



Capgemini

Module 5 – Working with the Command Line



Capgemini



Module Learning Objectives

Upon completion of this module, the student should be able to do the following:

- Understand the essential commands for moving around and working in the LINUX command line.
- Understand how to create and edit a document in the LINUX command line using text editors.
- Understand how scripting works and how it can be used to help automate routine tasks such as log analysis.



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 3

Command Line Basics



Capgemini

Foundational Analyst Security Training

The next section will cover working with directories and files.



Agenda



**BASIC LINUX COMMANDS | INPUT/OUTPUT (I/O) STREAMS |
OUTPUT REDIRECTION | PIPES**

- Basic Linux Commands
- Input / Output (I/O) Streams
- Output Redirection
- Pipes



Topic Learning Objectives

Upon completion of this topic, the student should be able to do the following:

- Understand the basics of the LINUX command line and how to access additional information on each command.
- Use the LINUX command line to determine the location in the file structure and move around in the file structure.
- Understand how to create and move files and folders using the LINUX command line.



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 6

Upon successful completion of this module, students will be able to:

- Apply Unix tools and techniques to efficiently identify trends and extract indicators from large data sources
- Properly use basic Linux commands
- Use Linux commands for Input/Output (I/O) streams, output redirection, and pipes



Terminus by MIT...

Terminus is a flash-based, LINUX Command Line Simulator that is great to practice with!

Feel free to play
with Terminus when
we have downtime.

Welcome! If you are new to the game, here are some tips:
Look at your surroundings with the command "ls".
Move to a new location with the command "cd LOCATION"
You can backtrack with the command "cd .".
Interact with things in the world with the command "less ITEM"
If you forget where you are, type "pwd"
Go ahead, explore. We hope you enjoy what you find. Do ls as your first command.
>



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 7

Command Line Interface (CLI) Basic Commands



Command: man

- Built-In Reference Manuals
 - man [OPTION(S)] keyword(s)
- Common Options
 - -a: Display manual pages that match keyword(s)
 - -t: Postscript formatting
 - -k: Filter by keyword
 - -i: Not case sensitive (default)
 - -I: Case sensitive
- Searching within man
 - "/<string>": Search forward
 - "?<string>": Search backward
- Example
 - # man vim

Also get help from -- **help**, which does not go into as much depth as **man** but can still be helpful.



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 8

The next command is the man command.

The man command will display a built-in reference manual.

The syntax for the man command is: man [OPTION(S)] keyword(s)

Common Options include:

- -a: Display manual pages that match keyword(s)
- -t: Postscript formatting
- -k: Filter by keyword
- -i: Not case sensitive (default)
- -I: Case sensitive

You can search within the man command using:

"/<string>": to search forward

"?<string>": to search backward

An example is:

man vim



CLI Basic Commands (cont.)

Command: ls

- Lists Directory Contents and Other File Information
 - ls [OPTION(S)]...[FILE]...
- Common Options
 - -a: List all files (includes hidden files)
 - -l: Long listing format
 - -t: Sorts by file modification times
 - -h: Human readable display
 - -r: Reverse order sorting
 - -S: Sort by file size
- Examples
 - # ls -l
 - # ls -a
 - # ls -al

```
ls(1)                               User Commands                               ls(1)

NAME
    ls - list directory contents

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

DESCRIPTION
    list information about the FILEs (the current directory by default). Sort entries alphabetically if none of -efts-
    [-vVux] nor --sort is specified.

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

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

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

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

    -b, --escape
        print C-style escapes for nongraphic characters

    -block-size=SIZE
        scale sizes by SIZE before printing them; e.g., '--block-size=M' prints sizes in units of 1,048,576 bytes;
        see SIZE format below

    -B, --ignore-backups
        do not list implied entries ending with ~

    -c          with -lt: sort by, and show, ctime (time of last modification of file status information); with -l: show
               ctime and sort by name; otherwise: sort by ctime, newest first

    -C          list entries by columns

    -color[when]
        colorize the output; WHEN can be 'always' (default if omitted), 'auto', or 'never'; more info below

    -d, --directory
        list directories themselves, not their contents

    -D, --dirend
        Manual page ls(1) line 3 (press h for help or q to quit)

Manual page ls(1) line 3 (press h for help or q to quit)
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 9

The ls command is used to list directory contents and other file information.

The syntax for the ls command is: ls [OPTION(S)]...[FILE]...

Common Options include:

- -a: List all files (includes hidden files)
- -l: Long listing format
- -t: Sorts by file modification times
- -h: Human readable display
- -r: Reverse order sorting
- -S: Sort by file size

Examples of using the ls command are:

- # ls -l to list files in a long listing format
- # ls -a to list all files including hidden files
- # ls -al to list all files in a long listing format

CLI Basic Commands (cont.)



Command: cd

- Change Directory
 - cd [OPTION(S)] [DIRECTORY]
- Common Options
 - .. : Move up one folder structure
 - ~: User's home directory
 - -P: Physical directory structure without following symbolic links
 - -L: Moves to directory the symbolic link points to
- Examples
 - # cd ..
 - # cd /documents
 - # cd ~/documents

The screenshot shows a terminal window with the following text:

```
File Edit View Search Terminal Help
No manual entry for cd
bzrtf80mghbz1f80mgh-Virtual-Machine:$ man cd
bzrtf80mghbz1f80mgh-Virtual-Machine:$ cd
bzrtf80mghbz1f80mgh-Virtual-Machine:$ ls
analysis Lab Files securityonion-kibana.desktop Videos
Desktop Documents Pictures securityonion-squirt.desktop
Downloads Public Templates
bzrtf80mghbz1f80mgh-Virtual-Machine:$ cd Desktop
bzrtf80mghbz1f80mgh-Virtual-Machine:~/Desktop$ ls
Captures securityonion-reader.desktop securityonion-squid.desktop
bzrtf80mghbz1f80mgh-Virtual-Machine:~/Desktop$
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 10

The cd command is used to change the directory.

The syntax for the cd command is: cd [OPTION(S)] [DIRECTORY]

Common Options include:

- .. : Move up one folder structure
- ~ : User's home directory
- -P : Physical directory structure without following symbolic links
- -L : Moves to directory the symbolic link points to

Examples using the cd command are:

- # cd .. to move up one folder structure
- # cd /documents to go to the documents folder
- # cd ~/documents to go to the user's home directory's documents folder

CLI Basic Commands (cont.)



Command: pwd

- Print Working Directory (full filename)
 - `pwd [OPTION]...`
- Common Options
 - `-L`: Include symbolic links
 - `-P`: Avoid symbolic links
- Examples
 - `# pwd`
 - `# pwd -L`
 - `# pwd -P`

```
Pwd(1)                               User Commands                               Pwd(1)
NAME
    pwd - print name of current/working directory
SYNOPSIS
    pwd [OPTION]...
DESCRIPTION
    Print the full filename of the current working directory.
    -L, --logical
        use PWD from environment, even if it contains symlinks
    -P, --physical
        avoid all symlinks
    -h, --help
        display this help and exit
    --version
        output version information and exit
    If no option is specified, -P is assumed.
    NOTE: your shell may have its own version of pwd, which usually supersedes the version described here. Please
          refer to your shell's documentation for details about the options it supports.
AUTHOR
    Written by Jim Meyering.
REPORTING BUGS
    GNU coreutils online help: <http://www.gnu.org/software/coreutils/>
    Report pwd translation bugs to <http://translationproject.org/team/>
COPYRIGHT
    Copyright © 2015 Free Software Foundation, Inc. License GPLv3+: GNU GPL version 3 or later
    http://gnu.org/licenses/gpl.html.
    This is free software: you are free to change and redistribute it. There is NO WARRANTY, to the extent permitted
    by law.
SEE ALSO
    getcwd(3)
    Full documentation at: <http://www.gnu.org/software/coreutils/pwd>
    or available locally via: info '(coreutils) pwd invocation'
    Manual page pwd(1) line 1 (press h for help or q to quit)
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 11

The `pwd` command is used to print the working directory, showing the full path.

The syntax for the `pwd` command is: `pwd [OPTION]...`

Common Options include:

- `-L`: Include symbolic links
- `-P`: Avoids symbolic links

Examples using the `pwd` command are:

- `# pwd` to print (or display) the working directory
- `# pwd -L` to print the working directory including symbolic links
- `# pwd -P` to print the working directory avoiding symbolic links

CLI Basic Commands (cont.)



Command: file

- Determines File Type
 - file [OPTIONS(S)] [FILE]
- Common Options
 - -b: Do not prepend filename to output line
 - -k: Do not stop at first match
- Example
 - # file audit.txt

```
file(1)                                BSD General Commands Manual                               file(1)

NAME
    file -- determine file type

SYNOPSIS
    file [-bEhiklLmoprsvzZ] [-apple] [-extension] [-mine-encoding] [-mine-type] [-e basename] [-# separator]
    file [-f namefile] [-m magicfiles] [-P namevalue] file ...
    file [-c] [-n] [-s] [-t] [-v] [-x] [-y] [-z] [-h]
    file [-?]

DESCRIPTION
This manual page documents version 5.29 of the file command.

file tests each argument in an attempt to classify it. There are three sets of tests, performed in this order: filesystem tests, magic tests, and language tests. The first test that succeeds causes the file type to be printed. The type printed will probably contain one of the words text (the file contains only printing characters and a few common control characters and is probably safe to read on an ASCII terminal), executable (the file contains the machine code for a particular architecture), data (the file contains no characters and is probably safe to read on an ASCII terminal), or binary (data is usually "binary" or non-printable). Exceptions are well-known file formats (core files, tar archives) that are known to contain binary data. When adding local definitions to /etc/magic, make sure to preserve these keywords. Note that all file reading done in this directory have the word "text" printed. Don't do an Berkeley-style did and change "shell commands text" to "shell script".

The filesystem tests are based on examining the return from a stat(2) system call. The program checks to see if the file is empty, or if it's some sort of special file. Any known file types appropriate to the system you are running (such as FIFOs) on those systems that implement them) are intuited if they are defined in the system header file <sys/stat.h>.

The magic tests are used to check for files with data in particular fixed formats. The canonical example of this is a binary executable (compiled program) a.out file, whose format is defined in <elf.h>, <a.out.h> and possibly <exec.h> in the standard include directory. These files have a file type stored in a particular place in the file header. If the file has the UNIX operating system type, then the file is a UNIX executable or one of several types thereof. The concept of a "magic" has been applied by extension to data files. Any file with some invariant sequence of bytes can be designated as a magic file and the associated magic file /usr/share/misc/magic.mgc, or the files in the directory /usr/share/misc/magic if the compiled file does not exist. In addition, if $HOME/.magic.mgc or $HOME/.magic exists, it will be used in preference to the system magic files.

If a file does not match any of the entries in a magic file, it is examined to see if it seems to be a text file, ISO-8859-1 encoded text, ISO-8859-1 extended ASCII character such as those used on Macintosh and IBM PC systems), UTF-8-encoded Unicode, UTF-16-encoded Unicode, and EBCDIC character sets can be distinguished by the different ranges of sequences of bytes that constitute primary characters in each set. If a file passes any of these tests, its character set is reported as ASCII, ISO-8859-1, UTF-8, and extended-ASCII files are identified as "text" because they will
Manual page file(1) line 1 (press h for help or q to quit)
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 12

The file command is used to determine the file type.

The syntax for the file command is: file [OPTIONS(S)] [FILE]

Common Options include:

- -b: Do not prepend filename to output line
- -k: Do not stop at first match

An example using the file command would be:

- # file audit.txt

CLI Basic Commands (cont.)



Command: find

- Find files, and allow command execution on results
 - find [OPTION(S)] [PATH] [EXPRESSION]
- Common Options
 - -L: Follow symbolic links
 - -P: Do not follow symbolic links (default)
 - -H: Do not follow symbolic links, except when processing the command line arguments
- Expressions
 - -name pattern: Find files with pattern in filename
 - -size [+/-]n: Find files greater than (+)/smaller than (-) n size
 - -type: Find files of a specified type (such as f = regular file, d = directory)
- Actions
 - -print: Print the path name of the current file to stdout
 - -exec <command>: Execute a command on the files found by the find search
- Example – Find files that are larger than 25 megabytes (k – kilobyte, M – Megabyte, G – Gigabyte).
 - # find / -type f -size +25M -exec ls -lh {} \;

```
FIND(1)                                General Commands Manual                               FIND(1)

NAME
       find - search for files in a directory hierarchy

SYNOPSIS
       find [-H] [-L] [-P] [-O debugopts] [-Olevel] [starting-point...] [expression]

DESCRIPTION
       This manual page documents the GNU version of find.  GNU find searches the directory tree rooted at each given
       starting-point evaluating the given expression to determine whether the file or directory in question matches.  Rule of precedence (see
       section OPERATORS) until the outcome is known (if the left hand side is false for AND operations, true for OR), at
       which point find moves on to the next file name.  If no starting-point is specified, '.' is assumed.

       If you are using find in an environment where security is important (for example if you are using it to search
       directories that are writable by other users), you should read the "Security Considerations" chapter of the findutils
       manual page.  It also contains the findxattr command with more information.  There is also a lot more detail and discussion than this manual page, so you may find it a more useful source of information.

OPTIONS
       The -M, -I and -P options control the treatment of symbolic links.  Command-line arguments following these are
       taken to be names of files or directories, except that if the first argument begins with a slash, it is taken to be the
       argument ('.' or '/').  That argument and any following arguments are taken to be the expression describing what is
       to be searched for.  If no paths are given, the current directory is used.  If no expression is given, the expression
       -print is used (but you should probably consider using -print0 instead, anyway).

       This manual page talks about options within the expression language.  These options control the behaviour of find but
       are specific immediately after the last argument to the find command.  -B, -P, -D and -O must appear
       before the first path name, if at all.  A double dash -- can also be used to signal that any remaining arguments
       are options.  Note that all start points begin with either '.' or '/' generally safer if you use
       wildcards in the list of start points.

       -P
           Never follow symbolic links.  This is the default behaviour.  When find examines or prints information a
           file, and the file is a symbolic link, the information used shall be taken from the properties of the sym-
           bolic link itself.

       -L
           Follow symbolic links.  When find examines or prints information about files, the information used shall be
           taken from the properties of the file to which the link points, not from the link itself (unless it is a
           broken symbolic link).  find will not examine the file to which the link points if this option
           implies -noleaf.  If you later use the -P option, -noleaf will still be in effect.  If -k is in effect and
           find finds a symbolic link to a subdirectory during its search, the subdirectory pointed to by the sym-
           bolic link will be searched.

       When the -L option is in effect, the -type predicate will always match against the type of the file that a
       symbolic link points to rather than the link itself (unless the symbolic link is broken).  Actions that can
       change the type of a file will not work correctly if this option is in effect.
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 13

The find command is used to find files and allow command execution on results.

The syntax for the find command is: `find [OPTION(S)] [PATH] [EXPRESSION]`

Common Options include:

- -L: Follow symbolic links
- -P: Do not follow symbolic links (default)
- -H: Do not follow symbolic links, except when processing the command line arguments

Additionally, there are expressions and actions that you can define with the file command.

Expressions include:

- -name pattern: Find files with pattern in filename
- -size [+/-]n: Find files greater than (+) / smaller than (-) than n size
- -type type: Find files of a specified type (such as f = regular file, d = directory)

Actions include:

- -print: Print the path name of the current file to stdout
- -exec <command>: Execute a command on the files found by the find search

An example using the file command to find files that are larger than 25 megabytes (note that k is used for kilobyte, M is used for Megabyte, and G is used for Gigabyte):

- # find / -type f -size +25M -exec ls -lh {} \;

CLI Basic Commands (cont.)



Command: chmod

- Change permissions of files or directories
 - chmod [OPTION(s)] [PERMISSIONS] [FILE]
 - Permissions can be alphanumeric or octal
- Common Options
 - -c: Verbose output only when a change is made
 - -f: Suppress most error messages
 - -R: Change files and directories recursively
 - -v: Verbose mode – output diagnostic for every file processed
- Example – Users can read, write, and execute; group can read and execute, and others can only read
 - # chmod u=rw,g=rx,o=r *.txt
 - # chmod 754 *.txt

Octal Permissions

- 4 – read
- 2 – write
- 1 – execute

400	read by user
040	read by group
004	read by other
200	write by user
020	write by group
002	write by others
100	execute by user
010	execute by group
001	execute by others

Common Modes

400	-r-----
644	-rw-r--r--
755	-wxr-xr-x
777	-rwxrwxrwx



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 14

The chmod command is used to change the permissions of files or directories.

The syntax for the chmod command is: chmod [OPTION(s)] [PERMISSIONS] [FILE]

The permissions can be alphanumeric or octal.

Octal Permissions include:

- 4 – read
- 2 – write
- 1 – execute
- 400 read by user
- 040 read by group
- 004 read by other
- 200 write by user
- 020 write by group
- 002 write by others
- 100 execute by user
- 010 execute by group
- 001 execute by others

Common Modes include:

- 400 -r-----
- 644 -rw-r--r--
- 755 -wxr-xr-x
- 777 -rwxrwxrwx

Common Options include:

- -c: Verbose output only when a change is made
- -f: Suppress most error messages
- -R: Change files and directories recursively
- -v: Verbose mode - output diagnostic for every file processed

An example of the chmod command where users can read, write, and execute; group can read and execute; and others can only read:

- # chmod u=rw,g=rx,o=r *.txt
- # chmod 754 *.txt

CLI Basic Commands (cont.)



Command: chown

- Change Ownership (files and directories)
 - chown [OPTION(S)] [FILE]
- Common Options
 - -R: Operate recursively
 - -f: Suppress most error messages
 - -v: Output diagnostic for each file
- Ownership
 - user: Name of user to own file
 - user:group: Name of user and group to own file
 - :group: Name of group to own file
- Example – Change the file ownership of “audit.log” to user “jsmith” and group “management”
 - # chown jsmith:management audit.log

```
chown(1)                               User Commands                               chown(1)

NAME
    chown - change file owner and group

SYNOPSIS
    chown [OPTION]... [OWNER[:GROUP]] FILE...
    chown [OPTION]... --reference=RFILE FILE...

DESCRIPTION
    manual page documents the GNU version of chown. chown changes the
    user and/or group ownership of each given file. If only an owner (a
    user name or numeric user ID) is given, that user is made the owner of
    each file. If both an owner and a group (or numeric group ID) are
    given, the owner of each file is changed to the given owner and the
    group of each file is changed to the given group. If only a group is
    given, the owner of each file is unchanged. If neither the owner nor the
    group is given, the group of each file is unchanged. If both the owner
    and the group are given, the owner is omitted, only the group of the files is
    changed. In this case, chown performs the same function as chgrp. If only a colon is given,
    or if the entire operand is empty, neither the owner nor the group is
    changed.

OPTIONS
    Change the owner and/or group of each FILE to OWNER and/or GROUP. With
    --reference, change the owner and group of each FILE to those of RFILE.
    -c, --changes
        like verbose but report only when a change is made
    -f, --silent, --quiet
        suppress most error messages
    -v, --verbose
        output a diagnostic for every file processed
    --dereference
        affect the referent of each symbolic link (this is the default),
        rather than the symbolic link itself
    -h, --no-dereference
        affect symbolic links instead of any referenced file (useful
        only on systems that can change the ownership of a symlink)

Manual page chown(1) line 1 (press h for help or q to quit)
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 15

The chown command is used to change Ownership of files and directories.

The syntax for the chown command is: chown [OPTION(S)] [FILE]

Common Options include:

- -R: Operate recursively
- -f: Suppress most error messages
- -v: Output diagnostic for each file

Ownership can be of a user or group:

- user: Name of user to own file
- user:group: Name of user and group to own file
- :group: Name of group to own file

An example of the chown command to change the file ownership of “audit.log” to user “jsmith” and group “management”:

- # chown jsmith:management audit.log

CLI Basic Commands (cont.)



Command: dd

- Convert and copy a file
 - dd [OPTION(S)]
- Common Options
 - -if=file: Input file
 - -of=file: Output file
 - -bs=BYTES: Read and write size of blocks
 - -count=n: Number (n) of blocks to read and write
- Example – Create an image of /dev/sda in current user's home directory:
 - # dd if=/dev/sda of=~/disk1.img

```
00(1)                               User Commands                         00(1)
NAME      dd - convert and copy a file
SYNOPSIS  dd [OPERAND]...
          dd OPTION
DESCRIPTION
  Copy a file, converting and formatting according to the operands.
  bs=BYTES
    read and write up to BYTES bytes at a time
  cbs=BYTES
    convert BYTES bytes at a time
  conv=CONVS
    convert the file as per the comma separated symbol list
  count=N
    copy only N input blocks
  ibs=BYTES
    read up to BYTES bytes at a time (default: 512)
  if=FILE
    read from FILE instead of stdin
  iflag=FLAGS
    read as per the comma separated symbol list
  obs=BYTES
    write BYTES bytes at a time (default: 512)
  of=FILE
    write to FILE instead of stdout
  oflag=FLAGS
    write as per the comma separated symbol list
  seek=N skip N obs-size blocks at start of output
  skip=N skip N ibs-sized blocks at start of input
  Manual page dd(1) line 3 (press h for help or q to quit)
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 16

The dd command is used to convert and copy a file.

The syntax for the dd command is: dd [OPTION(S)]

Common Options include:

- -if=file: Input file
- -of=file: Output file
- -bs=BYTES: Read and write size of blocks
- -count=n: Number (n) of blocks to read and write

An example using the dd command to create an image of /dev/sda in current user's home directory:

- # dd if=/dev/sda of=~/disk1.img

CLI Basic Commands (cont.)

Command: tar

- Create, maintain, modify, and extract files archived in the tar format (.tar, .tar.gz, .tar.bz2)
 - tar [FUNCTION] [OPTIONS] [ARCHIVE] [FILE(S)]
- Common Functions
 - A: Append tar files to archive
 - C: Create new archive
 - --delete: Delete from archive
 - r: Append files to the end
 - u: Append files newer than the file in the archive
- Common Options
 - -c: Create an archive
 - -x: Extract an archive
 - -f: File mode
 - -v: Verbose
 - -z: gzip
 - -j: bzip2
- Example – Extract a .tar file to the current directory:
 - # tar -xvf archive1.tar

```
TAR(1)                                BSD General Commands Manual                               TAR(1)

NAME  tar -- The GNU version of the tar archiving utility

SYNOPSIS
tar [-] A -cappendate -concatenate | c --create | d -diff --compare | --delete | r --append | t --list |
--test-label | u --update | x --extract -g[et] [options] [archive ...]

DESCRIPTION
Tar stores and extracts files from a tape or disk archive.

The first argument to tar should be a function; either one of the letters A, d, t, or u, or one of the long function names. The short-letter functions can be combined with each other, and can be combined with other single-letter options. A long function name must be prefixed with --. Some options take a parameter; with the single-letter form these must be given as separate arguments. With the long form, they may be given by appending =value to the option.

FUNCTION LETTERS
Main operation mode:
-A, --catenate, --concatenate
    append tar files to an archive
-C, --create
    create a new archive
-D, --diff, --compare
    find differences between archive and file system
--delete
    delete from the archive (not on mag tapes!)
-F, --append
    append files to the end of an archive
-T, --list
    list the contents of an archive
--test-label
    test the archive volume label and exit
-U, --update
    only append files newer than copy in archive
-X, --extract, --get
    Manual page tar(1) line 1 (press h for help or q to quit)

Manual page tar(1) line 1 (press h for help or q to quit)
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 17

The tar command is used to create, maintain, modify, and extract files archived in the tar format (.tar, .tar.gz, .tar.bz2).

Tar files are not in themselves compressed, but can be compressed as a function of the command.

CLI Basic Commands (cont.)



Command: unzip

- List or extract a compressed .zip archive
 - **unzip [OPTION(S)] [ARCHIVE] [DESTINATION]**
- Common Options
 - **-l:** List archive files
 - **-v:** Verbose
 - **-c:** Extract files to stdout
 - **-u:** Update existing files (create new file if one does not exist)
- Examples
 - **# unzip accounting.zip**
 - **# unzip -l accounting.zip**

```
UNZIP(1)                               General Commands Manual                               UNZIP(1)

NAME
       unzip - list, test and extract compressed files in a ZIP archive

SYNOPSIS
       unzip [-Z] [-cfptvuv[abjnoeqCDKLmVWwS[:*]] file[.zip] [file1 ...] [-x file[s] ...] [-e file[s]]

DESCRIPTION
       unzip will list, test, or extract files from a ZIP archive, commonly found on MS-DOS systems. The default behavior (with no options) is to extract the current directory (or directory below it) all files in the specified archive. The compression program PKZIP creates ZIP archives; both are compatible with archives created by PKWARE's PKZIP and PKUNZIP for MS-DOS, but in many cases the program options or default behaviors differ.

ARGUMENTS
       [file[.zip]]
           Path of the ZIP archive(s). If the file specification is a wildcard, each matching file is processed in an order determined by the operating system (or file system). Only the filename can be a wildcard; the path itself cannot. Wildcard expressions are similar to those supported in commonly used Unix shells (sh, ksh, csh) and may contain:  
       *      matches a sequence of 0 or more characters  
       ?      matches exactly 1 character  
       [...]  matches any single character found inside the brackets; ranges are specified by a beginning character, a hyphen, and an ending character. If an exclamation point or a caret (^ or ~) follows the left bracket, then the range of characters within the brackets is complemented (that is, anything except what is enclosed in the brackets is considered a match). To specify a verbatim left bracket, the three-character sequence "[[" has to be used.  
       (Be sure to quote any character that might otherwise be interpreted or modified by the operating system, particularly under Unix and VMS.) If no matches are found, the specification is assumed to be a literal filename; and if that also fails, the suffix .zip is appended. Note that self-extracting ZIP files are supported, as with any other ZIP archive; just specify the .exe suffix (if any) explicitly.  
       ([file1 ...])
           An optional list of archive members to be processed, separated by spaces. (VMS versions compiled with VMSCL1 defined list delimit files with commas instead. See -v in OPTIONS below.) Regular expressions (wildcards) can be used to name multiple members; see above. Again, be sure to quote expressions that would otherwise be expanded or modified by the operating system.  
       ([-x file[s]])
           An optional list of archive members to be excluded from processing. Since wildcard characters normally  
       Manual page UNZIP(1) line 3 (press h for help or q to quit)
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 18

The **unzip** command is used to list or extract a compressed .zip archive.

The syntax for the **unzip** command is: **unzip [OPTION(S)] [ARCHIVE] [DESTINATION]**

Common Options include:

- **-l:** List archive files
- **-v:** Verbose
- **-c:** Extract files to stdout
- **-u:** Update existing files (create new file if one does not exist)

Examples

- **# unzip accounting.zip** to unzip the accounting.zip file
- **# unzip -l accounting.zip** to unzip the accounting.zip file, using the list archive files option

CLI Basic Commands (cont.)



Command: cat/zcat

- Concatenate files and print files (zcat to compress or extract)
 - cat [OPTION(S)] [filename]
 - zcat [OPTION(S)] [compressed filename]
- Common Options
 - -n: Number output lines
 - -b: Same as -n, except to ignore blank lines
 - -s: Skip files not found
- Examples
 - # cat File1.txt
 - # cat File1.txt File2.txt > Files1_2.txt

```
CAT(1)                                         User Commands                                         CAT(1)
NAME
    cat - concatenate files and print on the standard output
SYNOPSIS
    cat [OPTION]... [FILE]...
DESCRIPTION
    Concatenate FILE(s) to standard output.
    With no FILE, or when FILE is -, read standard input.
    -A, --show-all
        equivalent to -vET
    -b, --number-nonblank
        number nonempty output lines, overrides -n
    -e, --show-ends
        display $ at end of each line
    -n, --number
        number all output lines
    -s, --squeeze-blank
        suppress repeated empty output lines
    -t, --show-tabs
        equivalent to -vT
    -T, --show-tabs
        display TAB characters as ^I
    -u
        (ignored)
    -v, --show-nonprinting
        use ^ and M- notation, except for LF and TAB
    --help
        display this help and exit
    --version
        output version information and exit
Manual page cat(1) line 1/1 50% (press h for help or q to quit)
```



Use > to overwrite and >> to append.



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 19

The cat/zcat commands are used to concatenate files and print files (use zcat to compress or extract).

The syntax for the cat/zcat command are:

- cat [OPTION(S)] [filename]
- zcat [OPTION(S)] [compressed filename]

Common Options include:

- -n: Number output lines
- -b: Same as -n, except to ignore blank lines
- -s: Skip files not found

Examples using the cat command include:

- # cat File1.txt to print file1.txt
- # cat File1.txt File2.txt > Files1_2.txt to concatenate file1.txt and file2.txt and send the output to files1_2.txt

NOTE: use > to overwrite and >> to append

CLI Basic Commands (cont.)



Command: diff

- Find the difference between two files
 - diff File1 File2
- Example
 - # diff File1.txt File2.txt

```
diff(1)                               User Commands                               diff(1)

NAME
    diff - compare files line by line

SYNOPSIS
    diff [OPTION]... FILES

DESCRIPTION
    Compare FILES line by line.

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

    --normal
        output a normal diff (the default)
    -q, --brief
        report only when files differ
    -s, --report-identical-files
        report when two files are the same
    -c, -C NUM, --context[=NUM]
        output NUM (default 3) lines of copied context
    -u, -U NUM, --unified[=NUM]
        output NUM (default 3) lines of unified context
    -e, --ed
        output an ed script
    -n, --rcs
        output an RCS format diff
    -y, --side-by-side
        output side-by-side in two columns
    -M, --width=NUM
        output at most NUM (default 130) print columns
    --left-column
        output only the left column of common lines
    --suppress-common-lines
        do not output common lines

Manual page diff(1), line 1 (press h for help or q to quit)
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 20

The diff command is used to find the difference between two files.

The syntax for the diff command is: diff File1 File2

An example using the diff command:

- # diff File1.txt File2.txt

CLI Basic Commands (cont.)



Command: wc

- Print the number of new lines, words, and/or bytes in a file
 - wc [OPTION(S)] [FILENAME]
- Common Options
 - -c: Print byte count
 - -m: Print character count
 - -l: Print new line count
 - -w: Print word count
- Example
 - # wc -lwc File1.txt

The screenshot shows a terminal window with the title 'wc(1)'. The window displays the man page for the 'wc' command. The man page includes sections for NAME, SYNOPSIS, DESCRIPTION, and AUTHOR. It details various options such as -c, -m, -l, -w, --files-from:, and --version. The 'DESCRIPTION' section notes that it prints newline, word, and byte counts for each file, or a total if multiple files are specified. The 'AUTHOR' section credits Paul Rubin and David MacKenzie.



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 21

The wc command is used to print the number of newlines, words, and/or bytes in a file.

The syntax for the wc command is: wc [OPTION(S)] [FILENAME]

Common Options include:

- -c: Print byte count
- -m: Print character count
- -l: Print newline count
- -w: Print word count

An example using the wc command to print the newline count, word count, and byte count:

- # wc -lwc File1.txt

CLI Basic Commands (cont.)



Command: grep/egrep

- Processes text line by line, and prints any lines that match a specified pattern
 - grep [OPTIONS] pattern [FILE]
 - egrep [OPTIONS] pattern [FILE]
- Common Options
 - -f <file>: Obtain patterns from <file>
 - -i: Ignore case
 - -v: Invert match (non-matching lines)
 - -w: Select whole word matches
 - -x: Exactly match whole line
 - -c: Print count of matching lines
- Examples
 - # grep "10.170." ./datafile
 - # egrep "10.170.[2-3]" ./datafile

```
GREP(1)                                General Commands Manual                               GREP(1)
NAME
    grep, egrep, fgrep, rgrep - print lines matching a pattern
SYNOPSIS
    grep [OPTIONS] PATTERN [FILE...]
    grep [OPTIONS] [-e PATTERN]... [-f FILE]...
DESCRIPTION
    grep searches the named input FILEs for lines containing a match to the given PATTERN. If no files are specified,
    or if the file "-" is given, grep searches standard input. By default, grep prints the matching lines.
    In addition, the variant programs egrep, fgrep and rgrep are the same as grep -E, grep -F, and grep -R,
    respectively. These variants are deprecated, but are provided for backward compatibility.
OPTIONS
    Generic Program Information
        --help Output a usage message and exit.
        -V, --version          Output the version number of grep and exit.
    Matcher Selection
        -E, --extended-regexp
            Interpret PATTERN as an extended regular expression (ERE, see below).
        -F, --fixed-strings
            Interpret PATTERN as a list of fixed strings (instead of regular expressions), separated by newlines, any of
            which is to be matched.
        -G, --basic-regexp
            Interpret PATTERN as a basic regular expression (BRE, see below).
        -P, --perl-regexp
            Interpret the pattern as a Perl-compatible regular expression (PCRE). This is highly experimental and grep
            -P may warn of unimplemented features.
    Matching Control
        -e PATTERN, --regexp=PATTERN
            Interpret PATTERN as the pattern. If this option is used multiple times or is combined with the -f (--file)
            option, search for all patterns given. This option can be used to protect a pattern beginning with "-".
        -f FILE, --file=FILE
            Obtain patterns from FILE, one per line. If this option is used multiple times or is combined with the -e
            option, search for all patterns given. This option can be used to protect a pattern beginning with "-".
    Manual page grep(1), line 1 (press h for help or q to quit).
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 22

The grep/egrep commands are used to processes text line by line and prints any lines which match a specified pattern.

The syntax for the grep/egrep commands are:

- grep [OPTIONS] pattern [FILE]
- egrep [OPTIONS] pattern [FILE]

Common Options include:

- -f <file>: Obtain patterns from <file>
- -i: Ignore case
- -v: Invert match (non-matching lines)
- -w: Select whole word matches
- -x: Exactly match whole line
- -c: Print count of matching lines

Examples using the grep/egrep commands are:

- # grep "10.170." ./datafile
- # egrep "10.170.[2-3]" ./datafile

CLI Basic Commands (cont.)



Command: strings

- Print strings of printable characters in files
 - strings [OPTION(S)] [FILENAME]
- Common Options
 - -a: Scan the whole file
 - -e: Select the character encoding of the strings to be found
 - -n: Change minimum length
 - -T: Specify object code format
- Example – Search /bin/cat for strings containing “Accounting”
 - # strings /bin/cat | grep Accounting

```
STRINGS(1)                                     GNU Development Tools                               STRINGS(1)
NAME
    strings - print the strings of printable characters in files.

SYNOPSIS
    strings [-eofV] [-minlen] [-maxlen] [-radixradial]
            [-e encoding] [-encodingencoding]
            [-l label] [-l filename]
            [-t targetendian]
            [-w] [-include-all-whitespace]
            [-i input-separatorsep_string]
            [-help] [-version] file...

DESCRIPTION
    For each file given, GNU strings prints the printable character sequences that are at least 4 characters long (or the number given with the options below) and are followed by an unprintable character.

    Depending upon how the strings program was configured it will default to either displaying all the printable sequences in the file or displaying only those sequences that are in initialized, initialized data sections. If the file type is unrecognizable, or if strings is reading from stdin then it will always display all of the printable sequences that it can find.

    For backwards compatibility any file that occurs after a command line option of just - will also be scanned in full, regardless of the presence of any -d option.

    strings is mainly useful for determining the contents of non-text files.

OPTIONS
    -a
    - Scan the whole file, regardless of what sections it contains or whether those sections are loaded or initialized. Normally this is the default behaviour, but strings can be configured so that the -d is the default instead.

    -d
    The -d option is position dependent and forces strings to perform full scans of any file that is mentioned after the -a on the command line, even if the -d option has been specified.

    -data
    Only print strings from initialized, loaded data sections in the file. This may reduce the amount of garbage in the output, but it also exposes the strings program to any security flaws that may be present in the BFD library used to scan and load sections. Strings can be configured so that this option is the default.

    -help
    Manual (page 1 of 1) printed | press h for help | exit QUIT!
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 23

The strings command is used to print strings of printable characters in files.

The syntax for the strings command is: strings [OPTION(S)] [FILENAME]

Common Options include:

- -a: Scan the whole file
- -e: Select the character encoding of the strings to be found
- -n: Change minimum length
- -T: Specify object code format

An example using the strings command to search /bin/cat for strings containing “Accounting”:

- # strings /bin/cat | grep Accounting

CLI Basic Commands (cont.)



Command: sort

- Sort contents of a file, line by line
 - sort [OPTION(S)] [FILE]
- Common Options
 - -b: Ignore leading blanks
 - -f: Ignore case
 - -r: Reverse sort results
 - -t[DELIM]: Use DELIM as a field delimiter
 - --sort=WORD: Sort according to WORD: general-numeric -g, human-numeric -h, month -M, numeric -n, random -R, version -V
- Example: Sort a four-column data set on the third field in reverse order
 - # sort -r +2 -3 File1 > File2
- Example: Sort a four-column data set on the fourth field, and ignore case
 - # sort -t, -f +3 -4 File1 > File2

```
sort(1)                               User Commands                               sort(1)

NAME
    sort - sort lines of text files

SYNOPSIS
    sort [OPTION]... [FILE]...
    sort [OPTION]... --files-from=FILE

DESCRIPTION
    Write sorted concatenation of all FILE(s) to standard output.

    With no FILE, or when FILE is -, read standard input.

    Mandatory arguments to long options are mandatory for short options too. Ordering options:
        -b, --ignore-leading-blanks
            ignore leading blanks
        -d, --dictionary-order
            consider only blanks and alphanumeric characters
        -f, --ignore-case
            fold lower case to upper case characters
        -g, --general-numeric-sort
            compare according to general numerical value
        -I, --ignore-nondigit
            consider only printable characters
        -M, --month-sort
            compare (unknown) < 'JAN' < ... < 'DEC'
        -h, --human-numeric-sort
            compare human readable numbers (e.g., 2K 1G)
        -n, --numeric-sort
            compare according to string numerical value
        -R, --random-sort
            shuffle, but group identical keys. See shuf(1)
        --random-source=FILE
            get random bytes from FILE
    Manual page sort(1) line 1 (press h for help or q to quit)
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 24

The sort command is used to sort contents of a file, line by line. Good Example: IP Addresses

The syntax for the sort command is: sort [OPTION(S)] [FILE]

Common Options include:

- -b: Ignore leading blanks
- -f: Ignore case
- -r: Reverse sort results
- -t[DELIM]: Use DELIM as a field delimiter
- --sort=WORD: Sort according to WORD: general-numeric -g, human-numeric -h, month -M, numeric -n, random -R, version -V

An example using the sort command to sort a 4 column data set on the 3rd field in reverse order:

- # sort -r +2 -3 File1 > File2

An example using the sort command to sort a 4 column data set on the 4th field and ignore case:

- # sort -t, -f +3 -4 File1 > File2

CLI Basic Commands (cont.)



Command: uniq

- Print or filter out repeated lines.
 - uniq [OPTIONS] [FILE]
- Common Options
 - -i: Ignore case
 - -u: Only print unique lines
 - -d: Only show multiples
 - -c: Count the number of occurrences
- Example
 - # uniq -c File1

```
UNIQ(1)                               User Commands                               UNIQ(1)

NAME
    uniq - report or omit repeated lines

SYNOPSIS
    uniq [OPTION]... [INPUT [OUTPUT]]

DESCRIPTION
    Filter adjacent matching lines from INPUT (or standard input), writing to OUTPUT (or standard output).
    With no options, matching lines are merged to the first occurrence.
    Mandatory arguments to long options are mandatory for short options too.

    -c, --count
        prefix lines by the number of occurrences
    -d, --repeated
        only print duplicate lines, one for each group
    -D
        print all duplicate lines
    -all-repeated[:METHOD]
        like -D, but allow separating groups with an empty line; METHOD=(none(default),prepend,separate)
    -f, --skip-fields=N
        avoid comparing the first N fields
    -group[:METHOD]
        show all items, separating groups with an empty line; METHOD=(separate(default),prepend,append,both)
    -i, --ignore-case
        ignore differences in case when comparing
    -u, --skip-chars=N
        avoid comparing the first N characters
    -U, --unique
        only print unique lines
    -Z, --zero-terminated
        line delimiter is NUL, not newline
    -W, --check-chars=N
        Manual page uniq(1) line 3 (press h for help or q to quit)
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 25

The uniq command is used to print or filter out repeated lines.

The reason for the unique command is the -c option. Note you MUST run sort before unique, you do not need any flags with sort.

The syntax for the uniq command is: uniq [OPTIONS] [FILE]

Common Options include:

- -i: Ignore case
- -u: Only print unique lines
- -d: Only show multiples
- -c: Count the number of occurrences

An example using the uniq command, using the option to count the number of occurrences:

- # uniq -c File1

CLI Basic Commands (cont.)



Command: xargs

- Build and execute command lines from standard input
 - xargs [OPTIONS] command [initial arguments]
- Common Options
 - -a: Read from file
 - -d[DELIM]: Set delimiter character
 - -e: Set End of File (EOF) string
 - -n: Max arguments, execute command for every n args
- Example – Run a Network Mapper (Nmap) scan on a list of Internet Protocol (IP) addresses
 - # cat FinanceIPs | xargs -n1 nmap -sV

```
XARGS(1)                                General Commands Manual                               XARGS(1)

NAME
    xargs - build and execute command lines from standard input

SYNOPSIS
    xargs [-0ptrx] [-E eof-str] [-e eof-str] [-o null] [-d delimiter] [-delim delimiter] [-I
    replace-str] [-A replace-str] [-replace-str] [-L lines] [-A max-lines] [-max-lines max-lines]
    [-r replace-args] [-R replace-args] [-r max-replace-args] [-R max-replace-args] [-c
    process-slot-var-name] [-interactive] [-verbose] [-exit] [-no-run-if-empty] [-arg-file=arg-file]
    [-show-limits] [-version] [-help] [command [initial-arguments]]

DESCRIPTION
    This manual page documents the GNU version of xargs. xargs reads items from the standard input, delimited by blanks (which can be protected with double or single quotes or a backslash) or newlines, and executes the command (default is /bin/echo) one or more times with any initial-arguments followed by items read from standard input. Blank lines on the standard input are ignored.

    The command line for command is built up until it reaches a system-defined limit (unless the -n and -L options are used). The specified command will be invoked as many times as necessary to use up the list of input items. In general, there will be many fewer invocations of command than there were items in the input. This will normally have significant performance benefits. Some commands can usefully be executed in parallel too; see the -P option.

    Because Unix filenames can contain blanks and newlines, this default behaviour is often problematic: filenames containing spaces or newlines will be interpreted as multiple arguments. To avoid this, use the -0 option, which prevents such problems. When using this option you will need to ensure that the program which produces the input for xargs also uses a null character as a separator. If that program is GNU find for example, the -print0 option does the right thing.

    If any invocation of the command exits with a status of 255, xargs will stop immediately without reading any further input. An error message is issued on stderr when this happens.

OPTIONS
    -0, --null
        Input items are terminated by a null character instead of by whitespace, and the quotes and backslash are not special (every character is taken literally). Disables the end-of-file string, which is treated like any other argument. Useful when input items might contain white space, quote marks, or backslashes. The GNU find -print0 option produces input suitable for this mode.

    -a file, --arg-file=file
        Read command line items from file instead of standard input. If you use this option, stdin remains unchanged when commands are run. Otherwise, stdin is redirected from /dev/null.

    -d delimiter, -d delm
        Input items are terminated by the specified character. The specified delimiter may be a single character, a
        Manual page xargs(1) (press h for help or q to quit)
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 26

The xargs command is used to build and execute command lines from standard input.

The syntax for the xargs command is: xargs [OPTIONS] command [initial arguments]

Common Options include:

- -a: Read from file
- -d[DELIM]: Set delimiter character
- -e: Set EOF (end of file) string
- -n: Max arguments, execute command for every n args

An example using the cat command combined with the xargs command to run an nmap scan on a list of ips:

- # cat FinanceIPs | xargs -n1 nmap -sV

CLI Basic Commands (cont.)



Command: cut

- Cut out selected portions of each line using a delimiter
 - `cut [OPTION(S)] [FILE]`
- Common Options
 - `-d`: Specify delimiter (tab is default)
 - `-c`: Specify character position
 - `-f`: Specify field by number
 - `-s`: Suppress lines when no delimiter
- Example – Extract fields 1 and 7 from `/etc/passwd`
 - `# cut -d : -f 1,7 /etc/passwd`

```
CUT(1)                               User Commands                               CUT(1)
NAME
    cut - remove sections from each line of files
SYNOPSIS
    cut OPTION... [FILE]...
DESCRIPTION
    Print selected parts of lines from each FILE to standard output.
    With no FILE, or when FILE is -, read standard input.
    Mandatory arguments to long options are mandatory for short options too.
    -b, --bytes=LIST
        select only these bytes
    -c, --characters=LIST
        select only these characters
    -d, --delimiter=DELIM
        use DELIM instead of TAB for field delimiter
    -f, --fields=LIST
        select only these fields; also print any line that contains no delimiter character, unless the -s option is
        specified
    -n
        (ignored)
    --complement
        complement the set of selected bytes, characters or fields
    -s, --only-delimited
        do not print lines not containing delimiters
    --output-delimiter=STRING
        use STRING as the output delimiter the default is to use the input delimiter
    -z, --zero-terminated
        line delimiter is NUL, not newline
    --help
        display this help and exit
    --version
        Manual page cut(1) line 3 (press h for help or q to quit)

Manual page cut(1) line 3 (press h for help or q to quit)
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 27

The `cut` command is used to cut out selected portions of each line using a delimiter.

The syntax for the `cut` command is: `cut [OPTION(S)] [FILE]`

Common Options include:

- `-d`: Specify delimiter (tab is default)
- `-c`: Specify character position
- `-f`: Specify field by number
- `-s`: Suppress lines when no delimiter

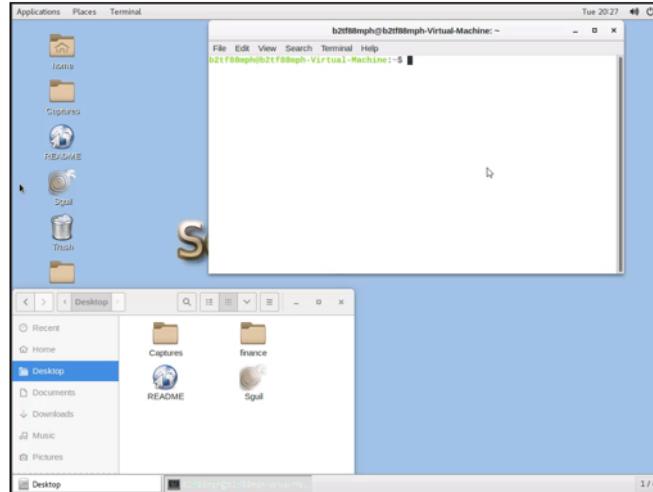
An example using the `cut` command to extract fields 1 and 7 from `/etc/passwd`

- `# cut -d : -f 1,7 /etc/passwd`

Working with Directories

Make and Remove Directories

- The `pwd` command prints the working directory
 - `# pwd`
- The `cd` command, without a pathname, changes to the home directory
 - `# cd`
- The `mkdir` command makes a new directory
 - `# mkdir Accounting`
- The `rmdir` command removes a directory
 - `# rmdir Accounting`
- The `cd` command changes to the directory
 - `# cd /finance`



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 28

Click the image to play the video

There are a few commands commonly used when working with directories.

The `mkdir` command makes a new directory, for example

```
# mkdir Accounting
```

The `rmdir` command removes a directory, for example

```
# rmdir Accounting
```

The `cd` command changes to the directory, for example

```
# cd /finance
```

The `cd` command without a pathname changes to your home directory, for example

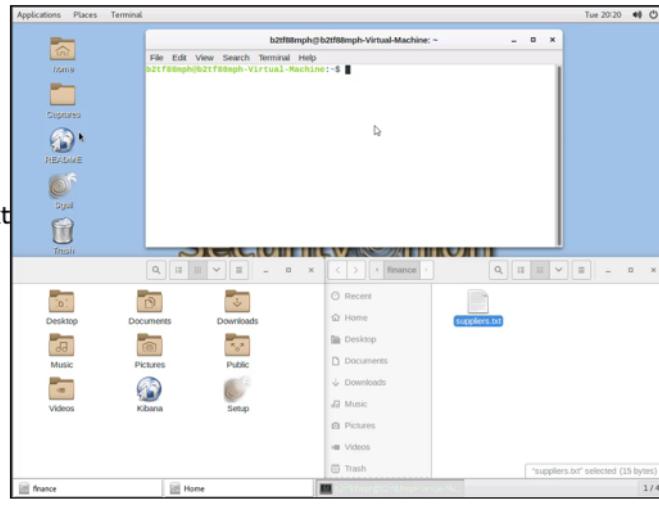
```
# cd
```

The `pwd` command prints the working directory, for example

```
# pwd
```

Moving Files

- Users have the permissions to create or write files to their home directory
- The ~ character saves files to the home directory
 - # File1 > ~/File1.txt
- The mv command moves files between directories
 - # mv finance/suppliers.txt ~suppliers.txt
- The rm command removes files or directories
 - rm [-i] [-r] File1 File 2 ...
 - i Inquire before removing
 - r Recursively remove a directory and all its contents and subdirectories
 - # rm file1.txt file2.txt
- The cp command copies files between directories
 - # cp ~/indicators.txt ~/solutions/indicators/



Click the image to play the video

There are a few commands commonly used when moving files.

Users have the permissions to create or write files to their home directory.

The ~ character saves files to your home directory, for example
File1 > ~/File1.txt

The mv command moves files between directories, for example
mv /finance/suppliers.txt ~/suppliers.txt

The rm command removes files or directories, for example
rm [-i] [-r] File1 File 2 ...
-i Inquire before removing
-r Recursively remove a directory and all its contents and subdirectories

An example using the rm command is:
rm file1.txt file2.txt

The cp command copies files to between directories, for example
cp ~/indicators.txt ~/solutions/indicators/



LINUX is Case Sensitive, Sensitive in General!

This is a very important aspect to LINUX...

When the command does not appear to be working, check to make sure that the commands are correct, spacing is correct, and the correct characters were used.

90 percent of the time, it will resolve the issue!



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 30



CLI Basic Commands

There are many more commands available in Linux. Explore the other commands available; listed here are a few additional commands.

Command	Description
passwd	Change password
paste	Display contents of files side by side
merge	Three-way file merge
join	Join lines of two files with a common field
lpr	Send to printer
touch	Change file timestamps
date	System's time and date information



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 31

There are many more commands available in Linux. Explore the other commands available; listed below are a few additional commands:

- | | |
|--------|--|
| passwd | to change the password |
| paste | to display contents of files side-by-side |
| merge | to do a three way file merge |
| join | to join lines of two files with a common field |
| lpr | to send to printer |
| touch | to change file timestamps |
| date | to display system's time and date information |

LAB002: Command Line Essentials



© Capgemini 2019. All rights reserved | 32

Content Source:



Questions?



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 33

Working with Directories and Files



Foundational Analyst Security Training

The next section will cover working with directories and files.

Agenda



WORKING WITH DIRECTORIES | I/O STREAMS | PIPES AND TEES

The topics covered in this module include:

- vi/vim
- Regular Expressions
- gawk
- sed



Topic Learning Objectives

Upon completion of this topic, the student should be able to do the following:

- Understand how data is output to streams, Tees, and Pipes in LINUX and how this can be used in file analysis.



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 36



I/O Streams

Linux receives and sends output in character streams.

- The three standard I/O streams are as follows:
 1. stdout: Standard command output, which is usually to the screen
 2. stdin: Standard command input, which is usually the keyboard
 3. stderr: Standard error output codes from commands



In a Terminal session, stdout and stderr are both output by default; change this with redirection.



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 37

Linux receives and sends output in character streams.

There are three standard I/O streams:

- stdout: Standard command output, which is usually to the screen
- stdin: Standard command input, which is usually the keyboard
- stderr: Standard error output codes from commands

NOTE: In a Terminal session, stdout and stderr are both output by default; you can change this with redirection.

Output Redirection

Output can be easily redirected using “>” (overwrite/create) and “>>” (append/create).

- Example – Redirect “ls -al” output to a new file named “audit.txt”
`# ls -al > audit.txt`
- Example – Append output from multiple cat commands to “AllFiles.txt”
`# cat File1.txt File2.txt >> AllFiles.txt`

Both “>” and “>>” implicitly work only with stdout (do one of the following also to handle stderr):

- Example – Redirect stdout and stderr from “ls -al” to a new file named “AllFiles.txt”

`# ls -al 1>AllFiles.txt 2>FileErrors.txt`



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 38

Output can be easily redirected using “>” (overwrite/create) and “>>” (append/create).

An example to redirect “ls -al” output to a new file named “audit.txt” is:

`# ls -al > audit.txt`

An example to append output from multiple cat commands to “AllFiles.txt” is:

`# cat File1.txt File2.txt >> AllFiles.txt`

Both “>” and “>>” implicitly work only with stdout (you must do one of the following to also handle stderr)

An example to redirect stdout and stderr from “ls -al” to a new file named “AllFiles.txt” is:

`# ls -al 1>AllFiles.txt 2>FileErrors.txt`

An example to suppress all output from a command is:

`# gedit AllFiles.txt 1>/dev/null 2>/dev/null`



Basic Pipes

A pipe is a form of redirection to transfer standard output to another destination.

- Combine multiple commands, and redirect the stdout of one command to the stdin of the second.

Uses “|” to delineate commands

command1 | command2 | command3 | ...

- Example – Display directory contents in long format, and paginate the results.

```
# ls -al | more
```

- Example – Use ls and find commands to list and print all lines matching a pattern.

```
# ls -l | find ./ -type f -name "*.txt" -exec grep "program" {} \;
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 39

A pipe is a form of redirection to transfer standard output to another destination (combine multiple commands and redirect the stdout of one command to the stdin of the second); it uses “|” to delineate commands, such as:
command1 | command2 | command3 | ...

An example to display directory contents in long format and paginate the results is:

```
# ls -al | more
```

An example to use ls and find commands to list and print all lines matching a pattern is:

```
# ls -l | find ./ -type f -name "*.txt" -exec grep "program" {} \;
```



Tees

A tee in Linux allows the following actions:

- Save the output from a command to a file.
- Output the results (to the screen or another pipe).

Example

- The filetype outputs are written to file1.txt, which will be displayed in the user's home directory and the terminal.

```
# ls -al | file -f - | tee ~/file1.txt
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 40

A tee in Linux allows you to save the output from a command to a file, and output the results (to the screen or another pipe).

An example where the filetype outputs are written to file1.txt, which will be displayed in the user's home directory and the terminal:

```
# ls -al | file -f - | tee ~/file1.txt
```



Questions?



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 41

Working with Text Editors

Analyze and Modify Text Files



Capgemini

Foundational Analyst Security Training

In this module, we will cover Working with Text Editors.

Agenda



**TEXT EDITORS | GAWK – STREAM EDITOR (SED) |
REGULAR EXPRESSIONS (REGEXES)**

The topics covered in this module include:

- Basic Scripting Concepts
- Bash Shell
- Other Scripting Languages



Topic Learning Objectives

Upon completion of this topic, the student should be able to do the following:

- Understand what a text editor is and how to use it to create and edit a file in LINUX.
- Understand how to use Global Regular Expression Print (grep), Regex, gawk, xargs, and sed to manipulate data in LINUX.



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 44

Upon successful completion of this module, student will be able to:

- Edit text files using vi and vim
- Use regular expressions (regexes) for string matching, parsing, and replacement
- Use gawk for field extraction, sorting, unique, and counts
- Use sed to search and replace



Text Editors

- It is often necessary to use built-in Linux command-line text editors.
 - "vi" provides basic text editing capabilities.
 - "vim" provides additional capabilities.
- What is vi/vim?
 - The default *nix editor is called vi (visual editor).
 - The vi editor has two modes:
 1. Command mode causes action to be taken on the file.
 2. Insert mode inserts text into the file.
 - Pressing the <Esc> key turns off the Insert mode.
 - To enter vi, type vi filename.



If the filename exists, it opens the file; if the filename does not exist, it creates a new file.



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 45

You will often need to use built-in Linux command line text editors. "vi" provides basic text editing capabilities, while "vim" provides additional capabilities.

The default editor is called vi, which stands for visual editor. The vi editor has the Command mode, causes action to be taken on the file; and it has the Insert mode, which inserts text into the file.

Pressing the <Esc> key turns off the Insert mode.

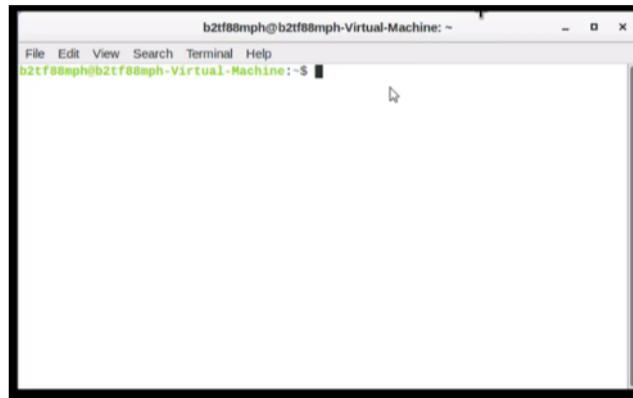
To enter vi, type vi followed by the filename.

NOTE: If the filename exists, it opens the file; if the filename does not exist, it creates a new file.

Text Editors (cont.)

Opening, Closing, and Saving

- To enter vi, type vi filename.
- NOTE: If the filename exists, it opens the file; if the filename does not exist, it creates a new file.
- To quit, type a ":" which moves the cursor to the bottom of the screen.
- Enter the desired command, followed by "Enter":
 - :w quit vi, write to disk
 - :w filename quit vi, writing to filename
 - :wq! quit vi, write to disk, override read only
 - q! quit vi, do not save changes



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 46

As previously mentioned, to enter vi, type vi followed by the filename.

To quit, type a ":" which moves the cursor to the bottom of the screen. Then, enter the desired command followed by "Enter":

- :w quit vi, write to disk
- :w filename quit vi, writing to filename
- :wq! quit vi, write to disk, override read only
- :q! quit vi, do not save changes

Text Editors (cont.)



- vi/vim example to change and save a file.
- # vi AddUserScript.pl
 - vi will start in command mode.
 - Use the arrow keys to move to the line(s) to change.
 - Type the letter for the edit mode to use:
 - a (append after cursor)
 - i (insert before cursor)
 - o (open line below)
 - O (open line above)
 - :r filename (insert filename after current line)
 - Change the text using the keyboard (see the Command Reference).
 - Also useful in command mode are the following commands:
 - x (delete the character at the cursor's position)
 - dd (cut the current line to buffer)
 - dw (cut the current word to buffer)
 - d (delete to end of sentence)
 - p (paste from buffer)
 - Save the file, and exit ":wq!" (quit vi, write to disk, override read only).



The following example sequence would change and save a file:

- Enter vi followed by the filename, such as AddUserScript.pl.
- vi will start in command mode
- Then you can use the arrow keys to move to the line(s) you want to change.
- Type the letter for the edit mode you want to use, such as:
 - a (append after cursor)
 - i (insert before cursor)
 - o (open line below)
 - O (open line above)
 - :r filename (insert filename after current line)
- Change the text as needed, using the keyboard (see the Command Reference on the next slide – or many that exist online).
- Also useful in command mode are the commands:
 - x (delete the character at the cursor's position)
 - dd (cut the current line to buffer)
 - dw (cut the current word to buffer)
 - d (delete to end of sentence)
 - p (paste from buffer)
- Save the file and exit ":wq!" (quit vi, write to disk, override read only).

vi/vim Command Reference



Invoking vi:	<code>vi filename</code>
Format of vi commands:	<code>[count]/[command]</code>
Command mode versus input mode	
Vi starts in command mode. The positioning commands operate only while vi is in command mode. You switch vi to input mode by entering any one of several vi input commands. When you type a command, the command is executed. After you type is taken to be text and is added to the file. You cannot execute any commands until you exit input mode. To exit input mode, press the escape key.	
Input commands (end with Esc)	
a	Append after cursor
i	Insert before cursor
o	Open line below
O	Open line above
x/file	Insert file after current line
Any of these commands leave vi in input mode until you press Esc. Pressing the RETURN key will not take you out of input mode.	
Change commands (Input mode)	
cc	Change word (Esc)
cc	Change line (Esc) - blanks line
c\$	Change to end of line
rc	Replace character with c
R	Replace (Esc) - typover
substitute (Esc) - 1 char with string	
S	Substitute (Esc) - 1 char with text
.	Repeat last change
Changes during insert mode	
<ctrl>h	Back one character
<ctrl>w	Back one word
<ctrl>u	Back to beginning of insert
File management commands	
:w name	Write edit buffer to file name
:wq	Write to file and quit
:q!	Quit without saving changes
:Z	Same as :wq
:sh	Execute shell commands (<ctrl>d)
Window motions	
<ctrl>d	Scroll down (half a screen)
<ctrl>u	Scroll up (half a screen)
<ctrl>f	Page forward
<ctrl>b	Page backward
String	Search forward
String	Search backward
<ctrl>g	Display current line number and file information
n	Repeat search
N	Repeat search reverse
G	Go to last line
gG	Go to line n
:	Go to line n
g<CR>	Reposition window: cursor at top
g<CR>	Reposition window: cursor in middle
g<CR>	Reposition window: cursor at bottom
Deletion commands	
dd	Delete a line to general buffer
dw	Delete word to general buffer
ds	Delete a word
d\$	Delete to end of sentence
db	Delete previous word
de	Delete end of line
x	Delete character
p	Put general buffer after cursor
P	Put general buffer before cursor
U	Undo last change
U	Undo all changes on line
Undo commands	
U	Undo last change
U	Undo all changes on line
Rearrangement commands	
yy or Y	Yank (copy) line to general buffer
9yy	Yank 9 lines to buffer a
yy	Yank line to general buffer
^yM	Delays 9 lines to buffer a
^yM	Delays 9 lines, Append to buffer a
^yap	Put next line from buffer a after cursor
^yP	Put next line from buffer a before cursor
P	Put general buffer before cursor
J	Join lines
Parameters	
set list	Show invisible characters
set nolist	Don't show invisible characters
set number	Show line numbers
set nonumber	Don't show line numbers
set autoindent	Turn off automatic carriage return
set noautoindent	Turn off automatic carriage return
set nowrap	Show matching sets of parentheses as they are typed
set nowrapmargin	Turn off nowrap mode
set noshowmode	Display mode on last line of screen
set showmode	Turn off showmode
set nowrapmode	Show values of all possible parameters
set all	
Cursor motions	
H	Upper left corner (home)
M	Middle line
L	Lower left corner
b	Back a character
j	Down a line
k	Up a line
^	Beginning of line
\$	End of line
l	Forward a character
w	One word forward
b	One word backward
e	Find e
:	Repeat find (find next e)
Move text from file old to file new	
vi old	vi old
"w!yy	yank 10 lines to buffer a
a	write work buffer
z new	edit new file
"ap	put text from a after cursor
30,60w new	Write lines 30 to 60 in file new
Regular expressions (search strings)	
.	Matches beginning of line
\$	Matches end of line
*	Matches any single character
#	Matches previous character
.*	Matches any character
Search and replace commands	
Syntax:	
:	[address]e[old_text]/new_text/[options]
Address components:	
n	Current line
^n	Line number n
^	Current line plus m lines
^	Last line
~string~	A line that contains "string"
%	Entire file
[addr1],[addr2]	Specifies a range
Examples:	
The following example replaces only the first occurrence of banana with kiwi in each of 11 lines starting with the current line (.) and continuing for the 10 that follow (.+10).	:.+10s/banana/kiwi/g
The following example replaces every occurrence (including the last) of pear with apple at the end of the command:	:\$s/pear/apple/g
The following example removes the last character from every line in the file. Use it if every line in the file ends with a character that's a file terminator. Execute it when the cursor is on the first line of the file:	:%s/\.\$//

This cheat sheet
will be available
in the laboratory
environment,
when needed.



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 48

This slide depicts common commands. Likewise, if you search online, you will see many command reference that you can view and/or download.



Xargs

- The “xargs” command is used to execute an additional Linux command on every line from standard input.
 - This allows performing operations on a list via command line execution.
 - Almost always receives information from a pipe.
 - The Linux command to run on each line is placed after “xargs.”

Example...

“ls -l” output creates a list of files in the current directory.

- # ls -l
- document1.txt
- important_file.doc
- config.sys

From this output, use piping and “xargs” to perform the Linux command “rm” on each file passed to STDIN from the pipe to delete the files.

- # ls -l | xargs rm



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 49



Text Files – Regular Expressions (Regexes)

- Used for string matching, parsing, and replacement
- Especially helpful to match a known pattern (such as dates, IP addresses, Uniform Resource Locators [URLs], etc.)
- Regular grep supports basic Regex (use egrep or grep -e for extended support).



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 50

Regular expressions (regexes) are used for string matching, parsing, and replacement, which is very common when analyzing files, such as log files.

They are especially helpful to match a known pattern (such as dates, IP addresses, URLs, etc.).

Regular grep supports basic regex; use egrep or grep -e for extended support.



Text Files – Regex Quantifiers

Specify characters to match.

- Search for each instance of the word "audit" (-i for not case sensitive).
`# grep -i "audit" file1`
- Search for every line that does not contain the word "audit." "-v" will invert the match (find those that do not match).
`# grep -v "audit" file1`

- Search for the number of matches ("-n" for the number, or count, of matches).
`# grep -n "audit" file1`
- "^" to match at the beginning of the line
`# grep "^audit" file1`
- "\$" to match at the end of the line
`# grep "audit$" file1`



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 51

This slide shows some examples using the grep command to match characters in a file.

To search for each instance of the word "audit" (-i for not case sensitive).

```
# grep -i "audit" file1
```

To search for every line that does not contain the word "audit". "-v" will invert the match (find those that do not match)

```
# grep -v "audit" file1
```

To search for the number of matches. "-n" for the number, or count, of matches.

```
# grep -n "audit" file1
```

Use "^" to match at the beginning of the line.

```
# grep "^audit" file1
```

Use "\$" to match at the end of the line.

```
# grep "audit$" file1
```



Text Files – Regex Quantifiers (cont.)

Specify how many instances of a character, group, or character class to match.

- Only apply to the character, group, or character class that immediately precedes it.
- The quantifiers only apply to the character “c” in the following examples:
 - “*”: matches character before it zero or more times; match “abcxyz,” “abxyz,” “abccxyz”
egrep “abc*xyz” file1
 - “?": matches zero or one time; matches "abcxyz," "abxyz" but does not match "abccxyz"
egrep "abc?xyz" file1
 - “+”: matches one or more; matches "abcxyz," "abccxyz" but does not match "abxyz"
egrep "abc+xyz" file1



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 52

As previously mentioned, egrep provides extended capabilities.

You can specify how many instances of a character, group, or character class to match. It only apply to the character, group, or character class that immediately precedes it.

The quantifiers only apply to the character “c” in the following examples.

The “*” is commonly used to match character before it zero or more times. # egrep “abc*xyz” file1 will match “abcxyz”, “abxyz”, “abccxyzg”

The “?” is commonly used to match zero or one time. # egrep “abc?xyz” file1 will match “abcxyz”, “abxyz”, but does not match “abccxyz”

The “+” is commonly used to match one or more. # egrep “abc+xyz” file1 will match “abcxyz”, “abccxyz”, but does not match “abxyz”



Text Files – Regex Character Classes

Use square brackets ("[]") to group characters.

- Match any of characters of "1," "2," and "3"

```
# grep "[123]" file1
```

- Match "hood" or "food"

```
# grep "[hf]ood" file1
```

- Match any characters between "a" and "z"

```
# grep "[a-z]" file1
```

- Match lines that begin with a capital letter

```
# grep "^[A-Z]" file1
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 53

You can also use square brackets ("[]") to group characters, as depicted in these examples.

To match any of characters of "1", "2", and "3"

```
# grep "[123]" file1
```

To match "good" or "hood"

```
# grep "[hf]ood" file1
```

To match any characters between "a" and "z"

```
# grep "[a-z]" file1
```

To match lines that begin with a capital letter

```
# grep "^[A-Z]" file1
```



Text Files – Regex Character Classes (cont.)

Use ^ inside the brackets to invert the results.

–Match anything other than characters “1,” “2,” or “3”

```
# grep "[^123]" file1
```

Use curly braces (“{ }”) to specify the number of occurrences of the preceding character.

–Match at least three occurrences of a digit character and no more than six occurrences.

```
# egrep "[0-9]{3,6}" file1
```



This applies to extended grep and Perl-Compatible Regular Expression (PCRE) only.



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 54

Use ^ inside the brackets to invert the results; for example, to match anything other than characters “1”, “2”, or “3”:

```
# grep "[^123]" file1
```

Use curly braces (“{ }”) to specify the number of occurrences of the preceding character. Note: This applies to extended grep and PCRE only. To match at least 3 occurrences of a digit character, and no more than 6 occurrences:

```
# egrep "[0-9]{3,6}" file1
```



Text Files – Regex Logic (Or)

Use the pipe character ("|") inside the quotes for the "OR" condition.

- Match "good" or "well" in the file1.
egrep "good|well" file1
- Match "good" or "gold" in file1.
grep "go(od|ld)" file1

Use multiple grep statements, separated by pipes, to search for two or more patterns (AND condition). Match "pattern1" in file1; and then, from that result, match "pattern2":

- # grep "pattern1" file1 | grep "pattern2"



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 55

Use the pipe character ("|") inside the quotes for the "OR" condition. For example, to match "good" or "well" in file1:

```
# egrep "good|well" file1
```

To match "good" or "gold" in file1:

```
# grep "go(od|ld)" file1
```

Use multiple grep statements separated by pipes to search for two or more patterns, applying the AND condition. Match "pattern1" in file1 then from that result match "pattern2":

```
# grep "pattern1" file1 | grep "pattern2"
```



Text Files – Regex Logic (Not)

Escaping Meta-Characters

- Meta-Characters have special meaning; thus, to search for those, such as a literal period or a literal opening bracket, "escape" these characters to ignore their special meaning.
- "\" before the character escapes the special meaning of the character.
- Search for lines that begin with a capital letter and end with a period:
- # grep "^[A-Z].*\\" file1



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 56

There are meta characters that have special meaning; thus, to search for those, such as a literal period or a literal opening bracket, you need to "escape" these characters to ignore their special meaning. Using the "\" before the character escapes the special meaning of the character. For example, to search for lines that begins with a capital letter and ends with a period:

```
# grep "^[A-Z].*\\" File1
```



Text Files – More Regex Examples

More Regex Examples:

- Internet Protocol Version 4 (IPv4) address
- # egrep "^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}" f.log
- Hexadecimal Values (e.g., "0x0ab3e")
- # egrep "0x[0-9a-fA-F]+" f.log



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 57

A couple other egrep examples to search for IP addresses or hexadecimal values are as follows:

An example to search IPv4 addresses is:

```
# egrep "^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}" f.log
```

An example to search Hexadecimal Values, such as "0x0ab3e", is:

```
# egrep "0x[0-9a-fA-F]+" f.log
```



Text Files – gawk

gawk contains pattern matching and field processing capabilities (among many other capabilities).

Example...

- Usage:
- gawk <options> <PROGRAM> Input-File1 Input-File2 ...
- Common Options:
- -f: Specify program commands in an input file.
- -F: Set custom field separator.



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 58

gawk contains pattern matching and field processing capabilities (among many other capabilities). The syntax for using gawk is:

gawk <options> <PROGRAM> Input-File1 Input-File2 ...

Common options include:

- -f: Specify program commands in an input file
- -F: Set custom field separator

An example using gawk to find all lines in file1 that start with a string between 2011 and 2019 is:

- # gawk '/^201[1-9]/' file1



Text Files – gawk (cont.)

Field Extraction and Fields

- Print the fifth field in every line.

```
# gawk '{print $5}' /var/log/file1
```

- Change the field delimiter to a colon + space, and print the fifth field.

```
# gawk -F ': ' '{print $5}'  
/var/log/file1
```

- Print the first and second field separated by a forward slash.

```
# gawk '{print $1 "/" $2}' /var/log/file1
```

- Print the number of fields in each line.

```
# gawk '{print NF}' /var/log/file1
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 59

Gawk examples for Field Extraction and working with Fields is depicted on this slide.

To print the fifth field in every line:

```
# gawk '{print $5}' /var/log/file1
```

To change the field delimiter to a colon + space, and print the fifth field:

```
# gawk -F ': ' '{print $5}' /var/log/file1
```

To print the first and second field separated by a forward slash:

```
# gawk '{print $1 "/" $2}' /var/log/file1
```

To print the number of fields in each line:

```
# gawk '{print NF}' /var/log/file1
```



Text Files – gawk (cont.)

Field Extraction and Fields

- Print every line that has at least five fields.
`# gawk 'NF>=5{print}' /var/log/file1`
- Print out the last field (\$NF) in each matching line.
`# gawk '{print $NF}' /var/log/file1`
- Print the second-to-last field in every line.
`# gawk '{print $(NF-1)}' /var/log/file1`



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 60

More examples of Field Extraction and working with Fields include:

To print every line that has at least 5 fields:

```
# gawk 'NF>=5{print}' /var/log/file1
```

To print out the last field (\$NF) in each matching line:

```
# gawk '{print $NF}' /var/log/file1
```

To print the second-to-last field in every line:

```
# gawk '{print $(NF-1)}' /var/log/file1
```

Text Files – sed



Search and Replace

- “sed” is short for stream editor.
- Provides search and replace capabilities
- Accepts regular expressions and is very useful for converting text from one format to another
- There are many pre-canned sed commands available online.
- By default, sed only replaces the first match.
- To execute for every match, add character “g” to the end of the search expression

For example...

- change 192.168 addresses to 172.16 addresses.
sed 's/^192\.168/172\.16/g'
IPAddresses.txt
- Remove leading and trailing whitespace for addresses that contain 192.
grep "192" IPAddresses.txt | sed 's|^[\t]*||;s|[\t]*\$||'



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 61

Sed is a commonly used command for searching and replacing.

“sed” is short for “stream editor, and it provides search and replace capabilities.

Sed accepts regular expressions and is very useful for converting text from one format to another. Additionally there are many pre-canned sed commands available online.

By default, sed only replaces the first match. To execute for every match, add the character “g” to the end of the search expression.

For example, to change 192.168 addresses to 172.16 addresses:

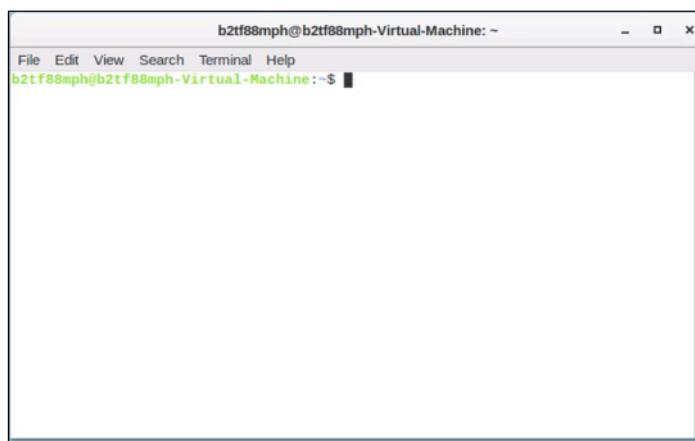
```
# sed 's/^192\.168/172\.16/g' IPAddresses.txt
```

To remove leading and trailing whitespace for addresses that contain 192:

```
# grep "192" IPAddresses.txt | sed 's|^[\t]*||;s|[\t]*$||'
```

Text Files – grep

- grep is a command-line utility for searching plaintext data sets for lines that match a regular expression.
- Its name comes from the ed command g/re/p, which has the same effect: doing a global search with the regular expression and printing all matching lines.
- grep was originally developed for the Unix operating system but later available for all Unix-like systems.



A screenshot of a terminal window titled "b2tf88mph@b2tf88mph-Virtual-Machine: ~". The window has standard OS X-style window controls at the top right. The menu bar includes "File", "Edit", "View", "Search", "Terminal", and "Help". The main pane shows a single line of text: "b2tf88mph@b2tf88mph-Virtual-Machine:~\$". The background of the slide features a faint watermark of a blue speech bubble icon in the top right corner.

LAB003: LINUX File Manipulation



© Capgemini 2019. All rights reserved | 63

Content Source:



Questions?



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 64

Scripting Basics



Capgemini

Foundational Analyst Security Training

In this topic, we will cover basic scripting concepts.

Agenda



SCRIPTING CONCEPTS | SCRIPTING LANGUAGES | PYTHON

The topics covered in this module include:

- Basic Scripting Concepts
- Bash Shell
- Perl
- Other Scripting Languages



Topic Learning Objectives

Upon completion of this topic, the student should be able to do the following:

- Understand the difference between compiled and non-compiled scripting languages.
- Identify some common programming and scripting languages.
- Understand how Python is used to code scripts to perform repetitive tasks.
- Understand how Python interacts and manipulates information using strings.



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 67

Scripting vs. Programming



Scripting Languages

- Programming language that support scripts or programs written for a special run time environment.
- Often used for the automation of repetitive tasks such as log analysis or searching.
- Operate at a very high level, and do not have to be compiled like a programming language.



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 68

A **scripting or script language** is a [programming language](#) that supports **scripts** — programs written for a special [run-time environment](#) that automates the [execution](#) of tasks^[1] that could alternatively be executed one-by-one by a human operator.

Scripting languages are often interpreted (rather than [compiled](#)). Primitives are usually the elementary tasks or [API](#) calls, and the language allows them to be combined into more complex programs.

Environments that can be automated through scripting include [software applications](#), [web pages](#) within a [web browser](#), usage of the [shells](#) of [operating systems](#) (OS), [embedded systems](#), as well as numerous games.

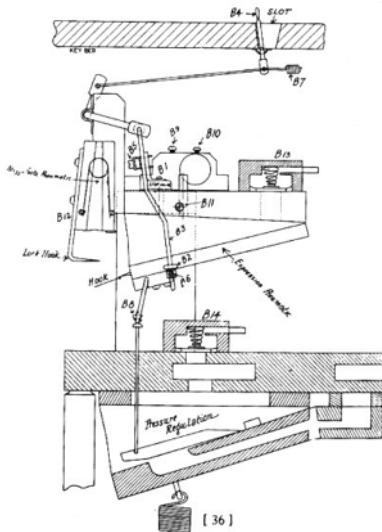
A scripting language can be viewed as a [domain-specific language](#) for a particular environment; in the case of scripting an application, it is also known as an [extension language](#). Scripting languages are also sometimes referred to as [very high-level programming languages](#), as they operate at a high level of abstraction, or as [control languages](#), particularly for job control languages on mainframes.

Scripting vs. Programming (cont.)



Programming Language

- A programming language is made up of a set of instructions that produce an output.
 - These types of languages are used in computer programming to create instructions for computers.
 - Modern-day programming languages are known as compiled languages that have to be compiled before they can be run.



© Cengage 2019. All rights reserved | 69

A **programming language** is a formal language, which comprises a set of instructions that produce various kinds of output. Programming languages are used in computer programming to implement algorithms.

Most programming languages consist of **instructions** for computers.

There are programmable machines that use a set of specific instructions, rather than general programming languages. Early ones preceded the invention of the digital computer, the first probably being the automatic flute player described in the 9th century by the brothers Musa in Baghdad, during the Islamic Golden Age.^[1]

Since the early 1800s, programs have been used to direct the behavior of machines such as [Jacquard looms](#), [music boxes](#) and [player pianos](#).^[2] The programs for these machines (such as a player piano's scrolls) did not produce different behavior in response to different inputs or conditions.

From 1832 until 1932, the firm produced mechanical musical instruments of the highest quality. The firm's founder, Michael Welte (1807-1880), and his company were prominent in the technical development and construction of orchestrions from 1850, until the early 20th century.

Scripting vs. Programming (cont.)



Programming Language

One of the earliest forms of programming was used by the Welte-Mignon automatic piano-playing systems, which used a prerecorded mechanical representation of the music to reproduce a song.



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 70

A **programming language** is a [formal language](#), which comprises a [set of instructions](#) that produce various kinds of [output](#). Programming languages are used in [computer programming](#) to implement [algorithms](#).

Most programming languages consist of [instructions](#) for [computers](#).

There are programmable machines that use a set of [specific instructions](#), rather than [general programming languages](#). Early ones preceded the [invention of the digital computer](#), the first probably being the automatic flute player described in the 9th century by the [brothers Musa](#) in [Baghdad](#), during the [Islamic Golden Age](#).^[1]

Since the early 1800s, programs have been used to direct the behavior of machines such as [Jacquard looms](#), [music boxes](#) and [player pianos](#).^[2] The programs for these machines (such as a player piano's scrolls) did not produce different behavior in response to different inputs or conditions.

From 1832 until 1932, the firm produced [mechanical musical instruments](#) of the highest quality. The firm's founder, Michael Welte (1807-1880), and his company were prominent in the technical development and construction of [orchestrions](#) from 1850, until the early 20th century.



Scripting: Process

- Scripting languages are tools to reach a goal.
- They all have similarities and differences, but whichever language helps to reach that goal is what should be used.
- Do not get hung up on a specific language or its syntax.

Approach: Start with a roadmap of what you are trying to do.

For example...

Take a proxy server log file, filter for lines with source address "222.216.0.45," and then display the results ranked by the amount of times the destination addresses appear (from most active destination address to least active).

Step 1	Read in the log file.
Step 2	Filter out all entries with "222.216.0.45" in the source address field.
Step 3	Group by destination IP.
Step 4	Display in reverse order according to size of the group, along with the count.



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 71

Other Scripting Languages



Today, we also will be discussing Python...



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 72

As we have learned A **scripting or script language** is a [programming language](#) that supports **scripts** — programs written for a special [run-time environment](#) that automates the [execution](#) of tasks^[1] that could alternatively be executed one-by-one by a human operator. Scripting languages are often interpreted (rather than [compiled](#)). Primitives are usually the elementary tasks or [API](#) calls, and the language allows them to be combined into more complex programs. Environments that can be automated through scripting include [software applications](#), [web pages](#) within a [web browser](#), usage of the [shells](#) of [operating systems](#) (OS), [embedded systems](#), as well as numerous games.

VBScript ("[Microsoft Visual Basic Scripting Edition](#)") is an [Active Scripting](#) language developed by Microsoft that is modeled on Visual Basic.

Lasso is an [application server](#) and server management interface used to develop internet applications and is a [general-purpose, high-level programming language](#).

JavaScript ([/ˈdʒɑːvə_ˌskript/](#)),^[8] often abbreviated as **JS**, is a [high-level, interpreted programming language](#) that conforms to the [ECMAScript](#) specification. It is a programming language that is characterized

as [dynamic](#), [weakly typed](#), [prototype-based](#) and [multi-paradigm](#).

Prototype-based programming is a style of [object-oriented programming](#) in which behavior reuse (known as [inheritance](#)) is performed via a process of reusing existing [objects](#) via delegation that serve as [prototypes](#). This model can also be known as *prototypal*, *prototype-oriented*, *classless*, or *instance-based* programming. [Delegation](#) is the language feature that supports prototype-based programming.

Programming paradigms are a way to classify [programming languages](#) based on their features. Languages can be classified into multiple paradigms.

Some paradigms are concerned mainly with implications for the [execution model](#) of the language, such as allowing [side effects](#), or whether the sequence of operations is defined by the execution model. Other paradigms are concerned mainly with the way that code is organized, such as grouping a code into units along with the state that is modified by the code. Yet others are concerned mainly with the style of syntax and grammar.

Python – Overview



- Python is a programming language created by Guido van Rossum that was released in 1991.
- Python is an interpreted, high-level programming language used for “utility” or general-purpose coding.
- Python focuses on code readability.
- Python is open sourced and is available for most Windows and NIX platforms.



```
$ python
Python 2.4.3 (#1, Nov 11 2010, 13:34:43)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 73

Python is a programming language created by Guido van Rossum that was released in 1991.

Python is an interpreted, high-level programming language used for ‘utility’ or general purpose coding.

Python focuses on code readability

Trivia....

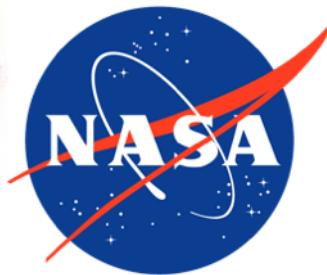
Van Rossum's long influence on Python is reflected in the title given to him by the Python community: [Benevolent Dictator For Life](#) (BDFL) – a post from which he gave himself permanent vacation on July 12, 2019.

Python – Overview (cont.)

Python is one of the most popular programming languages and is used by everyone from the Library of Congress to Google and NASA.



LIBRARY OF CONGRESS



Python – Overview (cont.)

```
1 #!/usr/bin/env python
2 print "Hello World!"
```



- Python is currently in version 2.7 release and is included on the laboratory machines, when needed.
- Python supports Object Oriented Programming (OOP), as well as Structured Programming.
- Python is what is referred to as an interpreted programming language.



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 75

Object-oriented programming (OOP) is a [programming paradigm](#) based on the concept of "[objects](#)", which can contain [data](#), in the form of [fields](#) (often known as *attributes*), and code, in the form of procedures (often known as [methods](#)). A feature of objects is an object's procedures that can access and often modify the data fields of the object with which they are associated (objects have a notion of "[this](#)" or "self"). In OOP, computer programs are designed by making them out of objects that interact with one another.[\[1\]](#)[\[2\]](#) OOP languages are diverse, but the most popular ones are [class-based](#), meaning that objects are [instances](#) of [classes](#), which also determine their [types](#).

Structured programming is a [programming paradigm](#) aimed at improving the clarity, quality, and development time of a [computer program](#) by making extensive use of the structured control flow constructs of selection ([if/then/else](#)) and repetition (while and [for](#)), [block structures](#), and [subroutines](#).

An **interpreted language** is a type of [programming language](#) for which most of its implementations execute instructions directly and freely, without previously [compiling](#) a [program](#) into [machine-language](#) instructions.

The [interpreter](#) executes the program directly, translating each statement into a sequence of one or more [subroutines](#), and then into another language (often [machine code](#)).

Python Philosophy...

Beautiful is better than ugly
Explicit is better than implicit
Simple is better than complex
Complex is better than complicated
Readability counts

Python – Syntax



Indentation

- Python groups code blocks using indentation and colons.
- This is different from many other programming languages, which use braces or curly brackets {}.
- In this example, the "else" actually belongs to the second "if" statement.

```
1  /*Bogus C code*/
2  if (FUN)
3      if (SUN)
4          baz (foo, bar);
5  else
6      qux();
7
8
8
10
11
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 76

Ashley can you offer some insight on this for the instructor notes?



Python – Syntax (cont.)

Indentation

- Now, see the same statement written in Python.
- It appears much less confusing.
- Clearly see that the “else” statement belongs to the second “if” statement.

```
1 #!/usr/bin/env python
2 if FUN:
3     if SUN:
4         baz (foo, bar)
5     else:
6         qux()
7
8
9
10
11
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 77

Ashley can you offer some insight on this for the instructor notes?



Python – Syntax (cont.)

Comments

- Comments are lines that exist in computer programs that are ignored by compilers and interpreters.
- Including comments in programs makes code more readable for humans, as it provides some information or explanation about what each part of a program is doing.
- Like many other languages, Python uses the pound sign (#) to denote a comment.
- Comment much, comment often, comment completely!

```
1 #!/usr/bin/env python
2 # Print 'Hello World!' to
3 console
4 print "Hello World!"
5
6
7
8
8
10
11
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 78

Another unique aspect to python is that any string not assigned to a variable is considered a comment



Python – Syntax (cont.)

Strings and Substrings

- Strings are amongst the most popular types in Python.
- Create them simply by enclosing characters in quotes.
- Python treats single quotes the same as double quotes.
- Creating strings is as simple as assigning a value to a variable.

```
1 #!/usr/bin/env python
2
3 var1 = 'Back to the Future!'
4 var2 = "Flux_Capacitor"
5
6
7
8
8
10
11
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 79

Ashley can you offer some insight on this for the instructor notes?

Python – Syntax (cont.)



Strings and Substrings

- Python does not support a character type; these are treated as strings of length one, thus also considered a substring.
- To access substrings, or components of a string, use the square brackets for slicing along with the index or indices to obtain a substring.
- In this example, get an output of different characters from the variables.

```
1 #!/usr/bin/env python
2
3 var1 = 'Back to the Future!'
4 var2 = "Flux_Capacitor"
5
6 print "var1[0]: ", var1[0]
7 print "var2[1:5]: ", var2[1:5]
8
9
10
11
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 80

Ashley can you offer some insight on this for the instructor notes?



Python – Syntax (cont.)

Strings and Substrings

- The output will look like this:
var1[0]: B
var2[1:5]: lux_C

```
1 #!/usr/bin/env python
2
3 var1[0]: B
4 var2[1:5]: lux_C
5
6
7
8
8
10
11
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 81

Ashley can you offer some insight on this for the instructor notes?



Python – Syntax (cont.)

Escape Characters

- Python also uses Escape Characters or non-printable characters that can be represented by backslash notation.
- This table contains a few of the possible escape characters to use in Python; a little online research will yield a more complete list.

Backslash Notation	Hex Character	Description
\a	0x07	Bell or Alert
\s	0x20	Space
\v	0x0b	Vertical Tab
\n	0x0a	Newline
\r	0x0d	Carriage Rtn
\b	0x08	Backspace
\nnn	Octal notation where n is the range 0-7	



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 82

Ashley can you offer some insight on this for the instructor notes?



Python – Syntax (cont.)

String Special Operators

- String special operators are used to manipulate variables in Python.
- The table contains a few of the string special operators that can be used in Python; once again, a more exhaustive list is available online.

Operator	Description
+	Concatenation: Adds values on either side of the operator
*	Repetition: Creates new strings, concatenating multiple copies of the same string
[]	Slice: Gives the Characters from the given index
[x:y]	Range Slice: Gives the Characters from the given range
%	Format: Performs string formatting



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 83

Ashley can you offer some insight on this for the instructor notes?

We will talk about sting formatting operators shortly

Python – Syntax (cont.)



String Special Operators – +

Operator	Description
+	Concatenation: Adds values on either side of the operator
*	Repetition: Creates new strings, concatenating multiple copies of the same string
[]	Slice: Gives the Character from the given index
[:]	Range Slice: Gives the Characters from the given range

```
1 #!/usr/bin/env python
2
3 var1 = 'hello'
4 var2 = 'python'
5
6 print var1 + var2
7 #this will be your output
8
9
10 hellopython
11
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 84

Ashley can you offer some insight on this for the instructor notes?

Python – Syntax (cont.)



String Special Operators – *

Operator	Description
+	Concatenation: adds values on either side of the operator
*	Repetition: Creates new strings, concatenating multiple copies of the same string
[]	Slice: Gives the Character from the given index
[:]	Range Slice: Gives the Characters from the given range

```
1 #!/usr/bin/env python
2
3 var1 = 'hello'
4 var2 = 'python'
5
6 print var2*
7 #this will be your output
8
9
10 pythonpython
11
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 85

Ashley can you offer some insight on this for the instructor notes?

Python – Syntax (cont.)



String Special Operators – []

Operator	Description
+	Concatenation: adds values on either side of the operator
*	Repetition: Creates new strings, concatenating multiple copies of the same string
[]	Slice: Gives the Characters from the given index
[:]	Range Slice: Gives the Characters from the given range

```
1 #!/usr/bin/env python
2
3 var1 = 'hello'
4 var2 = 'python'
5
6 print var2[3]
7 #this will be your output
8
9
10 h
11
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 86

Ashley can you offer some insight on this for the instructor notes?

Python – Syntax (cont.)



String Special Operators – [x:y]

Operator	Description
+	Concatenation: adds values on either side of the operator
*	Repetition: Creates new strings, concatenating multiple copies of the same string
[]	Slice: Gives the Character from the given index
[x:y]	Range Slice: Gives the Characters from the given range

```
1 #!/usr/bin/env python
2
3 var1 = 'hello'
4 var2 = 'python'
5
6 print var1[2:4]
7 #this will be your output
8
9
10 llo
11
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 87

Ashley can you offer some insight on this for the instructor notes?

Python – Syntax (cont.)



String Formatting Operator

- One of Python's most interesting features is the string format operator "%"
- This operator is unique to strings and makes up for the lack of having functions from the C programming languages "printf()" functions.
- Combining the "%" operator with different conversions, reformat data used in the string.
- For example, the "%s" string formatting operator will perform a string conversion via "str()" prior to formatting.
- The string formatting operator "%d" will output a signed decimal integer.

```
1 #!/usr/bin/env python
2
3 Print "My name is %s and my
4 height is %d ft!" % ('Marty',
5 6)
6
7 My name is Marty and my height
8 is 6 ft!
8
10
11
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 88

Ashley can you offer some insight on this for the instructor notes?

Signed and unsigned integers deal with how computer calculate the binary representations of the integers into binary formatting. It is beyond the scope of this class, but feel free to do your own research.



Python – Summary

- Python is a great flexible tool for automating complex or repetitive tasks using simple scripts that you can write yourself.
- An example of where Python might be an effective tool is the repetitive analysis of Windows log files from hundreds of host computers.

```
1 #!/usr/bin/env python
2
3
4
5
6
7
8
8
10
11
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 89

Ashley can you offer some insight on this for the instructor notes?

Signed and unsigned integers deal with how computer calculate the binary representations of the integers into binary formatting. It is beyond the scope of this class, but feel free to do your own research.

Python – Example

```
1 #!/usr/bin/env python
2 """ This simple script looks at all files starting with ac and
3 treats it as an standard web log file. Reports on all domains that
4 have visited a course description page"""
5
6 import os
7 import re
8 from hname import *
9
10 shorttable = {}
11 ctable = {}
12 lookfor = re.compile("GET\s/course/([a-z]+)")
13
14 for filename in os.listdir("."):
15     if not filename.startswith("ac"):
16         continue # eliminate files that are not logfiles
17     for line in open(filename).readlines():
18         parts = line.split(" ")
19     # How much of the name / IP address do we report?
20     host = hname(parts[0])
21     dsum = host.getShort()
22     # Count the host usage and log any course files
23     shorttable[dsum] = 1 + shorttable.get(dsum,0)
24     gotten = lookfor.findall(line)
25     if (not gotten): continue
26     ctable[dsum] = ctable.get(dsum,"") + " " + gotten[0]
27
28 def byhits(one,two):
29     global shorttable
30     return shorttable[two].__cmp__(shorttable[one])
31
32 visitors = ctable.keys()
33 visitors.sort(byhits)
34
35 for browser in visitors:
36     print browser,ctable[browser],shorttable[browser]
```

This simple script looks at all files starting with ac and treats it as an standard web log file.

Reports on all domains that have visited a course description page



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 90

Python – Example (cont.)

```
1  #!/usr/bin/env python
2  """ This simple script looks at all files starting with ac and
3  treats it as an standard web log file. Reports on all domains that
4  have visited a course description page"""
5
6  import os
7  import re
8  from hname import hname
9
10 13
11 14  for filename in os.listdir("."):
12 15      if not filename.startswith("ac"):
13 16          continue # eliminate files that are not log
14 17      for line in open(filename).xreadlines():
15 18          parts = line.split(" ")
16 19          # How much of the name / IP address do we report?
17 20          host = hname(parts[0])
18 21          dpsumm = host.getshort()
19 22          # Count the host usage and log any course files
20 23          shorttable[dpsumm] = 1 + shorttable.get(dpsum
21 24
22 25      if (not gotten): continue
23 26      ctable[dpsumm] = ctable.get(dpsumm,"") + " " + gotten[0]
24
25
26
27
28 def byhits(one,two):
29     global shorttable
30     return shorttable[two].__cmp__(shorttable[one])
31
32 visitors = ctable.keys()
33 visitors.sort(byhits)
34
35 for browser in visitors:
36     print browser,ctable[browser],shorttable[browser]
37
38
39
```

See the use of indentations.



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 91

Python – Example (cont.)

```
1  #!/usr/bin/env python
2  """ This simple script looks at all files starting with ac and
3  treats it as an standard web log file. Reports on all domains that
4  have visited a course description page"""
5
6  import os
7  import re
8  from hname import *
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
```

See the use of indentations and comments in this example script.

Ensuring that any future users can ascertain the reason for any particular piece of code



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 92



Scripting in Bash

Bash is a “sh” – compatible command language interpreter that executes commands read from standard input or from a file.

This allows advanced users to automate routine repetitive tasks that do not require the advanced functionality of a structured programming language.



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 93

Bash Logic



File Redirection	Test Operators	Variable Substitution
<pre>> file create (overwrite) file >> file append to file < file read from file a b Pipe 'a' as input to 'b'</pre>	<pre>if ["\$x" -lt "\$y"]; then # do something fi</pre>	<pre>\$V-default \$V, or "default" if unset \${V-default} \$V (set to "default" if unset) \${V-?err} \$V, or "err" if unset</pre>
Common Constructs	Numeric Tests	Conditional Execution
<pre>\$ while read f > do > echo "Line is \$f" > done < file</pre> <p>note: '\$' prompt becomes ">"</p> <pre>\$ grep foo myfile aroo foo foobar</pre> <pre>\$ cut -d : -5 /etc/passwd Steve Parker</pre> <pre>\$ cmd1 cmd2</pre> <pre>\$ cmd1 && cmd2</pre> <pre>case \$foo in a) echo "foo is A" ;; b) echo "foo is B" ;; *) echo "foo is not A or B" ;; esac</pre> <pre>myvar="ls"</pre> <pre>doubleit() { expr \$1 ^ 2 } doubleit 3 # returns 6</pre>	<pre>lt less than gt greater than eq equal to ne not equal ge greater or equal le less or equal</pre>	<pre>cmd1 cmd2 cmd1 && cmd2</pre> <pre>run cmd1; if fails, run cmd2 run cmd1; if ok, run cmd2</pre>
File Tests	String Tests	Files
	<pre>nt newer than d is a directory f is a file x executable r readable w writable</pre>	<pre>mv /src /dest move /src into /dest</pre> <p>is a* list files beginning with "a"</p> <p>is *a list files ending with "a"</p> <p>is -xr list oldest first, newest last</p> <p>is -Sr list smallest first, biggest last</p> <p>is -a list all files, including hidden</p> <pre>find /src -print cpio -pdvm copy /src into current directory, preserving links, special devices, etc.</pre>
Logical Tests	Arguments	Preset Variables
	<pre>&& logical AND logical OR !</pre>	<pre>\$SHELL what shell am I running? \$RANDOM provides random numbers \$\$ PID of current process \$\$? return code from last cmd \$\$! PID of last background cmd</pre>
Arguments	String Tests	Generally Useful Commands
	<pre>\$0 program name \$1 1st argument \$2 2nd argument ... \${#} no. of arguments \${*} all arguments</pre>	<pre>file fetchhosts determine file type basename basename strip directory name (ls) dirname dirname get directory name (lbn) ifconfig -a show all network adapters netstat -r show routers netstat -a show open ports date +%Y-%m-%d date Year, Month, Day date +%H:%M:%S Hours, Minutes wc -l count number of lines pwd present working directory</pre>



Foundational Analyst Security Trail

© Capgemini 2019. All rights reserved | 94



Bash Shell Scripting: Loops

- Loops in shell scripts allow for the creation of repeating processes that do not terminate until a specific condition is met.
- Example: Loop through the specified user directory, and determine if each line is a file or a directory.

```
#!/bin/bash
# search through all files in a directory
for file in /home/b2tf88mph/Documents
do
    if [ -d "$filename" ]
    then
        echo "$file is a directory"
    elif [ -f "$filename" ]
    then
        echo "$file is a file"
    fi
done
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 95



Bash Shell Scripting: Other Loop Types

Until

```
count = 1
until ["$*" = ""]
do
    echo "Count = $count"
    shift
    count = `expr $count + 1`
done
```

Do-while

```
while ["$*" != ""]
do
    echo "Argument value is: $1"
    shift
done
```

For

```
# expects array as input ($@)
for item in "$@"
do
    echo "Argument value is: $item"
done
```



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 96

Bash Shell Scripting: In Practical Application



Example: List all IP addresses that have visited a website via access.log.

```
#!/bin/bash
OUT=/tmp/spam.ip.%%
HTTPDLOG="/var/log/apache2/entry.log"
if [ -f $HTTPDLOG ];
then
    awk '{print}' $HTTPDLOG >$OUT
    awk '{ print $1}' $OUT | sort -n | uniq -c | sort -n
else
    echo "$HTTPDLOG not found. Make sure domain exists and setup correctly."
fi
/bin/rm -f $OUT
```



*Try running this against
dmz_web.log, and see
what IPs show up!*



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 97

Other Scripting Languages

There are many other scripting and programming languages, but the most common include the following:

1	VBScript	<ul style="list-style-type: none">Developed by MicrosoftModeled on Visual Basic
2	Lasso	<ul style="list-style-type: none">Server management interface used to develop Internet applicationsGeneral purpose and used for high-level programming
3	JavaScript (JS)	<ul style="list-style-type: none">One of the more popular languages for coding, especially for web appletsJS is prototype based and multi-paradigm



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 98

As we have learned A **scripting or script language** is a [programming language](#) that supports **scripts** — programs written for a special [run-time environment](#) that automates the [execution](#) of tasks^[1] that could alternatively be executed one-by-one by a human operator. Scripting languages are often interpreted (rather than [compiled](#)). Primitives are usually the elementary tasks or [API](#) calls, and the language allows them to be combined into more complex programs. Environments that can be automated through scripting include [software applications](#), [web pages](#) within a [web browser](#), usage of the [shells](#) of [operating systems](#) (OS), [embedded systems](#), as well as numerous games.

VBScript ("[Microsoft Visual Basic Scripting Edition](#)") is an [Active Scripting](#) language developed by Microsoft that is modeled on Visual Basic.

Lasso is an [application server](#) and server management interface used to develop internet applications and is a [general-purpose, high-level programming language](#).

JavaScript ([/dʒɑːvə_ſkript/](#)),^[8] often abbreviated as **JS**, is a [high-level, interpreted programming language](#) that conforms to the [ECMAScript](#) specification. It is a programming language that is characterized

as [dynamic](#), [weakly typed](#), [prototype-based](#) and [multi-paradigm](#).

Prototype-based programming is a style of [object-oriented programming](#) in which behavior reuse (known as [inheritance](#)) is performed via a process of reusing existing [objects](#) via delegation that serve as [prototypes](#). This model can also be known as *prototypal*, *prototype-oriented*, *classless*, or *instance-based* programming. [Delegation](#) is the language feature that supports prototype-based programming.

Programming paradigms are a way to classify [programming languages](#) based on their features. Languages can be classified into multiple paradigms.

Some paradigms are concerned mainly with implications for the [execution model](#) of the language, such as allowing [side effects](#), or whether the sequence of operations is defined by the execution model. Other paradigms are concerned mainly with the way that code is organized, such as grouping a code into units along with the state that is modified by the code. Yet others are concerned mainly with the style of syntax and grammar.

LAB004: Introduction to Scripting



© Capgemini 2019. All rights reserved | 99

Content Source:



Questions?



Foundational Analyst Security Training

© Capgemini 2019. All rights reserved | 100



People matter, results count.

This presentation contains information that may be privileged or confidential
and is the property of the CapGemini Group.
Copyright © 2019 CapGemini. All rights reserved.

About CapGemini

A global leader in consulting, technology services and digital transformation, CapGemini is at the forefront of innovation to address the entire breadth of clients' opportunities in the evolving world of cloud, digital and platforms. Building on its strong 50-year heritage and deep industry-specific expertise, CapGemini enables organizations to realize their business ambitions through an array of services from strategy to operations. CapGemini is driven by the conviction that the business value of technology comes from and through people. It is a multicultural company of 200,000 team members in over 40 countries. The Group reported 2017 global revenues of EUR 12.8 billion.

Learn more about us at

www.capgemini.com