operator:/root:/sbin/nologin
4 #
5 # grep ^root /etc/passwd
6 root:x:0:0:root:/root:/bin/bash

```

The ^ only applies to the string it prefixes. The expression below searches for the letter *r* or the letter *l* at the beginning of a line.

```

# egrep '^r|^l' /etc/passwd
root:x:0:0:root:/root:/bin/bash
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
libstoragemgmt:x:995:992:daemon account for libstoragemgmt:/var/run/lsm:/sbin/nologin
rpc:x:32:32:Rpcbind Daemon:/var/lib/rpcbind:/sbin/nologin
rtkit:x:172:172:RealtimeKit:/proc:/sbin/nologin
radvd:x:75:75:radvd user://sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
linuxuser:x:1000:1000:Linux User:/home/linuxuser:/bin/bash

```

<string>\$

The string must be found at the end of a line.

The expression on line 5 searches the file */etc/passwd* for the string bash at the end of the line.

```

1 # grep bash /etc/passwd
2 root:x:0:0:root:/root:/bin/bash
3 linuxuser:x:1000:1000:Linux User:/home/linuxuser:/bin/bash
4 #
5 # grep bash$ /etc/passwd
6 root:x:0:0:root:/root:/bin/bash
7 linuxuser:x:1000:1000:Linux User:/home/linuxuser:/bin/bash

```

. Period

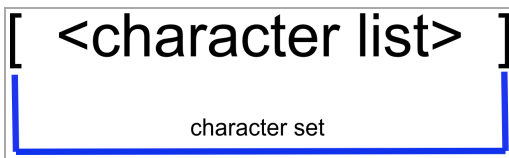
Holds a single character position for any character except a new line (line break) character.

The expression below looks in the file `/etc/passwd` for a string which starts with the letter `r`, ends with the letter `t`. The characters `r` and `t` are separated by 2 characters (`r..t`).

```
1 # grep r..t /etc/passwd
2 root:x:0:0:root:/root:/bin/bash
3 operator:x:11:0:operator:/root:/sbin/nologin
4 ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
```

Character Set

A character set contains a list of characters. A regular expression matches a single character from the character list for the character position which contains the character set.



The character set may contain multiple characters, range of characters or POSIX class operators.

Table: Class Operator

Class	Definition
<code>[<character><character>]</code>	<p>Multiple characters act as an or.</p> <p>The character set <code>[ab]</code> would accept either the letter <u>a</u> or letter <u>b</u> for the current character position.</p>
<code>[<range>]</code>	<p>Specify a character range</p> <p><code>[a-z]</code> Specifies the character set lower case a through lower case z.</p> <p><code>[A-Z]</code> Specifies the character set upper case A through upper case Z.</p> <p><code>[0-9]</code> Specifies the range of digits 0 through 9.</p>

Table: Class Operator (continued)

Class	Definition
[^<range>]	The caret (^) inside the brackets acts as a not. The character set [^ab] excludes the characters a and b from the current cursor position.

The command **grep [nN]network /etc/passwd** will search for a string which begins with a member of the character list [nN] followed by the string `etwork` in the file `/etc/passwd`.

```
# grep [nN]network /etc/passwd
systemd-network:x:192:192:systemd Network Management:/:/sbin/nologin
```

Multiple ranges may be specified in a character list. In the example the expression will match the digits 0,1,2 or 3 or the characters f, g and h.

```
# grep [0-3f-h] ifcfg-enp0s3
TYPE="Ethernet"
BOOTPROTO="dhcp"
NAME="enp0s3"
UUID="f4849362-01ce-4cec-ae33-dfcf242557ad"
DEVICE="enp0s3"
```

The illustration below illustrates matching two sequential character positions.

The first character position must contain a 0,1,2, or 3, and the second character position must contain the character s or the digit 6.

```
# grep [0-3][s6] ifcfg-enp0s3
NAME="enp0s3"
UUID="f4849362-01ce-4cec-ae33-dfcf242557ad"
DEVICE="enp0s3"
```

If a character list is preceded by a caret (^) the characters list becomes a non matching listing.

The regular expression will find all characters except those in the character list.

```
# grep [^abc] /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

(Output Edited)
```

POSIX Character Sets

POSIX class operators are predefined character sets. These operators must be placed inside a left and right bracket, `[:<posix-operator>:]`

Table: POSIX Class Operator

Operator	Definition
<code>[:alphnum:]</code>	Specifies the character set which includes all alphanumeric characters (A-Z, a-z, and 0-9).
<code>[:alpha:]</code>	Specifies the character set A-Z, and a-z.
<code>[:digit:]</code>	Specifies the character set 0-9.
<code>[:upper:]</code>	Specifies a character set of upper case characters (A-Z).
<code>[:lower:]</code>	Specifies a character set of lower case characters (a-z).

The illustrations below show examples of using POSIX character sets.

```
# cat characlass
student1
studenta
studentA
student2
studentb
studentB
```

```
# egrep student[:digit:] characlass
student1
student2
```

```
# egrep student[:alpha:] characlass
studenta
studentA
studentb
studentB
```

Vi

Lesson Overview

Vi is a cross platform, version of the vi text editor created by Bill Joy and Chuck Haley. Joy and Haley took features from the editors ed and em and developed the editor ex. Bill Joy developed vi from the editor ex. Bram Moolenaar developed vim for the Amiga in 1988, and released it as a text editor in 1991. A graphical version of Vim, gvim, was released in 1996

This chapter is designed to give you a working knowledge of the vi editor.

Lesson Objectives

1. edit files using vi
2. Setup vi's operating environment

Web References

vim website	http://www.vim.org
documentation	http://vim.org/docs.php
	http://vimdoc.sourceforge.net/
	http://www.eandem.co.uk/mrw/vim/usr_doc/doc_ltrm.pdf

Starting Vim

Syntax

Table: vim start options

<u>Option</u>	<u>Argument</u>	<u>Description</u>
-o		Open multiple files in separate, horizontal windows
-O		Open multiple files in separate, vertical windows
-R		Open a document in read only mode. The view command will also open a document in read only mode.
	<code><file_name></code>	When starting vi the argument <filename> specifies the file to edit. If the file does not exist, a new file will be created

The command **vi** will execute the command `/usr/bin/vi` and the command **vim** will execute the command `/usr/bin/vim`. **vim** is the enhanced version of vi.

The screen below illustrates the start screen for vi and vim. A clear screen will appear once a command mode is entered.

```

-
-
-          VIM - Vi IMproved
-           version 7.0.237
-           by Bram Moolenaar et al.
- Vim is open source and freely distributable
-
-   Help poor children in Uganda!
- type  :help iccf<Enter>      for information
-
- type  :q<Enter>              to exit
- type  :help<Enter> or <F1>   for on-line help
- type  :help version7<Enter> for version info
-
-
-
-

```


Modes of Operation

Command Mode

This mode consists of full screen editor operations such as moving the cursor, cutting and pasting, and marking text. To enter the command mode press the ESC key once. Vi will remain in the command mode until the ESC key is pressed again.

Command line

This mode performs search and replace, edit operations on multiple lines and external operating system commands. This mode is entered by pressing the ESC key and the colon (:) key once. Once this sequence is pressed, the cursor will move to the bottom of the page and present the user with a colon (:) as a prompt.

Edit Mode

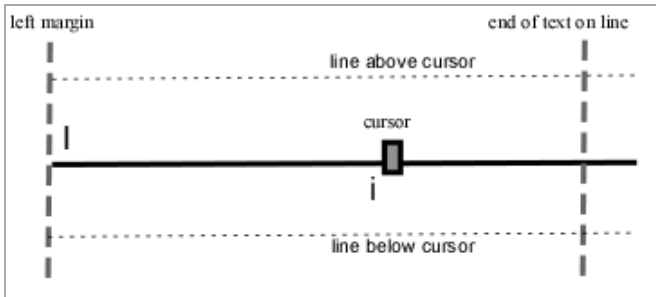
The edit mode is a subset of the command mode. It consists of three modes;

- insert
- append
- open

Edit modes are entered by pressing the ESC key once and the appropriate edit key.

Insert

The insert mode inserts text to the left of the cursor. The insert mode is entered by the pressing the ESC key followed by a lowercase or uppercase i. The lowercase i (ESC + i) inserts text to the left of the current cursor position. The uppercase i (I) (ESC + I) inserts text at the beginning of the line.



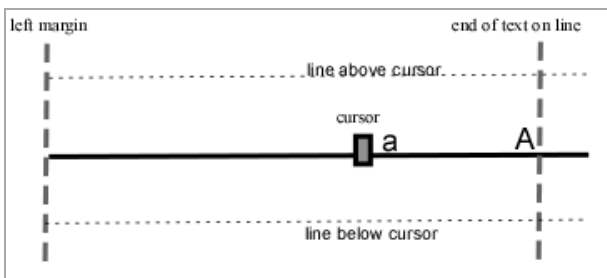
In the example below, we have inserted the numbers seven (7) and eight (8) to the left of the number five (5). Characters will continue to be inserted to the left of five (5) as long as we remain in the insert mode.



To exit the insert mode press the ESC key once.

Append

The append mode “appends” text to the right of the cursor. A lower case a (ESC + a) appends to the right of the current cursor position and an uppercase a (ESC + A) appends to the end of the line.



In the example below we have appended the letter a to the right of the number five (5). We will continue to append characters to the right of (5) as long as we remain in the append mode.

```
1 2 3 478 | 6
```

```
1 2 3 478 | 5 6
```

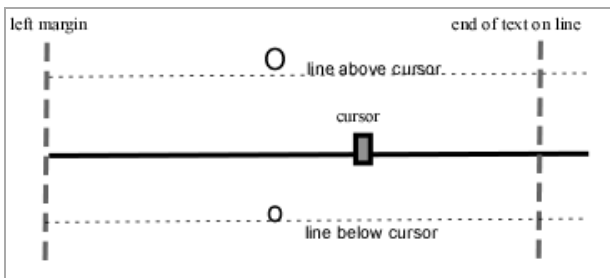
```
1 2 3 478 | 5a 6
```

```
1 2 3 478 | 5ab 6
```

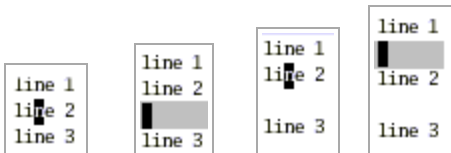
To exit the append mode press the ESC key once.

Open

The open mode uses the letter o to open a new line above or below the current cursor position. A lower case o (ESC+ o) opens a new line below the current cursor position puts the editor in insert mode. An uppercase O (ESC+ O) opens a new line above the current cursor position and puts the editor in insert mode.



In the example below we have opened a line below the current cursor position and opened a line above the current cursor position.



To exit the open mode press the ESC key once.

Cursor Movement

The cursor may be moved by character, word, line or multiple characters, words and lines.

Table: vim cursor movement

<u>Key Sequence</u>	<u>Description</u>
ESC + h	Move one character to the left.
ESC + l	Move one character to the right

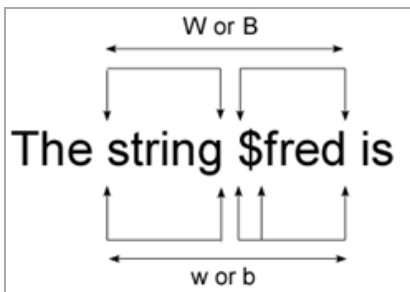
All commands may be preceded by a number *n* to repeat the command *n* number of times. For example, **5h** would move the cursor 5 characters to the left. This applies to most Vi commands.

Cursor Word Movement

A word is defined two ways.

A word may be a group of characters delimited by a space or tab.

A word may also be delimited by a space, tab or meta character.



Any word or line action begins from the current cursor position.

Table: vim word movement

Key Sequence	Description
ESC+w	Move one word to the right using a space(s), tab(s) or metacharacter(s) as a word delimiter
ESC + W	Move one word to the right using a space(s) or tab(s) as the word delimiter. Metacharacters are not considered as a word delimiter.
ESC+b	Move one word to the left using a space(s), tab(s) or metacharacter(s) as a word delimiter
ESC+B	Move one word to the left using a space(s) or tab(s) as the word delimiter. Metacharacters are not considered as a word delimiter

Cursor Line Movement

Table: vim line movement

Key Sequence	Description
ESC+j	Move one line down.
ESC +k	Move one line up.
ESC+\$	Move cursor to the end of the line.
ESC+0	Move cursor to the beginning of the line.
ESC+^	Move cursor to the first character of a line.

Manipulating Text

Deleting Text

When text is deleted, it is placed in a buffer. This buffer is used when copying and moving text.

Table: vim delete characters words lines.

Key Sequence	Description
ESC+x	Delete the character the cursor is on.
ESC +X	Delete the character immediately to the left of the cursor.
ESC+dw	Delete the current word from the current cursor position If the cursor is on the letter e in the word break ESC+dw will delete eak.
ESC+dd	Delete the current line.

Replacing Text

Table: vim delete characters words lines.

Key Sequence	Description
ESC+r	Replaces the character the cursor is on with the next character typed.
ESC +R	Replace mode. Continues to replace (overwrite) characters until the ESC key is pressed
ESC+cw	Change word. Changes the current word from the current cursor position.
ESC+xp	Transposes text. Deletes the character the cursor is currently (ESC x) and places it to the right of the next character (ESC p)

Copying Text

Table: vim copying text

Key Sequence	Description
ESC+yy	Copy the line starting with the current cursor position to the unnamed buffer. Placing a number in front of yy wil copy <i>n</i> number of lines. 3yy will copy three lines from the current cursor position
ESC +p	Places the lines in the unnamed buffer on the line below the current cursor position
ESC+P	Places the lines in the unnamed buffer on the line above the current cursor position.

Moving Text

Table: vim moving text

Key Sequence	Description
ESC+dd	dd is used to move a line or lines from one place to another. Cut the line starting with the current cursor position to the unnamed buffer. Placing a number in front of dd wil cut <i>n</i> number of lines 3dd will cut three lines from the current cursor position
ESC +p	Places the lines in the unnamed buffer on the line below the current cursor position
ESC +P	Places the lines in the unnamed buffer on the line above the current cursor position
ESC+xp	Transposes text.

Table: vim moving text(continued)

Key Sequence	Description
	Deletes the character the cursor is currently (ESC x) and places it to the right of the next character (ESC p).

Undoing commands

Table: vim undoing commands

Key Sequence	Description
ESC +u	Undo the last change. When executed on the command line will undo the last command executed.
ESC +U	Undo all changes that occurred on the current line

Screen Positioning

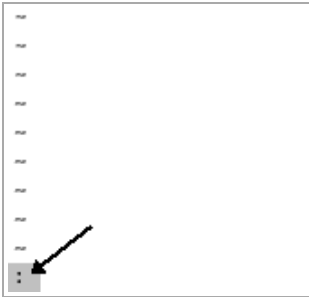
Table: vim screen positioning

Key Sequence	Description
ESC +H	Move the cursor to the highest line on the screen.
ESC +M	Move the cursor to the middle of screen.
ESC +L	Move the cursor to the last line of the screen.
ESC +gg	Move the cursor to the top of the file.
ESC +#G	Move the cursor to a specific line. ESC +3G will move the cursor to line 3.

Command Line Mode

The command line mode is entered by pressing the ESC key and then the colon (:) key.

The cursor will move to the bottom of the page and present a colon as a prompt.



Opening Documents from inside Vi

Table: vim opening documents

Key Sequence	Description
ESC +: +r <file_name>	Reads the file (file_name) into the current document at the current cursor position
ESC +: +e <file_name>	Edit a new file <file_name>. You must save the current file you are editing before executing this command
ESC +: +e! <file_name>	Edit a new file <file_name> without saving current file

Saving and Exiting

Any time a change is made to the buffer, vi will require you to save the file before exiting. The ! character overrides any vi requirement to save.

Table: vim saving documents

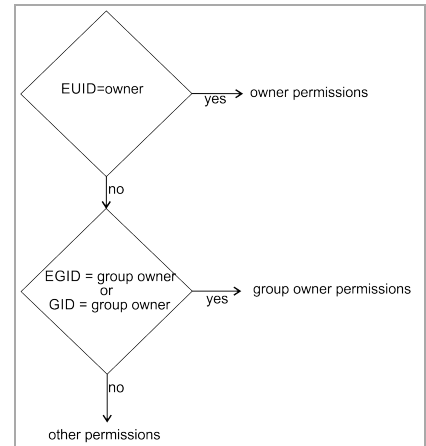
Key Sequence	Argument	Description
ESC +ZZ		write and quit
ESC +: +w		Write the contents of the buffer to the file currently open.
ESC +: +w	<file_name>	Write the contents of the buffer to the file indicated by <file_name>
ESC +: +w!		Override permission issues (root)
ESC +: +q		Quit . This may only be used if all buffer changes have been written to the file.
ESC +: +q!		Quit without writing buffer changes to file.

Permissions

File permissions (read, write and execute) are granted to permission sets (owner, group, other).

To determine which access permission a user will have the system compares the user's effective user and group ids, and secondary group memberships against the ownership permissions of the file.

The system ascertains the highest permission set the user belongs to and grants the access permissions assigned to that permission set. There are three sets of permissions



Permission Set	Description
owner	Sets permissions for users that have the same UID or EUID as the file's current owner.
group	Sets permissions for users that are members of the group that owns the file.
other	Permissions for all other users.

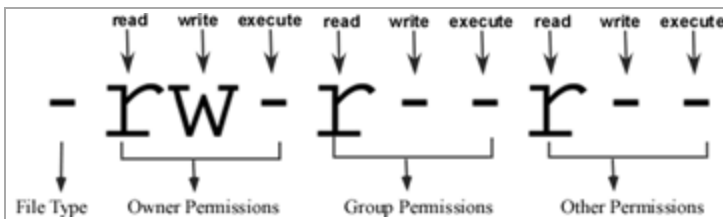
Symbolic Notation

Symbolic notation is a method of representing access permissions.

Symbolic notation uses the symbols:

- r (read)
- w (write)
- x (execute)

Each permission set has three columns .



Each column represents a column for each permission. If the symbol appears in the appropriate column the permission is granted for that permission set. If a tac (-) appears in the column the permission is not granted for that permission set.

Permission	Description
r	<p>Read privileges allow a user to read, copy or move a file</p> <p>You must have proper target directory privileges to move or copy a file.</p> <p>Directory read privileges allow the user to see what files are located in the directory (ls), but not the properties of the files.</p> <p>A non root user must have read and execute permissions to a directory to see the properties of files in a directory.</p>
w	<p>Write privileges to a file allow a user to add or remove content from a file .</p> <p>A non root user requires both write and execute permission to add or remove files from a directory.y.</p>
x	<p>Execute file permissions permit a user to execute a command or script. The file's magic number determines how the file is executed.</p>

Permission	Description
-	Permission is not granted

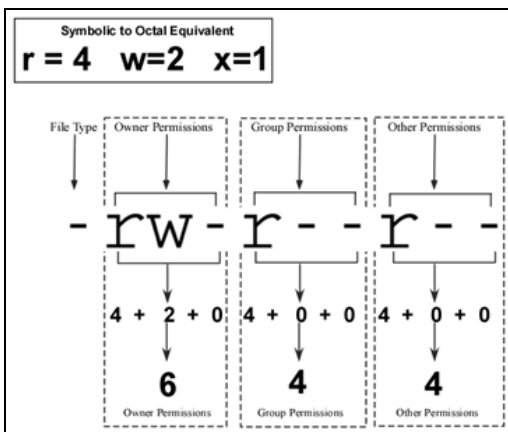
Octal Notation

Octal notation represents file permissions as a numeric value using a single column for each set. The numeric value of a permission set is equal to the sum of all access permissions granted to that permission set.

Each symbol is assigned an octal value.

- read = 4
- write = 2
- execute = 1

To convert access permissions we convert the value of each permission set. Therefore a permission set with rw- permissions would be 6 octal.



A file with the permissions rw-r-r would have an octal permission of 644.

cp

When a file is copied the original remains and the a new file is created. Therefore the inode of the new file will be different from that of the original.

Copying a file requires a user have read permissions to the file being copied and write and execute permissions to the directory which the file is being copied to.

Ownership of a copied file is the userid and primary group id of the user copying the file.

By default, permissions of the copied file are the same as the source file minus any permissions masked by the new owner's umask.

suid and sgid properties are not copied.

Any user may execute the command cp.

Syntax

```
cp <option(s)> <source file> <target file>
```

```
cp <option(s)> <file list> <target directory>
```

```
cp <option(s)> <source directory> <target directory>
```

Table: cp command options

Option	Argument	Description
	<source file>	File to be copied
	<target file>	File to be copied to
	<source directory>	Directory to be copied
	<target directory>	Directory to be copied to
	<file list>	List of files to be copied delimited by a space

Table: cp command options(continued)

Option	Argument	Description
-b		Make a backup copy of the existing file The backup file's filename will be followed by a tilde (~)
-i		Ask prior to overwriting an existing file.
-p		Preserves permissions, ownership and timestamps.
-R		Copies recursively (directories).
-u		Copies the source file only when the source file is newer than the target file or the target file does not exist.
--preserve=<attribute>		Preserves those attributes , specified by the argumet <attributes>. Multiple attributes must be comma delimited.
	mode	Preserves permissions
	ownership	Preserve ownership
	timestamps	Preserve timestamps
	links	Preserve symbolic links

mv

When a file is moved the source filename is changed (removing the original filename), but the ownership and permissions remain the same. When a file is moved the original file no longer exists.

If mv is moving a file within the same filesystem it creates a hard link (ln) to the target file and then removes (rm) the source file (inode numbers of source and target file will be the same).

If the file is being moved to a different filesystem the command copies the file to the target directory and then removes the source file therefore the inode is different than that of the original file.

Any user may execute the command mv.

Syntax

```
mv <option(s)> <source file> <target file>
```

```
mv <option(s)> <file list> <target directory>
```

```
mv <option(s)> <source directory> <target directory>
```

Table: mv command options

Option	Argument	Description
	<source file>	File to moved
	<target file>	File to be moved to
	<source directory>	Directory to be moved
	<target directory>	Directory to be moved to
	<source file>	File to moved
	<target file>	File to be moved to
-i		Ask prior to overwriting an existing file.
-n		Do not overwrite an existing file.

chmod

chmod stands for change mode. The word mode stands for permissions. Permissions may be changed using symbolic or octal notation.

Symbolic Method

chmod <option(s)> <permission set(s)> <operator> <permission(s)> <file name(s)>

<u>Set</u>	<u>Operator</u>	<u>Permissions</u>
U (ser)	+	r (ead)
g (roup)	-	W (rite)
O (ther)	=	(e) X (ecute)
a (ll)		

<permission set(s)>

<u>Permission</u>	<u>Description</u>
u	owner permission set
g	group permission set
o	other permission set
a	all permission sets

<operator>

defines whether or not to add or remove permissions

<u>Operator</u>	<u>Description</u>
+	add permission(s)
-	remove permissions
=	remove all current permissions and replace them with the following permissions.

<permissions>

<u>Symbol</u>	<u>Description</u>
r	read
w	write
x	execute

To add file permissions use the chmod command with a plus (+) operator. This example adds execute permissions to the group permission set.

```
chmod g+x <file name>
```

To remove permissions use the chmod command with the minus (-) operator. This example removes read permissions from the other permission set.

```
chmod o-r <file name>
```

To remove all existing access permissions and replace them use the = operator. The example removes all permissions for all permission sets and replaces them with all read permissions for all permission sets.

```
chmod a=r <file name>
```

Multiple changes may be made at once by making individual changes by separating individual changes with a comma (,) or grouping similar changes.

Changing permissions on a symbolic changes permissions on the file that it is linked to.

```
$ touch chmodsymlink1
$ ln -s chmodsymlink1 chmodsymlink2
$ ls -l chmodsymlink[12]
-rw-rw-r-- 1 student student 0 2007-10-03 05:14 chmodsymlink1
lrwxrwxrwx 1 student student 13 2007-10-03 05:14 chmodsymlink2 -> chmodsymlink1
$ chmod ug+x,ug-w,o-r chmodsymlink2
$ ls -l chmodsymlink[12]
-r-xr-x--- 1 student student 0 2007-10-03 05:14 chmodsymlink1
lrwxrwxrwx 1 student student 13 2007-10-03 05:14 chmodsymlink2 -> chmodsymlink1
```

Octal Method

No operators are required when changing file permissions using the octal method. It is easier to ascertain the resultant permissions you require and then execute the command.

Assume you wish the permissions of a file to allow read (octal value 4) and execute (octal value 1) for both the owner and group permission sets and no permissions for the other permission set. The permissions for both the owner and group permission sets equals 5 (4+1) and the other permission set would be equal to 0. Therefore the octal permissions are 550

```
chmod ug+x, ug-w, o=-r <file name>  
chmod 550 <file name>
```

chown

Ownership properties of a file are established when the file is initially created. The default owner of the file is the same as the effective userid, EUID, of the user of creating the file. The default group owner of the file is the same group as the effective group id, EGID of the user creating the file.

root privileges are required to change the ownership of a file.

Syntax

```
chown <option(s)> <user name | user id> <file name(s)>
```

```
chown <option(s)> <user name | user id>:<group name | gid> <file name(s)>
```

```
chown <option(s)> <user name | user id>.<group name | gid> <file name(s)>
```

Symbol	Argument	Description
-R		Recursive
	<user name user id>	username or user id of the new owner of the file.
	<group name gid>	groupname or gid of the new group which owns the file
	<file name(s)>	file name or list of file names to be changed

The example below illustrates changing both the owner and group owner of a file using a colon as a separator.

```
[root@FEDORA4 student1]# ls -l chowntest3
-rw-r--r-- 1 student1 student1 0 Nov  7 21:34 chowntest3
[root@FEDORA4 student1]# chown student4:root chowntest3
[root@FEDORA4 student1]# ls -l chowntest3
-rw-r--r-- 1 student4 root 0 Nov  7 21:34 chowntest3
```

The example below illustrates changing both the owner and group owner of a file using a period as a separator.

```
[root@FEDORA4 student1]# ls -l chowntest4
-rw-r--r-- 1 student1 student1 0 Nov  7 21:34 chowntest4
[root@FEDORA4 student1]# chown student4.root chowntest4
[root@FEDORA4 student1]# ls -l chowntest4
-rw-r--r-- 1 student4 root 0 Nov  7 21:34 chowntest4
```

If a user name (or uid) and a colon (:) or period (.) is used as a separator and no group name is supplied then the group ownership is changed to the primary group of the user name or user specified by the argument <username> or <userid>.

```
[root@FEDORA4 student1]# ls -l chowntest6
-rw-r--r-- 1 student1 student1 0 Nov  7 21:57 chowntest6
[root@FEDORA4 student1]# chown :student4 chowntest6
[root@FEDORA4 student1]# ls -l chowntest6
-rw-r--r-- 1 student1 student4 0 Nov  7 21:57 chowntest6
```

Glossary

A

absolute path

Specifies the path to a file from the system's root (/) directory (node). e.g. /etc/pam.d/passwd.

alias

An alias is a command shortcut used to automatically apply options and or arguments to a command.

alpha characters

A set of: upper and lower case characters, symbols, and punctuation marks

alphabet

Set of letters, or symbols used to write a language.

alphanumeric characters

A set of numbers, upper and lower case characters, symbols, and punctuation marks

ANSI

ANSI, American National Standards Institute, oversees the development of standards. The ANSI encoding standard extended the capabilities of the ASCII encoding standard.

argument

A command component which specifies the data source (input) to the command.

ASCII

ASCII, American Standard Code for Information Interchange, is a 7 bit character set originally designed for teletype machines.

B

basic regular expression

regular expression executed by grep command

bit

Also called a binary digit a bit stores a value of one or 0

builtin

Builtin commands are integrated into the shell and executed as part of the shell process. Builtin commands are sometimes referred to as internal commands.

byte

A unit of information consisting of 8 bits. The number of bits in a byte is not a standard value Although 8 bits to a byte is the expected value, the size of a byte may be hardware dependent.

C

case sensitive

The ability to distinguish between upper and lower case letters

character

A symbol used in a written language

character encoding

A process which maps a printable character to a binary value.

character set

A defined list of characters

code point

The numerical value assigned (mapped) to a character.

code unit

The number of bits used to store a character

command

A command is an instruct to perform a task.

command line interface

Text-based interface used to submit commands to the operating system.

command line interpreter

A program which reads a command, or sequence of commands from a command line, file, or other data stream and parses the command before submitting it to the operating system.and parses the command before submitting it to the operating system..

command substitution

Replaces an embedded command with the output of the embedded command.

compound command

A compound command consists of two or more commands separated by a control operator

concatenate

link together

control operator

A control operator determines the flow of a command line.

D**data boundary**

A character or series of characters used to separate data fields or words

data stream

A data stream is a sequence of characters transmitted between two devices

database

A table which contains records and fields.

delimiter

A character or string used to establish a data boundary

digit

Refers to the symbols 0 through 9

E**EBCDIC**

EBCDIC, Extended Binary Coded Decimal Interchange Code, is an 8 bit (1 byte) character set created by IBM. There were multiple incompatible versions of EBCDIC

effective userid

A temporary user identification number assigned to a user when the su command is executed or a file with suid privileges is accessed.

embedding

Embedding is the process of taking a section of code and making it an integral part of another section of code.

encoding

Encoding maps characters to their binary values

external command

External commands are executable binary files

F

field (database)

A field is a single unit of information associated with a database record

file descriptor

A file descriptor is a reference (also called a handle) used to access a file.

file descriptor table

A file descriptor table is created when most processes open. A unique index number is assigned to a file when it is opened by a process. This index number is linked to the file's inode. If a process may opens the same file multiple times using different permissions (modes) additional file descriptor numbers are assigned

flat file database

A flat file database is a database stored in a text file. Each line of the text file is a record and the record is divided into fields using a delimiter.

format specifier

A character or group of characters used to specify what fields are displayed in a command's output.

function

A function is a named section of a program which contains a series of commands designed to perform a specific task. Functions may be created on the command line.

H

hidden file

Files are hidden to prevent accidental modification by a user. A hidden file is any file whose filename is prefixed by a period (.) e.g. .bashrc. To list hidden files the -a option must be used with the ls command (ls -a).

I

identifier

A sequence of characters used to name a variable, account or file.

IFS

Internal Field Separator, is the environmental variable used to store those characters used as a delimiter

inode

A file system data structure which stores file ownership, timestamp and data location information.

integer

The set of whole numbers and their negative values (-3, -2, -1, 0, 1, 2, 3).

integer (data type)

A value stored as an integer data type represents a quantity which may be used in a mathematical expression.

irrational numbers

Numbers which cannot be expressed as a ratio (fraction).

K

keyword

A word which has special meaning to a program or operating system, which may be used as an identifier.

L

least significant bit

The least significant bit is the first digit on the right of a binary number

letter

A character which represents a basic sound in an alphabet

literal character

A character which is interpreted as its encoded value rather than any special meaning (meta character).

login session

The time between a user's logon and logoff

lowercase characters

Small letters e.g a. b. c

M

meta character

A character which has a special meaning to a program.

metacharacter

Metacharacters are special characters which have special significance to the shell interpreter

most significant bit

The most significant bit is the first digit on the left of a binary number

N

node

A link or connection point

number

Character which references a quantity.

O

octet

A unit of information consisting of 8 bits

one way hash

A binary encryption created from a text string. A one way hash is difficult to reverse.

operator

An operator is a character or symbol which determines how a command line is processed.

option

A command component which determines how the command is executed

P

POSIX

POSIX, Portable Operating System Interface, is an IEEE standard which defines application programming interfaces. This standard is used to create a program which will work in multiple operating systems.

precedence

A set of rules which determines the order in which an object is evaluated.

process

A process is an instance of a program.

program

A program is an ordered list of instructions, (commands) designed to

perform a specific task in a specific manner

prompt

A prompt is a text character used to indicate the shell is ready to accept user interaction.

R

rational numbers

Numbers created by dividing two numbers.

real numbers

The set of whole numbers, rational numbers, irrational numbers, and their negative values.

real userid

An identification number assigned to a user as part of the login process.

record

A database record is a unit of information which may contain one or more associated data fields.

regular expression

A series of characters which form a search pattern used to match character patterns in a command line or file.

Regular Expression

A regular expression is a string of characters used to define a pattern to search for within a file or files.

reserved word

A word which has special meaning to a program or operating system, which may not be used as an identifier

S

semantics

A set of rules which determines what operations may be performed and the order in which they are performed

shell

An interface which accepts and interprets user or application input for the operating system and receives and processes output from the operating system. A shell may also be called a command line interpreter or command line processor.

stderr

stderr is the data stream designator for error messages. The standard error device is the device or file unsuccessful results of a command are displayed on or written to. The default standard output device, is the console monitor.

stdin

stdin designates the source data stream (input) for command line programs. The default input device is the system keyboard.

stdout

stdout is the data stream designator for non error messages. The standard output device is the device or file successful results of a command are displayed on or written to. The default standard output device, is the console monitor.

string

A series of characters.

symbol

A symbol (glyph) is a written mark which represents an idea.

symbolic link

A symbolic link (also called soft link) is a file or directory which points to another file or directory.

word

A series of printable characters (string) terminated by a delimiter.

syntax

Syntax rules evaluate the elements of a command line to determine if they are valid and presented in the proper format. and order

T

terminal session

Time between a user opening a terminal and closing a terminal

U

uppercase characters

Capital letters. e.g. A B C

UTF

Character set which supports one to four eight bit code units

V

variable expansion

The process of substituting the value of a variable with a word (or word list) value.

W

white space character

A character which occupies space, but is not printable.

whole number

The set of natural numbers (0,1,2,3...). Natural numbers do not include fractions.

Index

'	
''	69
''	
""	69
\$	
\$LANG	23
/	
/etc/shells	42
/proc/<pid>/fd	75
\	
\	25
<	
<<	77
>	
>	78
>&	81
>>	78
2	
2>	79
2>>	79
A	
absolute path	32
alpha character	17
alphabet	17
alphanumeric character	18
argument	65
arithmetic expansion	70
ASCII	21
ASCII, extended	21
B	
Bash Shell	42
bit	20
Bourne Shell	41
byte	20
C	
C Shell	41
case sensitive	19
cat	34, 50
cat -n	51
cat -s	51
cd	33
certification reference	
103.8 -- vi	103
character	17
character, literal	24
chsh	43
code point	20
code unit	20
Commands	
cat	34, 50
cat -n	51
cat -s	51
cd	33
chsh	
chsh -l	43
chsh -s	43
date	37, 58
date -r	59
date -s	58
date -u	59

echo	36, 52	uname -r	60
echo -e	52	uname -s	60
echo -E	52	uname -v	60
echo -n	52		
grep	86	D	
grep -c	89	data stream	
grep -i	88	stderr	74, 79
grep -n	89	stdin	74
grep -r or -R	91	stdout	74, 78
grep -v	89	date	37, 58
grep -w	90	date -r	59
ls	34	date -s	58
ls -a	49	date -u	59
ls -d	48	date+%	64
ls -i	48	format descriptor	63
ls -l	48	date math	59
ls -n	49	delimiter	27
ls -R	50	IFS	28
lsuf	75	digit	16
mkfifo	82	E	
mknod	82	EBCDIC	21
ps	37, 53	echo	36, 52, 68
ps -a	54	echo -e	52, 68
ps -e	54	echo -E	52
ps -f	56	echo -n	52
ps -l	57	egrep	94
ps -t	56	embedding	70
ps a	54	encoding	20
pwd	32	code unit	20
tac	51	encoding format	23
tee	83	escape sequences	
tty	38	\n	68
uname	38	\r	68
uname -i	61	\t	68
uname -n	61	\v	68
uname -o	61		
uname -p	61		

F

field	27
file descriptor	74
file descriptor table	74
flat file database	27
format descriptor	63
format specifier	62

G

grep	86
------------	----

I

IFS	28
-----------	----

K

keyword	30
Korn Shell	42

L

least significant bit	20
letter	17
lower case	19
LPI 101 & LX0-103 by Domain	
103.7 Using Regular Expressions	
grep	86
ls	34
ls -a	49
ls -d	48
ls -i	48
ls -l	48
ls -n	49
ls -R	50
lsof	76

M

meta character	24
mkfifo	82
mknod	82
most significant bit	20

N

named pipe	82
number	16
integer	17
irrational	17
real	17
whole	16

O

octet	20
option	66
options	
BSD	47
long form	46
short form	46

P

pipe	82
named pipe	82
unnamed pipe	82
ps	37, 53
format descriptor	63
format specifier	62
ps -a	54
ps -e	54
ps -f	56
ps -l	57
ps -t	56
pwd	32

Q

quoting	
double quotes	69
single quotes	69

R

record	27
redirection	
stderr	79
stdout	78

S

shell	41
Bash	42
Bourne	41
C	41
Korn	42
Z	42
stderr	79
stdout	78
string	27
symbol	16

T

tac	51
tee	83
tee -a	83
tty	38

U

uname	38
uname -n	61
uname -o	61
uname -p	61
uname -r	60

uname -s	60
----------------	----

uname -v	60
----------------	----

unnamed pipe	82
--------------------	----

upper case	19
------------------	----

UTF	22
-----------	----

V

variable expansion	70
--------------------------	----

vi	103
----------	-----

W

white space character	28
-----------------------------	----

word	29
------------	----

keyword	30
---------------	----

Z

Z Shell	42
---------------	----

This page is intentionally blank