COMMAND

BOOK OF SPELLS

LINE

Command Line Wizardry

Command Quick Reference

COMMAND LINE WIZARDRY

grep

Search the contents of files

`-c`	Count matching lines
`-E`	Enable extended regex
`-i`	Ignore case
`-P`	Enable Perl regex
`-R`	Recursively search

find

Search the system for files

`-exec`	Execute specified command for each file found
`-name`	Search by filename
`-size`	Search by file size
`-type`	Search by file type

file

Identify file type by magic number

`-f`	Read list from specified file
`-k`	List all type matches
`-z`	Look inside compressed files

Regular Expressions

Character	Meaning
`.`	Single wildcard character
`?`	Preceding item is optional
`*`	Match the preceding item zero or more times
`+`	Match the preceding item one or more times
`^`	Anchor pattern to the beginning of the string
`$`	Anchor pattern to the end of the string
`[ ]`	Character classes and ranges
`( )`	Group
`{ }`	Quantifier

cut

Extract portions of data from a file

`-c`	Character(s) to extract
`-d`	Field delimiter
`-f`	Field(s) to extract

head

Output the first few lines/bytes of file

`-n`	Number of lines to output
`-c`	Number of bytes to output

tail

Output the last few lines of a file

`-f`	Continuously monitor end of file
`-n`	Number of lines to output

uniq

Remove duplicate lines from a file

`-c`	Print number of times line is repeated
`-f`	Ignore the specified number of fields
`-i`	Ignore case

curl

Network data transfer

`-A`	Specify user agent
`-d`	Send using HTTP POST
`-G`	Send using HTTP GET
`-I`	Only fetch header
`-L`	Follow redirects
`-s`	Do not show errors

sort

Order the lines of a file

`-r`	Sort in descending order
`-f`	Ignore case
`-n`	Use numerical ordering
`-k`	Sort based on key
`-o`	Write output to file

join

Combine two files

`-j`	Join using specified field
`-t`	Field delimiter

vi commands

`b`	Back one word
`cc`	Replace current line
`cw`	Replace current word
`dw`	Delete current word
`dd`	Delete current line
`w`	Forward one word
`:q!`	Quit without save
`:wq`	Quit with save
`/`	Search forward
`?`	Search backward
`n`	Find next occurrence

xxd

Display file in binary or hexadecimal

`-b`	Display using binary rather than hex
`-l`	Print specified number of bytes
`-s`	Start printing at specified position

sdiff

Compare two files

`-a`	Treat files as text
`-i`	Ignore case
`-s`	Suppress common lines
`-w`	Max characters to output per line

tr

Translate one character to another

`-d`	Delete character
`-s`	Squeeze repeated characters

wevtutil

View and manage Windows logs

`el`	Enumerate available logs
`qe`	Query a log's events
`/c`	Specify max number of events
`/f`	Format output as XML
`/rd`	Read direction, if true read most recent first

base64

Encode/decode data using Base64

`-d`	Decode

AWK Quick Reference

COMMAND LINE WIZARDRY

Syntax

```
awk 'pattern {action}
     pattern {action}
           ...
     pattern {action}'
```

One line of input may match several patterns, and will execute the corresponding actions in order. No pattern means all input lines; no action will print the whole line.

Referencing Fields

The `awk` command parses input lines into fields using whitespace as the default field delimiter

Syntax	Field
`$1`	Field 1
`$2`	Field 2
`$n`	N'th field
`$0`	Entire Line
`$NF`	Last field
`NF`	Number of fields

Use the `-F` option to specify a field delimiter other than whitespace.

Patterns

Testing Equality

```
$1=="Hello"
```

Conditional Operators

!	!=	==	<	<=	>	>=

Regex can also be used as a pattern by enclosing the expression in `/ /`

`BEGIN {action}`
Action executes before any input is processed

`END {action}`
Action executes after all input is processed

Variables

Assigning a Variable

```
MYVAR="Hello"

MYVAR=$1
```

Referencing a Variable

```
print MYVAR

MYVAR=MYVAR + $3
```

Arrays

Assigning an Array Element

```
name["index"] = "value"
```

Accessing an Element

```
print name["index"]
```

Deleting an Element

```
delete name["index"]
```

Deleting an Entire Array

```
delete name
```

If Statement

```
if (Condition)
  body
else
  body
```

Enclose the statement body in curly-brackets if it is more than one line.

Output

Writing to standard out

```
print "Hello World"

print $1

printf "Hello World\n"

printf "%s %d\n", $1, var
```

While Loop

```
while (Condition)
{
  body
}
```

For Loop

Numerical looping

```
for (i=0; i < 100; i++)
{
  body
}
```

Iterating over an array

```
for (i in array)
{
  body with array[i]
}
```

Keywords and Functions

`next`
Skip to next input line

`nextfile`
Skip to next input file

`exit`
Exit the awk program

`index(str1,str2)`
Returns index in `str1` where `str2` occurs; returns 0 if not found.

`int(str)`
Truncate to integer

`length(str)`
String length of argument

`rand()`
Random number between 0 and 1

`split(str,array,separator)`
Separate `str` into `array` using `separator`. FS is used if no separator is given.

`system(cmd)`
Execute command, returns exit status

Bash Quick Reference

COMMAND LINE WIZARDRY

Output

Writing to the screen

```
echo 'Hello World'

printf 'Hello World\n'
```

Format Strings

Format strings for `printf`

`%s`	String
`%d`	Decimal
`%f`	Floating point
`%x`	Hexadecimal
`\n`	Newline
`\r`	Carriage return
`\t`	Horizontal tab

Positional Parameters

Script parameters

`$#`	Number of parameters
`$0`	Name of the script
`$1`	First parameter
`$2`	Second parameter …

Default parameters

```
MYVAR=${1:-Cake}
```

Note: If parameter 1 is unset, the value of `MYVAR` will default to `Cake`

User Input

Read from stdin

```
read MYVAR
```

Prompting

```
read -p 'Name: ' USRNAME
```

Reading a File

```
while IFS="" read MYLINE
do
  echo "$MYLINE"
done < "somefile.txt"
```

Note: `IFS=""` preserves whitespace

Variables

Declaring a Variable

```
MYVAR='Hello'
```

Referencing a Variable

```
echo $MYVAR

echo "$MYVAR World"
```

Assigning Shell Output

```
CMDOUT=$(pwd)
```

If Statements

Command conditional (`cmd` will return `0` if success)

```
if cmd
then
  some cmds
else
  other cmds
fi
```

File and numeric conditionals

```
if [[ -e $FILENAME ]]
then
  echo $FILENAME exists
fi
```

File Test	Use
`-d`	Directory exists
`-e`	File exists
`-r`	File is readable
`-w`	File is writable
`-x`	File is executable

Numeric Test	Use
`-eq`	Equal
`-gt`	Greater than
`-lt`	Less than

While Loop

```
i=0
while (( i < 1000 ))
do
  echo $i
  let i++
done
```

For Loop

Numerical looping

```
for ((i=0; i < 1000; i++))
do
  echo $i
done
```

Iterating over a list

```
for VAL in 20 3 dog 7
do
  echo $VAL
done
```

Case Statement

```
case $MYVAR in
  "carl")
    echo 'Hi Carl!'
    ;;
  "paul")
    echo 'Hi Paul!'
    ;;
  *)  # default
    echo 'Goodbye'
    exit
    ;;
esac
```

Functions

Declaring a function

```
function myfun ()
{
  # function body
  echo 'This is myfun()'
}
```

Invoking a function

```
myfun param1 param2
```

Windows Batch Script Quick Reference

COMMAND LINE WIZARDRY

Output

Write to the screen

```
echo Hello World
```

Turn off command echoing

```
@echo off
```

Comments

```
rem This is a comment

::This is a comment too
```

Positional Parameters

%0	Name of the script as called
%1	First parameter
%~1	First parameter, quotes removed
%2	Second parameter …
%*	All parameters

Note - Use `shift` to move all parameter indexes one to the left.

User Input

Read user input

```
set /p USERIN=
```

Prompting

```
set /p USERIN="Name: "
```

Wait for user input

```
pause
```

Goto

```
if %MYVAR%==1 goto Place1
if %MYVAR%==2 goto Place2

:Place1
  echo Place 1
  goto END

:Place2:
  echo Place 2

:END
```

Variables

Declaring a Variable

```
set MYVAR=Hello
```

Referencing a Variable

```
echo %MYVAR%

echo %MYVAR% World!
```

If Statements

Single-line

```
if %VAR%==%VAL% echo True
```

Multi-line

```
if %VAR%==%VAL% (
  echo True
  echo VAR is 5
) else (echo False)
```

File Exists

```
if exist file.txt (
  echo File exists
)
```

Comparison Operators

Comparator	Use
EXIST	File exists
==	Equal
EQU	Equal
NEQ	Not equal
LSS	Less than
LEQ	Less than or equal
GTR	Greater than
GEQ	Greater than or equal
NOT	Inverse

Reading a File

```
for /f %%L in (f.txt) do (
  echo %%L
)
```

While Loop

```
set i=0

:Top
  echo %i%
  set /a "i = i + 1"
  if %i% lss 10 goto Top
```

Note – This is simulated while loop functionality

For Loop

Numerical looping

```
for /l %%i in (0,1,10) do (
  echo %%i
)
```

Iterating over a list

```
for %%i in (a, 7, t) do (
  echo %%i
)
```

Functions

Declaring a function

```
:Function1
  echo This is function 1
  echo 1st parameter: %1
  echo 2nd parameter: %2
  exit /b 0
```

Invoking a function

```
call :Function1 p1 p2
```

Error Handling

Execute if command is successful

```
whoami && echo Success!
```

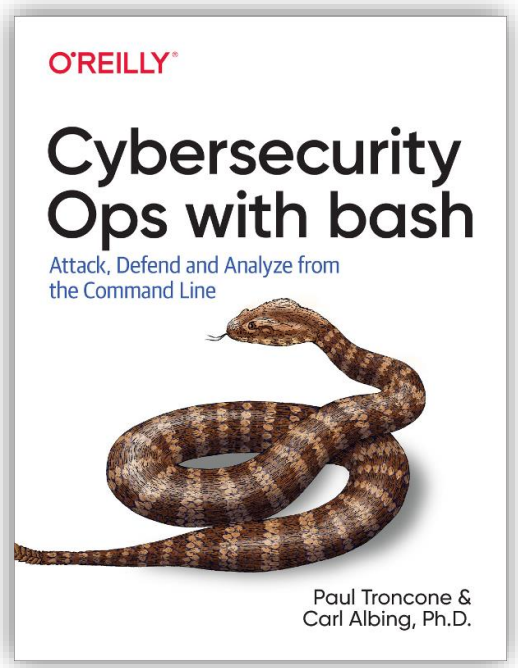
Execute if command fails

```
whoami || echo Failure!
```

Previous command execution result

```
whoami
echo %errorlevel%
```

Level Up!



Learn how you can leverage the command line to enhance your capabilities as a security practitioner, penetration tester, or system administrator.

Master the Command Line

If you hope to outmaneuver threat actors, speed and efficiency are key components of cybersecurity operations. Mastery of the standard command line interface (CLI) is an invaluable skill in times of crisis because no other software application can match the CLI's availability, flexibility, and agility. This practical guide shows you how to use the CLI with the bash shell to perform tasks such as data collection and analysis, intrusion detection, reverse engineering, and administration.

https://www.commandlinewizardry.com/cyberops