iGNITE Technologies

A Detailed Guide on CRUNCH

Contenido

Introducción

Muchas veces los atacantes tienen la necesidad de generar una lista de palabras basada en ciertos criterios que se requieren para escenarios de pentest como la pulverización de contraseñas o la fuerza bruta. Otras veces podría ser una situación trivial como la enumeración de directorios. Crunch es una herramienta desarrollada en C por bofh28 que puede crear listas de palabras personalizadas y altamente modificables que pueden ayudar a un atacante en las situaciones mencionadas anteriormente. Toma como entrada el tamaño mínimo, el tamaño máximo y los conjuntos de caracteres alfanuméricos y genera cualquier combinación posible de palabras con o sin significado y la escribe en un archivo de texto. En este artículo, demostraremos los filtros crujientes en detalle.

Instalación y primera ejecución.

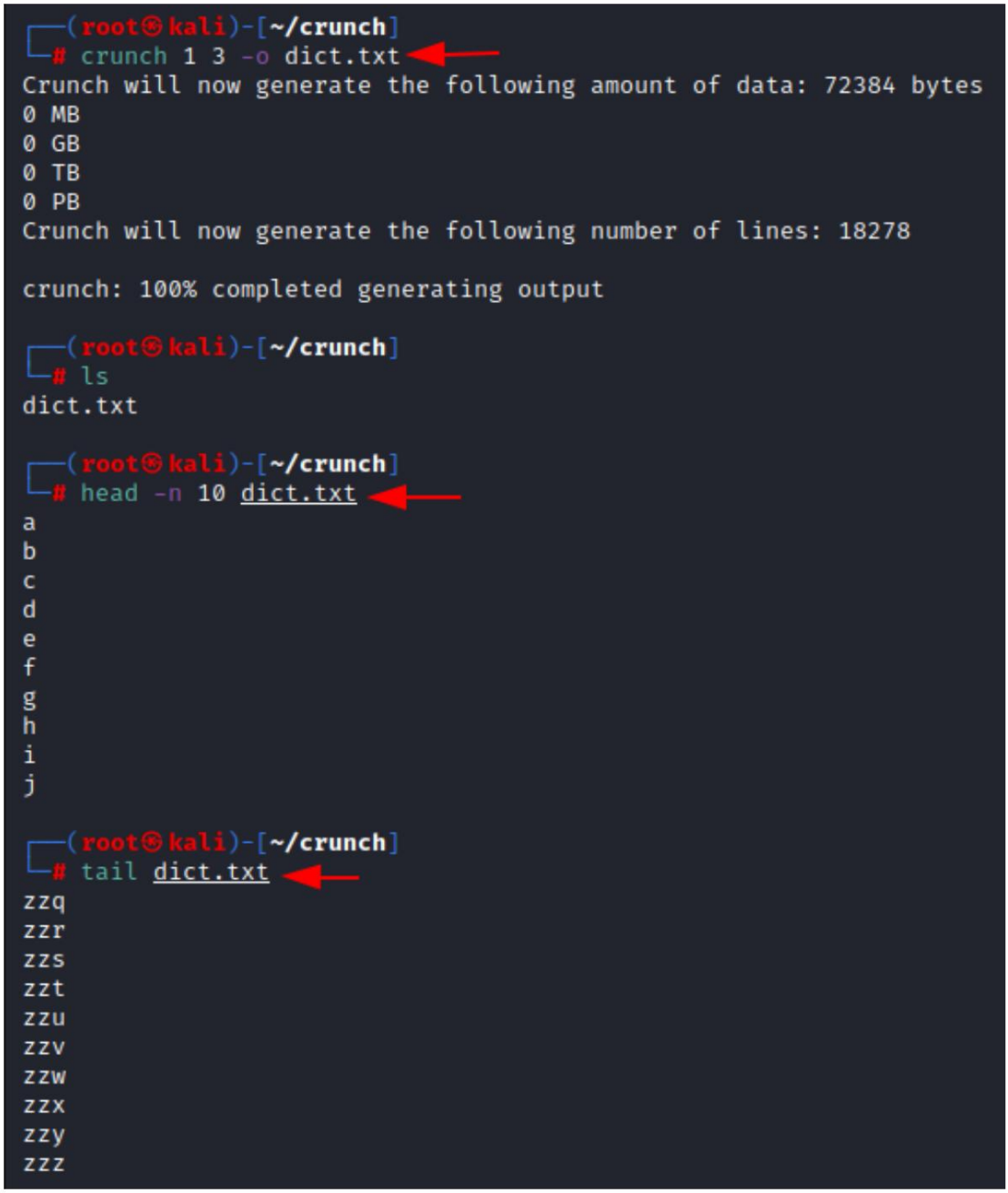
Crunch se instala de forma predeterminada en Kali Linux, pero se puede instalar usando el administrador de paquetes apt usando

```
crisis de instalación adecuada
```

Una vez instalado, podemos ejecutar Crunch para generar una lista de palabras. Cuando ingresamos el tamaño mínimo y máximo de la palabra que se generará y solo el archivo de salida, automáticamente toma alfabetos en minúsculas como conjuntos de caracteres y genera palabras.

Por ejemplo, aquí se generan de 1 a 3 caracteres por palabra en minúsculas y se almacenan en el archivo dict.txt.

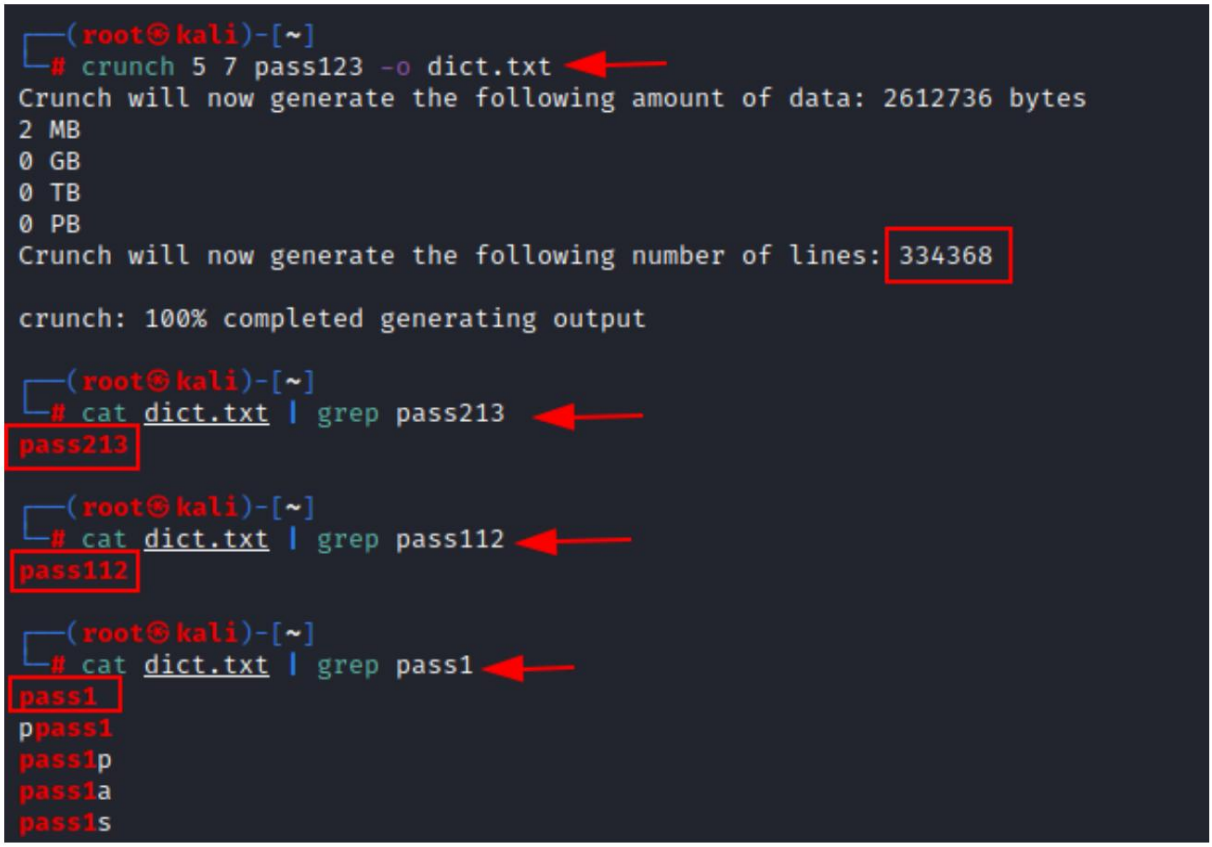
```
crujido 1 3 -o dict.txt
```



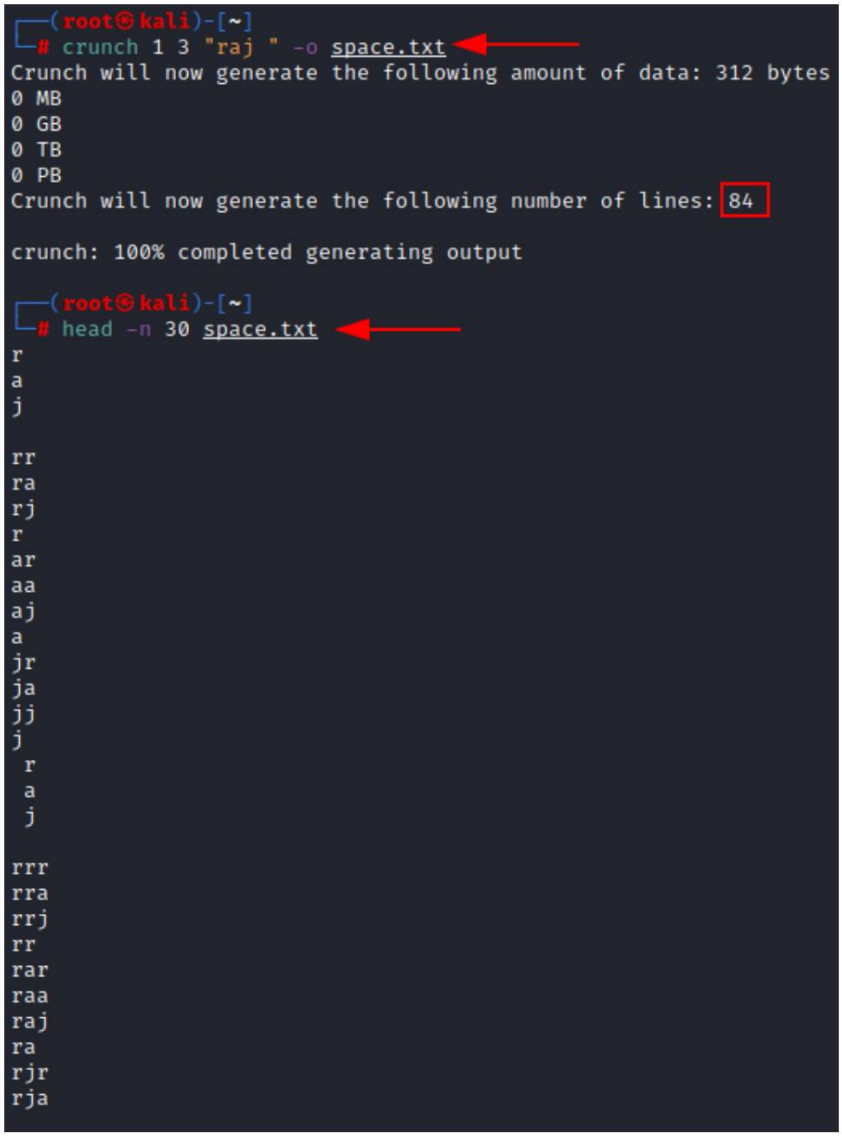
Caracteres alfanuméricos definidos

Un usuario también puede definir los caracteres seleccionados que se utilizarán al generar una lista de palabras. Aquí, se generan caracteres de tamaño mínimo 5 y tamaño máximo 7 por palabra mientras se utilizan los caracteres "p, a, s, s, 1, 2 y 3" como entrada. De ahí que el diccionario comience con “ppppp, ppppa…”. Y termina con “3333333” y contiene combinaciones como pass213, pass1, etc.

```
crujido 5 7 pase123 -o dict.txt
```



> Lista de palabras de caracteres espaciales
>
> Un buen truco es incluir espacios en la lista de palabras. Muchas veces necesitamos espacios en los escenarios para las contraseñas y muchas listas de palabras genéricas o herramientas no tienen esta característica. En crisis, podemos definir el espacio como un carácter colocando un espacio después del conjunto de caracteres que se utilizará. De 1 a 3 caracteres por palabra, incluido el espacio, podemos hacer esto:
>
> crujido 1 3 "raj " -o espacio.txt



Ver conjuntos de caracteres disponibles

En el directorio /usr/share/crunch , se puede encontrar un archivo de lista (charset.lst) que menciona todos los diferentes conjuntos de caracteres admitidos por crunch. Esto es muy útil como referencia inmediata. Se pueden especificar manualmente conjuntos de caracteres o incluso utilizar los nombres en clave escritos a la izquierda. Es

Aunque es bastante sencillo de entender. La descripción de cada juego de caracteres se proporciona a continuación:

Description:	charset code-name
Lowercase alphabets:	lalpha
Uppercase alphabets:	ualpha
Mixture of lowercase and uppercase alphabets:	mixalpha
Numbers:	numeric
Top 14 symbols:	symbol14
All symbols:	all
Then, we can make combinations of character sets too. It can be understood like the following:	
To include space with charset:	charset-space. For example, lalpha-space will have {abcdefghijklmnopqrstuvwxyz } (note space at the end)
To include numeric with charset:	charset-numeric
Combination of charset plus numeric plus space:	charset-numeric-space
To include top 14 symbols:	charset-symbol14
To include all symbols:	charset-all
To include all symbols and space:	charset-all-space

Para ver el archivo del juego de caracteres:

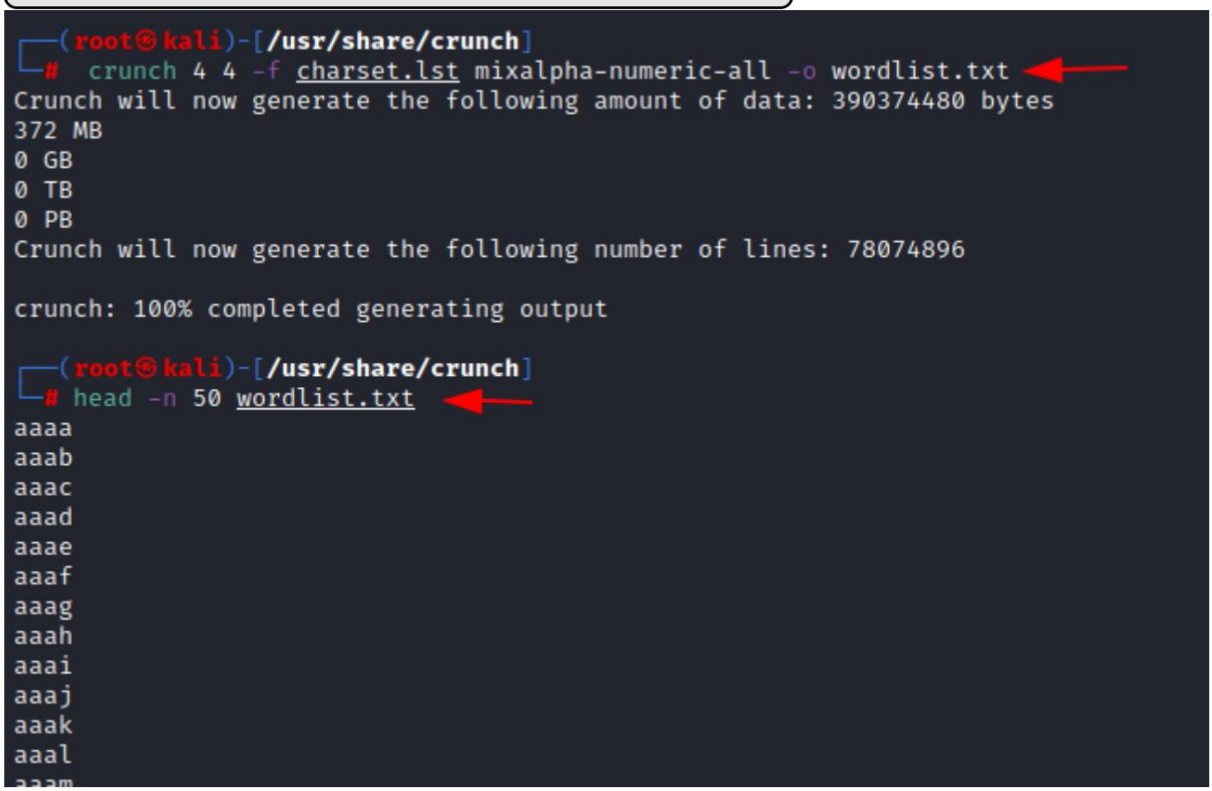
gato /usr/share/crunch/charset.lst

```
┌──(root㉿kali)-[/usr/share/crunch]
└─# cat charset.lst
# charset configuration file for winrtgen v1.2 by Massimiliano Montoro (mao@oxid.it)
# compatible with rainbowcrack 1.1 and later by Zhu Shuanglei <shuanglei@hotmail.com>


hex-lower                      = [0123456789abcdef]
hex-upper                      = [0123456789ABCDEF]

numeric                        = [0123456789]
numeric-space                  = [0123456789 ]

symbols14                      = [!@#$%^&*()-_+=]
symbols14-space                = [!@#$%^&*()-_+= ]

symbols-all                    = [!@#$%^&*()-_+=~`[]{}|\:;"'<>,.?/]
symbols-all-space              = [!@#$%^&*()-_+=~`[]{}|\:;"'<>,.?/ ]

ualpha                         = [ABCDEFGHIJKLMNOPQRSTUVWXYZ]
ualpha-space                   = [ABCDEFGHIJKLMNOPQRSTUVWXYZ ]
ualpha-numeric                 = [ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789]
ualpha-numeric-space           = [ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 ]
ualpha-numeric-symbol14        = [ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()-_+=]
ualpha-numeric-symbol14-space  = [ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()-_+= ]
ualpha-numeric-all             = [ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()-_+=~`[]{}|\:;"'<>,.?/]
ualpha-numeric-all-space       = [ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()-_+=~`[]{}|\:;"'<>,.?/ ]

lalpha                         = [abcdefghijklmnopqrstuvwxyz]
lalpha-space                   = [abcdefghijklmnopqrstuvwxyz ]
lalpha-numeric                 = [abcdefghijklmnopqrstuvwxyz0123456789]
lalpha-numeric-space           = [abcdefghijklmnopqrstuvwxyz0123456789 ]
lalpha-numeric-symbol14        = [abcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()-_+=]
lalpha-numeric-symbol14-space  = [abcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()-_+= ]
lalpha-numeric-all             = [abcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()-_+=~`[]{}|\:;"'<>,.?/]
lalpha-numeric-all-space       = [abcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()-_+=~`[]{}|\:;"'<>,.?/ ]
```

Usar juegos de caracteres de nombre en clave

Estos nombres en clave se pueden utilizar al crear archivos de diccionario. Por ejemplo, para crear una lista de palabras de 4 caracteres por palabra usando una combinación de alfabetos, números y caracteres especiales, se puede especificar el archivo charset.lst usando la opción "-f" y luego especificar la palabra clave "mixalpha-numeric-all".

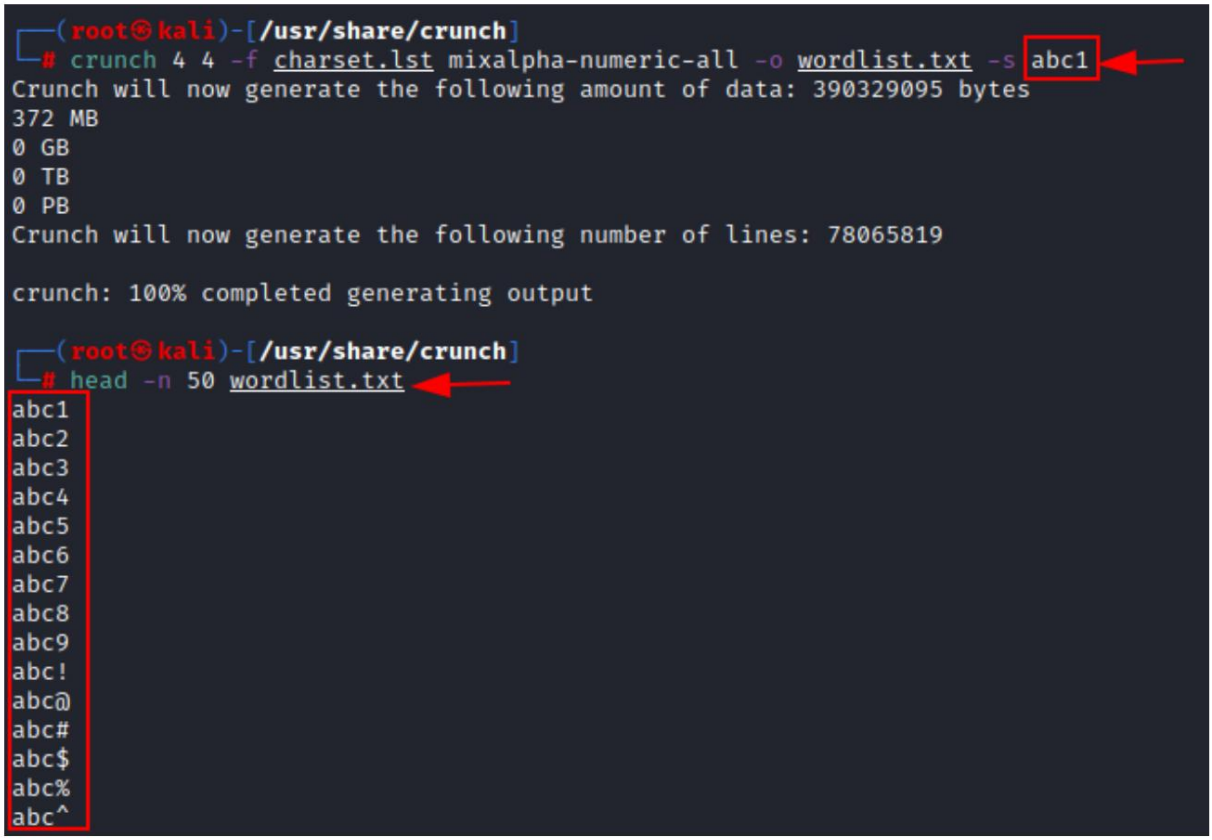
```
crunch 4 4 -f charset.lst mixalfa-numérico-todo -o lista de palabras.txt
```



Bloque de inicio en listas de palabras

Se puede definir un bloque de inicio utilizando el filtro "-s". Al usar esto, podemos definir desde dónde debe comenzar a generarse una lista de palabras. Esto es útil para descartar combinaciones no deseadas. Por ejemplo, para comenzar una lista de palabras desde abc1, se pueden crear 4 caracteres por palabra, incluidos caracteres alfanuméricos y especiales, como se muestra a continuación. De esta manera, el diccionario comienza con “abc1, abc2,..abd1, abd2…” y termina en “////”

```
crunch 4 4 -f charset.lst mixalfa-numérico-todo -o lista de palabras.txt -s abc1
```



Creando un diccionario con varios patrones

Tenga en cuenta que los siguientes símbolos, cuando se definen como entrada en juegos de caracteres, significan lo siguiente:

@ insertará caracteres en minúscula

, insertará caracteres en mayúsculas

% insertará números

^ insertará símbolos

Ahora, si un usuario quiere crear una palabra con 3 caracteres con el primer carácter en minúscula, el número como segundo carácter y el símbolo como tercero, puede especificar esto:

crujido -t @%^ -o dict.txt

Con “-t” como bandera para proporcionar los símbolos. Si no va a utilizar un conjunto de caracteres en particular, utilice un signo más como marcador de posición.

Posicionamiento del operador +: el operador + se puede utilizar cuando no se utilizan conjuntos de caracteres específicos y se puede reemplazar cualquier valor por el mismo. Pero esto es en el siguiente orden:

Alfabetos en minúsculas, alfabetos en mayúsculas, números, símbolos

Por ejemplo,

crujido 3 3 + + 123 +

Esto tomaría la siguiente entrada:

Minúscula: abcdefghijklmnopqrstuvwxyz

Mayúsculas: ABCDEFGHIJKLMNOPQRSTUVWXYZ

Números: 123

Símbolos: !@#$%^&*()-_+=~`[]{}|\:;"'<>,.?/

Caso 1: Palabra fija + 3 números

Digamos que si queremos arreglar las primeras 3 letras como "raj" e insertar combinaciones aleatorias de dígitos en los últimos 3 lugares en una lista de palabras de 6 caracteres por palabra, se puede hacer especificando el patrón sin el uso de comas como arriba en " -t" filtro.

```
crujido 6 6 -t raj%%% -o num.txt
```

```
┌──(root㉿kali)-[~/crunch]
└─# crunch 6 6 -t raj%%% -o num.txt

Crunch will now generate the following amount of data: 7000 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 1000

crunch: 100% completed generating output

┌──(root㉿kali)-[~/crunch]
└─# head num.txt
raj000
raj001
raj002
raj003
raj004
raj005
raj006
raj007
raj008
raj009
```

Caso 2: Palabra fija + 3 alfabetos en mayúsculas

Digamos que si queremos arreglar las primeras 3 letras como "raj" e insertar combinaciones aleatorias de alfabetos en mayúsculas en los últimos 3 lugares en una lista de palabras de 6 caracteres por palabra, podemos hacerlo mediante

```
crujido 6 6 -t raj,,, -o superior.txt
```

```
┌──(root㉿kali)-[~/crunch]
└─# crunch 6 6 -t raj,,, -o upper.txt

Crunch will now generate the following amount of data: 123032 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 17576

crunch: 100% completed generating output

┌──(root㉿kali)-[~/crunch]
└─# head upper.txt
rajAAA
rajAAB
rajAAC
rajAAD
rajAAE
rajAAF
rajAAG
rajAAH
rajAAI
rajAAJ
```

Caso 3: Palabra fija + 3 alfabetos en minúsculas

Digamos que si queremos arreglar las primeras 3 letras como "raj" e insertar combinaciones aleatorias de alfabetos en minúsculas en los últimos 3 lugares en una lista de palabras de 6 caracteres por palabra, podemos hacerlo mediante

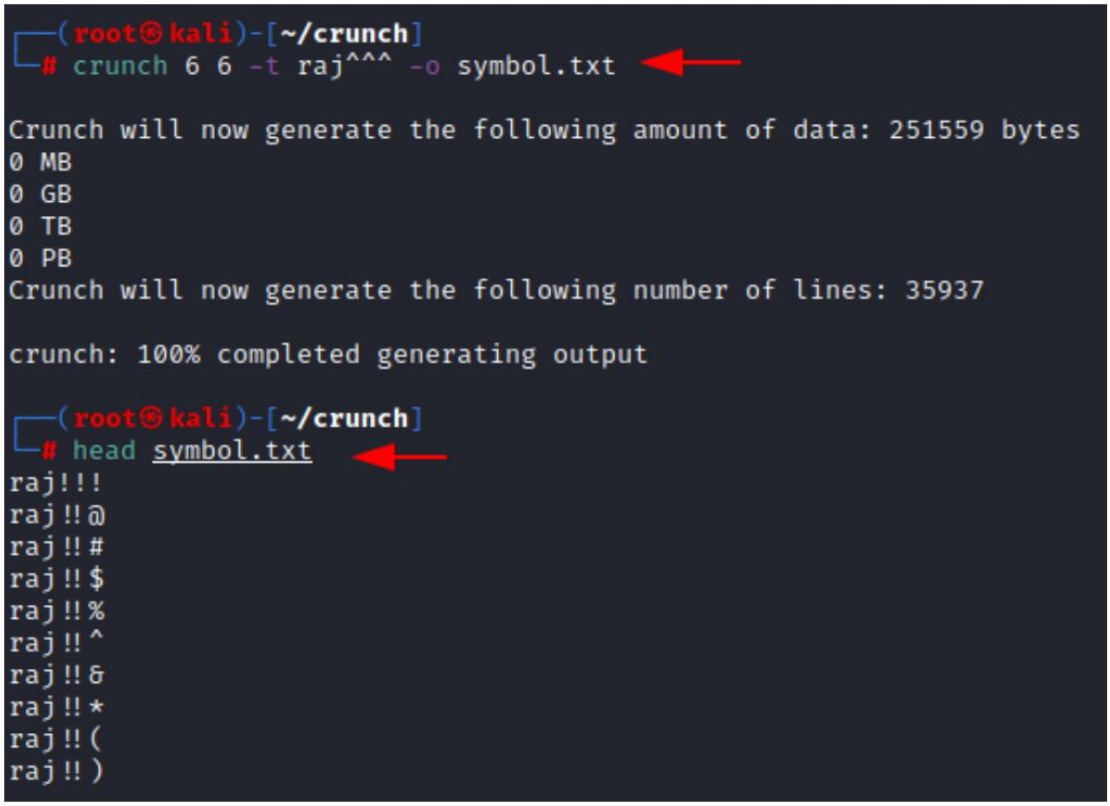
```
crujido 6 6 -t raj@@@ -o lower.txt
```

```
┌──(root㉿kali)-[~/crunch]
└─# crunch 6 6 -t raj@@@ -o lower.txt

Crunch will now generate the following amount of data: 123032 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 17576

crunch: 100% completed generating output

┌──(root㉿kali)-[~/crunch]
└─# head lower.txt
rajaaa
rajaab
rajaac
rajaad
rajaae
rajaaf
rajaag
rajaah
rajaai
rajaaj
```

Caso 4: Palabra fija + 3 símbolos

Digamos que si queremos arreglar las primeras 3 letras como "raj" e insertar combinaciones aleatorias de caracteres especiales en los últimos 3 lugares en una lista de palabras de 6 caracteres por palabra, podemos hacerlo mediante

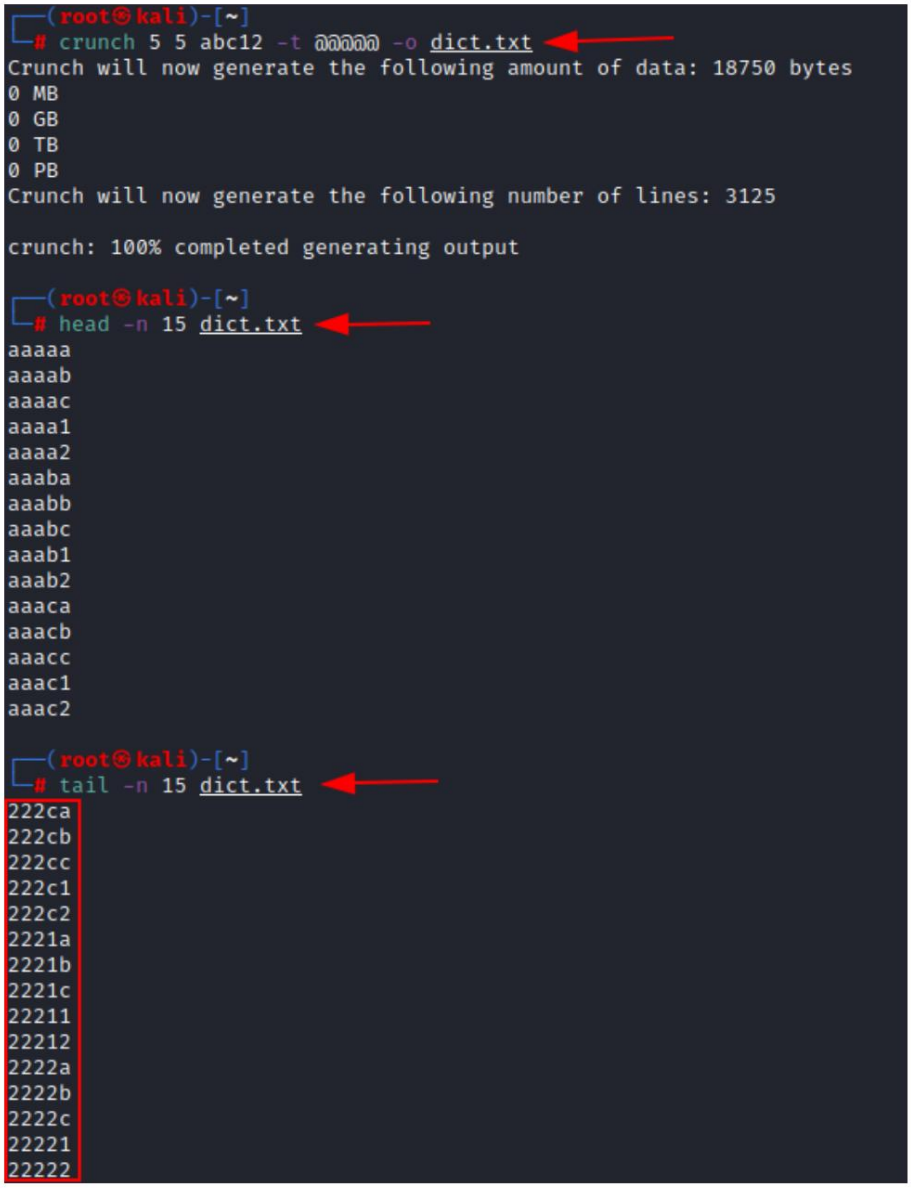
```
crujido 6 6 -t raj^^^ -o símbolo.txt
```



Caso 5: patrón fijo de marcador de posición

Digamos que en lugar del marcador de posición en minúscula ingresamos abc12 y con "-t" ingresamos @, entonces el patrón también contendrá 1 y 2 aunque acabamos de ingresar el indicador "@". Vea el siguiente ejemplo:

```
crujido 5 5 abc12 -t @@@@@ -o dict.txt
```



Caso 6: Alfabeto en minúsculas (a,b o c) + número (1,2 o 3) + símbolo (CUALQUIER)

Ahora, un usuario también puede proporcionar un conjunto de caracteres a partir del cual se creará un patrón. En el siguiente ejemplo, se han utilizado abc y 123. También se utiliza un operador "+" que indica que el indicador de patrón para el cual no se proporciona el juego de caracteres se tratará como valor "CUALQUIER".

Entonces, si un usuario desea crear un diccionario con el primer carácter en minúscula, un número como segundo carácter y un símbolo como tercero, pero solo "a,b o c" como caracteres, "1,2 o 3" como números y cualquier símbolo aleatorio en el último posición respectivamente, puede hacer lo siguiente:

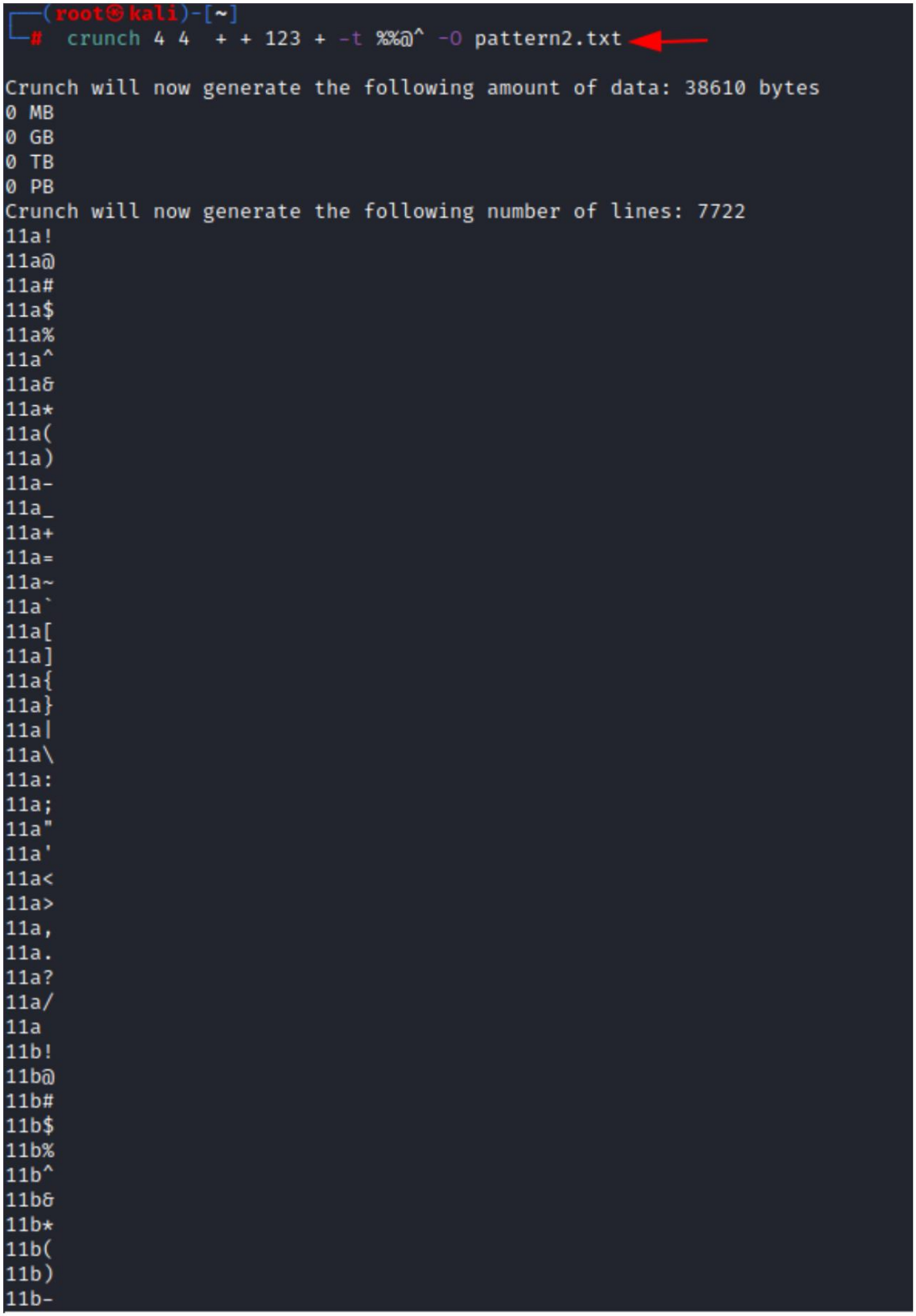
```
crujido 3 3 abc + 123 -t @%^ -o patrón.txt
```

```
┌──(root㉿kali)-[/usr/share/crunch]
└─# crunch 3 3 abc + 123 -t @%^ -o pattern.txt

Crunch will now generate the following amount of data: 1188 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 297

crunch: 100% completed generating output

┌──(root㉿kali)-[/usr/share/crunch]
└─# head -n 30 pattern.txt
a1!
a1@
a1#
a1$
a1%
a1^
a1&
a1*
a1(
a1)
a1-
a1_
a1+
a1=
a1~
a1`
a1[
a1]
a1{
a1}
a1|
a1\
a1:
a1;
a1"
a1'
a1<
a1>
a1,
a1.
```

Caso 7: Dos números (1,2 o 3) + alfabeto en minúsculas (CUALQUIER) + símbolo (CUALQUIER)

De manera similar, para crear un patrón de 4 caracteres por palabra de 2 dígitos (que contenga solo 1, 2 o 3) + alfa minúscula + símbolo, podemos hacer esto:

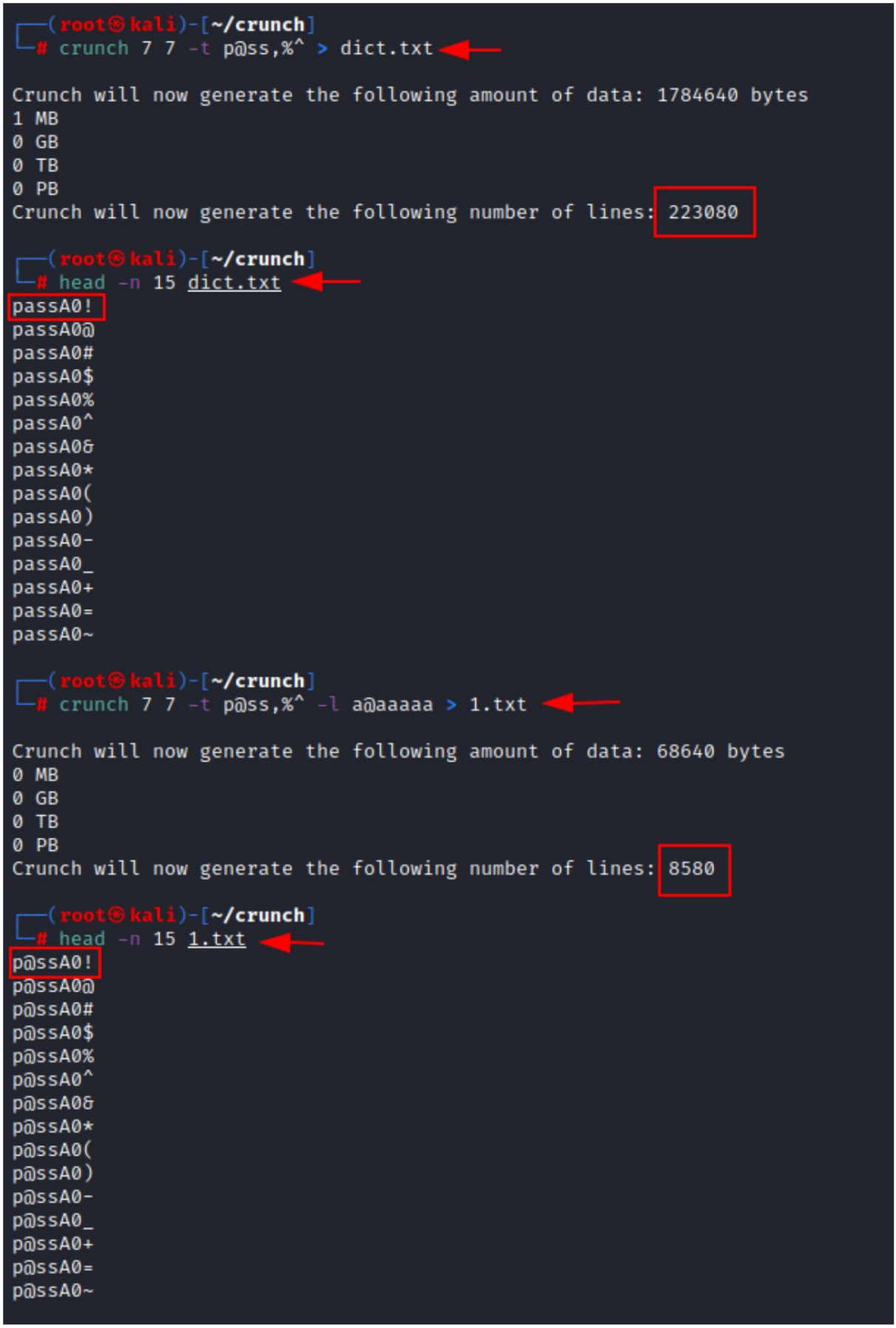
```
crujido 4 4 + + 123 + -t %%@^ -O patrón2.txt
```



Caso 8: Tratar los símbolos como literales

Cuando se usa "-l" de acuerdo con el filtro "-t", le indica a Crunch qué símbolos deben tratarse como literales. Por ejemplo, sabemos que @ se usa para indicar una letra minúscula. Entonces, si queremos generar una lista de palabras de 7 caracteres por palabra usando la palabra “p@ss” fija, considerará @ como un indicador de patrón de alfabetos en minúsculas. A partir de entonces, se puede utilizar el filtro -l para definir qué carácter se tratará como literal y no se convertirá como patrón. Esto se puede hacer como:

```
crujido 7 7 -tp@ss,%^ > dict.txt
crujido 7 7 -tp@ss,%^ -la@aaaaa > 1.txt
```



Invertir la lista de palabras

Una lista de palabras generada corrige, de forma predeterminada, los primeros caracteres y crea combinaciones en el último carácter. Por ejemplo, una lista de palabras que contiene “a, byc” tiene

aaa
ab
acac
aba
tejido
a B C
acá
…

Pero esto se puede invertir usando la opción "-i". Crunch arreglaría primero la última letra y haría combinaciones a partir de las primeras letras. Por ejemplo, un diccionario de 5 caracteres por palabra que tiene 3 alfabetos, 2 dígitos e invertido se ve así:

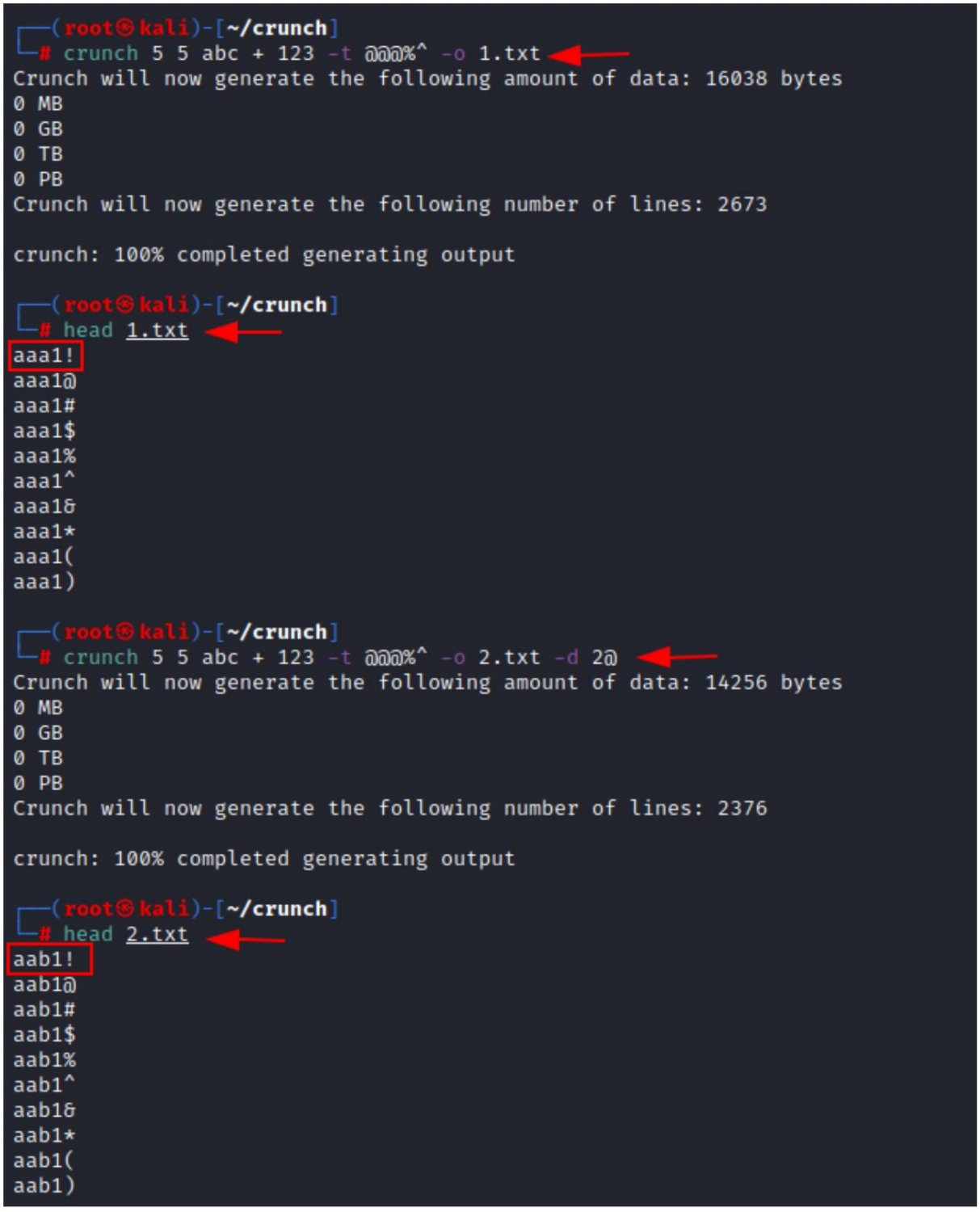
```
crujido 5 5 abc12 -t @@@%% -o dict.txt crujido
5 5 abc12 -t @@@%% -i -o invert.txt
```

```
┌──(root㉿kali)-[~]
└─# crunch 5 5 abc12 -t @@@%% -o dict.txt
Crunch will now generate the following amount of data: 75000 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 12500

crunch: 100% completed generating output

┌──(root㉿kali)-[~]
└─# head -n 20 dict.txt
aaa00
aaa01
aaa02
aaa03
aaa04
aaa05
aaa06
aaa07
aaa08
aaa09
aaa10
aaa11
aaa12
aaa13
aaa14
aaa15
aaa16
aaa17
aaa18
aaa19

┌──(root㉿kali)-[~]
└─# crunch 5 5 abc12 -t @@@%% -i -o invert.txt
Crunch will now generate the following amount of data: 75000 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 12500

crunch: 100% completed generating output

┌──(root㉿kali)-[~]
└─# head -n 20 invert.txt
aaa00
baa00
caa00
1aa00
2aa00
aba00
bba00
cba00
```

Limitar patrones duplicados

Un usuario puede poner un límite a la cantidad de caracteres que pueden aparecer juntos. Por ejemplo, para crear una lista de palabras de 5 caracteres por palabra usando 3 alfabetos en minúsculas, se puede realizar 1 número y 1 símbolo como primer comando. Pero si un usuario desea limitar la aparición de caracteres duplicados juntos a solo 2 lugares, puede usar el operador "-d". Observe cómo en el primer comando ocurrieron 3 "a", pero en el segundo comando los duplicados se limitan a solo 2, por lo que solo se produjeron 2 "a".

```
crujido 5 5 abc + 123 -t @@@%^ -o 1.txt
crujido 5 5 abc + 123 -t @@@%^ -o 2.txt -d 2@
```

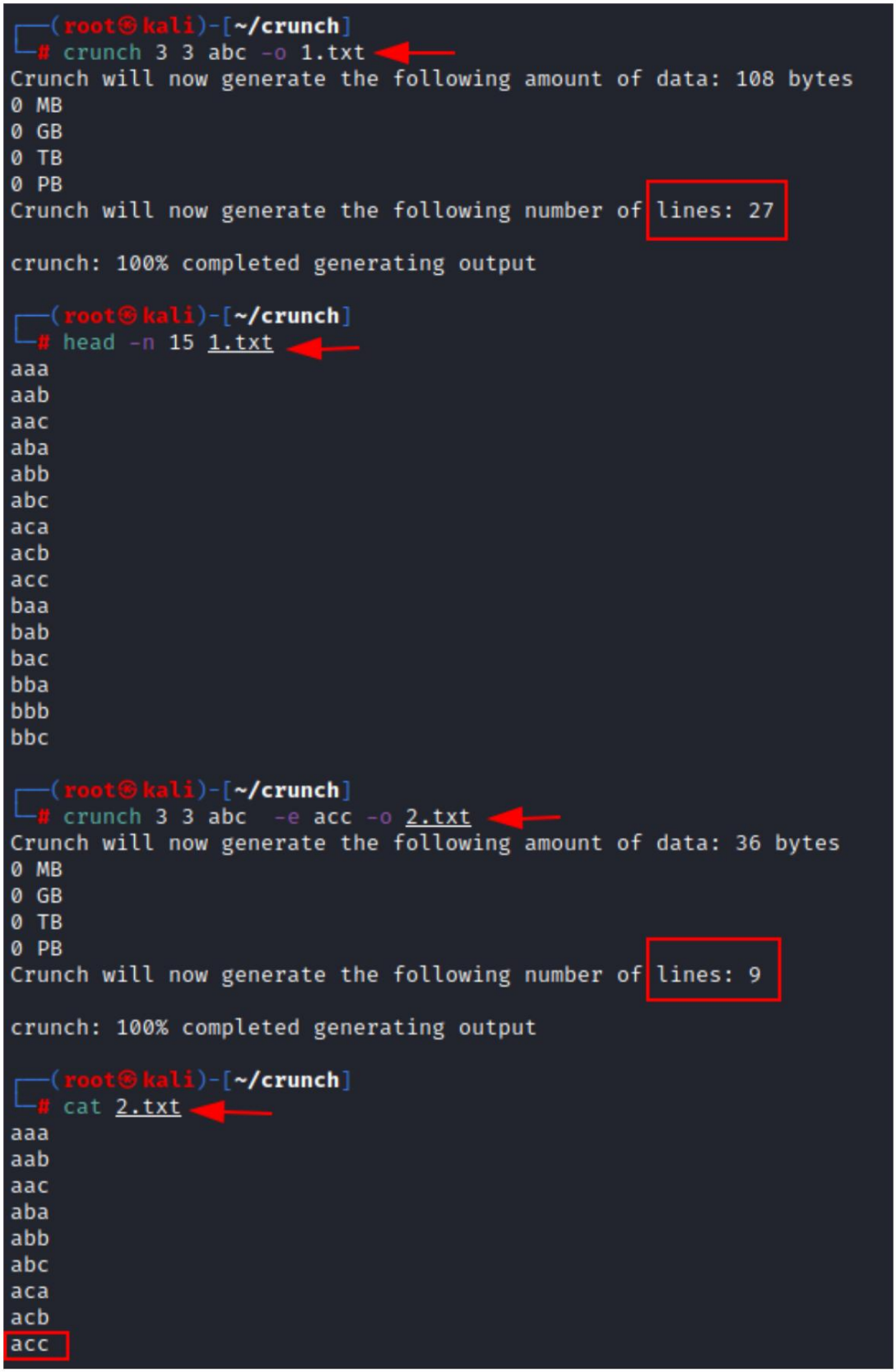


Poner paradas tempranas en las listas de palabras

Según los requisitos del usuario, también puede existir la posibilidad de que un usuario quiera acortar una lista a una determinada combinación. Por ejemplo, si un usuario desea crear una lista de palabras de 3 caracteres por palabra

usando “a, byc” como caracteres pero quiere cortarlo tan pronto como la lista de palabras genere la combinación “acc”, puede hacerlo así:

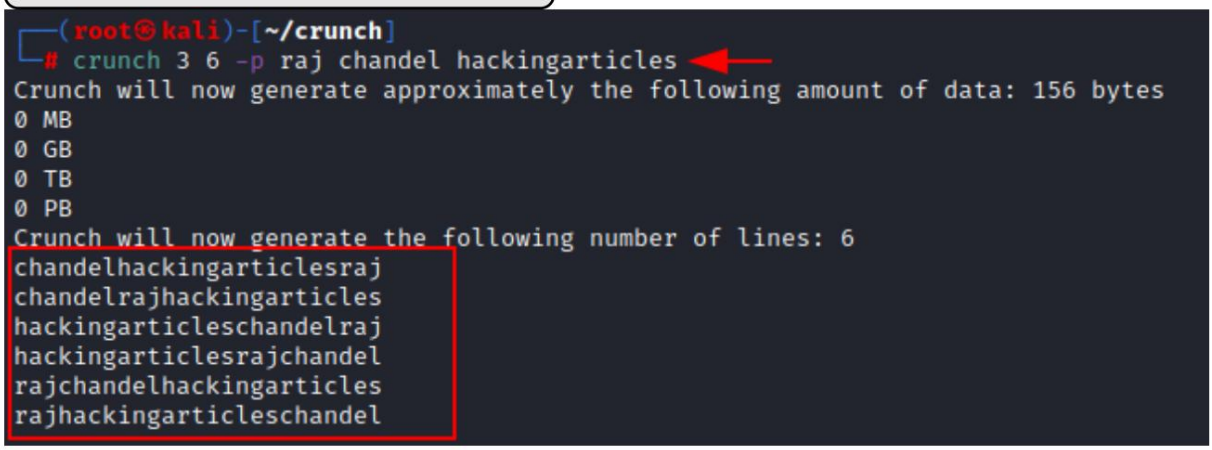
```
crujido 3 3 abc -o 1.txt
crujido 3 3 abc -e acc -o 2.txt
```



Permutaciones de palabras

En matemáticas, las permutaciones representan combinaciones no repetidas de ciertos eventos. Entonces, para generar listas de palabras que no se repiten mediante permutaciones podemos usar el filtro "-p". Aquí, proporcionamos 3 palabras como entrada, ninguna de las cuales se repetirá incluso si el tamaño máximo de la lista de palabras es 6.

```
crunch 3 6 -p raj chandel hackingartículos
```



Permutaciones de listas de

palabras Al igual que las palabras se pueden permutar, las listas de palabras se pueden permutar. Usando la opción "-q", Crunch puede tomar información de una lista de palabras y hacer permutaciones sobre lo que se lee en el archivo. Por ejemplo, si la lista de archivos es:

A

B

C

Luego, crunch -q list.txt generaría:

A B C

ACB

bachillerato

ACB

TAXI

ACB

De manera similar, podemos hacer permutaciones en una lista de palabras de 3 caracteres por palabra de la siguiente manera:

```
crujido 3 3 abc -e acc -o 2.txt
```

crujido 3 3 abc -q 2.txt -o 3.txt

```
┌──(root㉿kali)-[~/crunch]
└─# crunch 3 3 abc -e acc -o 2.txt
Crunch will now generate the following amount of data: 36 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 9

crunch: 100% completed generating output

┌──(root㉿kali)-[~/crunch]
└─# cat 2.txt
aaa
aab
aac
aba
abb
abc
aca
acb
acc

┌──(root㉿kali)-[~/crunch]
└─# crunch 3 3 abc -q 2.txt -o 3.txt
Crunch will now generate approximately the following amount of data: 1016064
9 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 362880
^C
crunch: 100% completed generating output

┌──(root㉿kali)-[~/crunch]
└─# head -n 15  3.txt
aaaaabaacabaabbabcacaacbacc
aaaaabaacabaabbabcacaaccacb
aaaaabaacabaabbabcacbacaacc
aaaaabaacabaabbabcacbaccaca
aaaaabaacabaabbabcaccacaacb
aaaaabaacabaabbabcaccacbaca
aaaaabaacabaabbacaabcacbacc
aaaaabaacabaabbacaabcaccacb
aaaaabaacabaabbacaacbabcacc
aaaaabaacabaabbacaacbaccabc
aaaaabaacabaabbacaaccabcacb
aaaaabaacabaabbacaaccacbabc
aaaaabaacabaabbacbabcacaacc
aaaaabaacabaabbacbabcaccaca
aaaaabaacabaabbacbacaabcacc
```

Dividir la lista de palabras según el recuento de palabras

Una lista de palabras se puede acortar usando la opción "-c". Aquí se ha generado un archivo con 94 palabras.

Ahora, para dividir eso en varios archivos, cada uno con un máximo de 60 palabras, se puede hacer así.

Tenga en cuenta que esto solo funciona con "-o START", que nombrará automáticamente los archivos en el formato:

Carácter inicial - Carácter final.txt

Aquí, el inicio y el final son a,7 y para el siguiente archivo, 8 y /(espacio)

```
crunch 1 1 -f charset.lst mixalfa-numérico-todo-espacio -o archivo.txt
crunch 1 1 -f charset.lst mixalfanumérico-todo-espacio -o INICIO -c 60
```

```
┌──(root㉿kali)-[/usr/share/crunch]
└─# crunch 1  1 -f charset.lst mixalpha-numeric-all-space -o file.txt
Crunch will now generate the following amount of data: 190 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 95

crunch: 100% completed generating output

┌──(root㉿kali)-[/usr/share/crunch]
└─# crunch 1  1 -f charset.lst mixalpha-numeric-all-space -o START -c 60
Crunch will now generate the following amount of data: 120 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 60

crunch: 100% completed generating output

crunch: 158% completed generating output

┌──(root㉿kali)-[/usr/share/crunch]
└─# ls
'8- .txt'   a-7.txt   charset.lst   file.txt

┌──(root㉿kali)-[/usr/share/crunch]
└─# wc file.txt
 95  94 190 file.txt

┌──(root㉿kali)-[/usr/share/crunch]
└─# wc a-7.txt
 60  60 120 a-7.txt

┌──(root㉿kali)-[/usr/share/crunch]
└─# wc 8-\ .txt
35 34 70 8- .txt

┌──(root㉿kali)-[/usr/share/crunch]
└─#
```

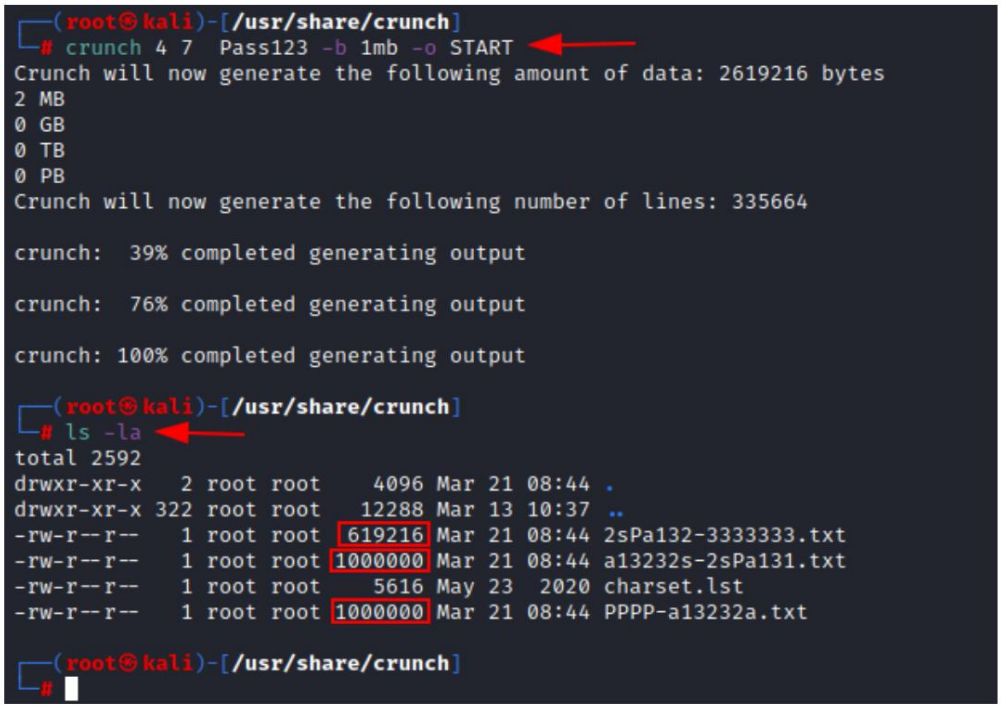
Dividir la lista de palabras según el tamaño

Para acortar un archivo según el tamaño, podemos usar el filtro "-b". Por ejemplo, para dividir una lista de palabras en varios archivos, cada uno de un máximo de 1 MB, podemos hacer:

```
crujido 4 7 Pass123 -b 1mb -o INICIO
```

Recuerde, -o START es obligatorio ya que dividirá automáticamente el archivo en el formato:

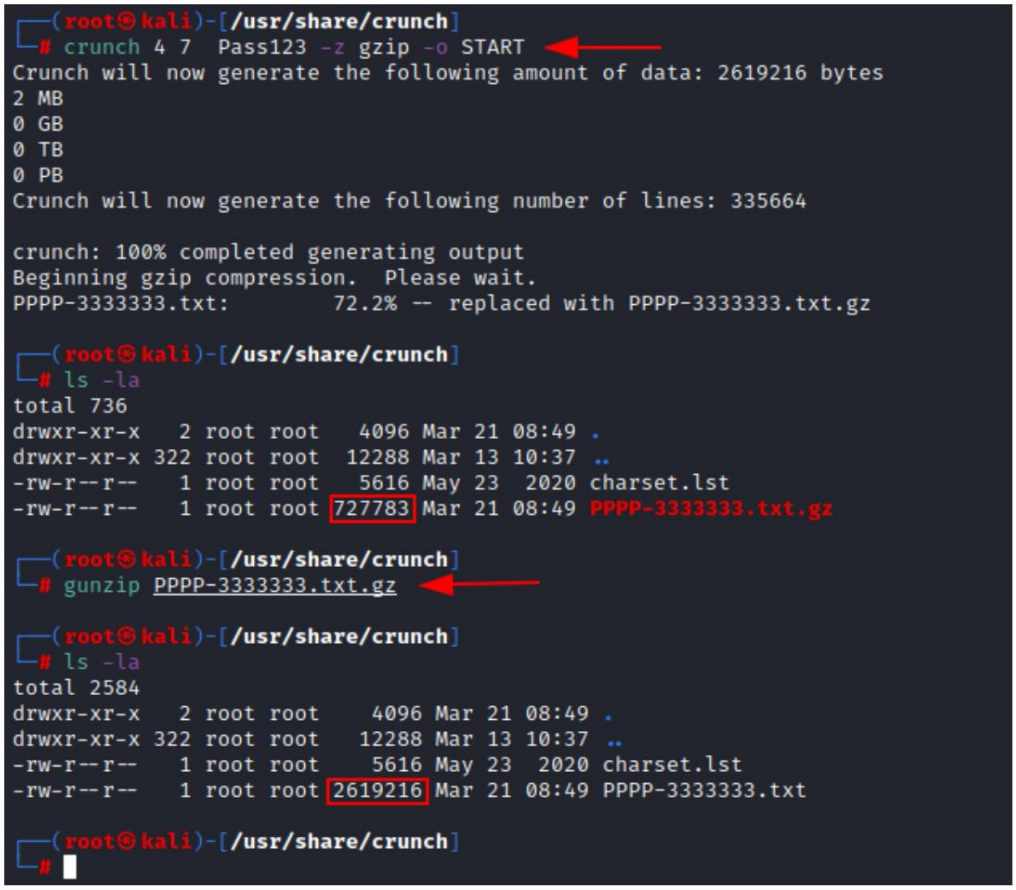
Carácter inicial - Carácter final.txt



Comprimir lista de palabras

A menudo, las listas de palabras tienen un tamaño demasiado grande cuando están en formato de texto y se puede usar gzip para comprimirlas a más del 60-70%. Por ejemplo, para comprimir un archivo con un máximo de 7 conjuntos de caracteres alfanuméricos mixtos y nombre automático usando INICIO, podemos hacer esto:

```
crujido 4 7 Pass123 -z gzip -o INICIO
gunzip PPPP-3333333.txt.gz
```



Conclusión

El artículo debe considerarse como una referencia inmediata para la generación rápida y sucia de listas de palabras mediante crunch. Crunch es una herramienta potente y muy rápida escrita en C que está disponible de forma predeterminada en Kali Linux y puede usarse en exámenes competitivos de certificación de seguridad. Espero que te haya gustado el artículo y gracias por leer.

iGNITE Technologies

ÚNETE A NUESTRO PROGRAMAS DE ENTRENAMIENTO

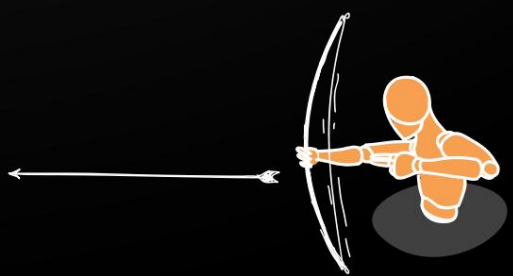
PRINCIPIANTE

- Hackeo ético
- Pentest de red
- Recompensa de errores
- Pentest inalámbrico
- Seguridad de la red Esenciales

AVANZADO

- Eructar Suite Pro
- Web Servicios-API
- Pro VAPT de infraestructura
- Computadora forense
- Pentest de Android
- Avanzado metasploit
- CTF

EXPERTO



- Operación del equipo rojo
  - APT's - Tácticas de ataque MITRE
  - Ataque de directorio activo
  - Evaluación de seguridad de MSSQL
- Escalada de privilegios
  - ventanas
  - linux

www.ignitetechnologies.in