



# Sed & Awk & Grep - The Tree Swordsmen (English version)

---

A book from the Documentation Team

Version : 2024/03/20

*Rocky Documentation Team*

*Copyright © 2023 The Rocky Enterprise Software Foundation*

## Table of contents

---

1. Licence	3
2. Overview	4
3. Regular expressions and wildcards	5
3.1 Wildcards in GNU/Linux	5
3.2 Regular expressions in GNU/Linux	6
3.2.1 BRE	7
3.2.2 ERE	8
3.2.3 POSIX character	10
3.2.4 Introducing regular expressions	10
4. grep command	11
4.1 Examples of usage	12
5. sed command	18
5.1 Examples of usage	20
6. awk command	37
6.1 Instructions for using awk	39
6.1.1 printf commands	40
6.2 Basic usage example	42
6.3 Built-in variable	49
6.4 Operator	56
6.5 Flow control	61
6.6 Array	64
6.7 Built-in function	72
6.8 I/O statement	80
6.9 Concluding remarks	85

## 1. Licence

---

RockyLinux offers Linux courseware for trainers or people wishing to learn how to administer a Linux system on their own.

RockyLinux materials are published under Creative Commons-BY-SA. This means you are free to share and transform the material, while respecting the author's rights.

**BY : Attribution.** You must cite the name of the original author.

**SA : Share Alike.**

- Creative Commons-BY-SA licence : <https://creativecommons.org/licenses/by-sa/4.0/>

The documents and their sources are freely downloadable from:

- <https://docs.rockylinux.org>
- <https://github.com/rocky-linux/documentation>

Our media sources are hosted at [github.com](https://github.com). You'll find the source code repository where the version of this document was created.

From these sources, you can generate your own personalized training material using [mkdocs](#). You will find instructions for generating your document [here](#).

How can I contribute to the documentation project?

You'll find all the information you need to join us on our [git project home page](#).

We wish you all a pleasant reading and hope you enjoy the content.

## 2. Overview

---

The GNU/Linux operating system follows the philosophy of "everything is a file". A side consequence of this philosophy is that system administrators often have to interact with files, file names, and file contents.

Regarding processing file content, the three tools `grep`, `sed`, and `awk` are potent and frequently used, so people call them the "Three Swordsmen".

## 3. Regular expressions and wildcards

---

In the GNU/Linux operating system, regular expressions and wildcards often have the same symbol (or style), so people often confuse them.

What is the difference between regular expressions and wildcards?

Similarities:

- They have the same symbol but represent entirely different meanings.

Differences:

- Regular expressions match file content; Wildcards are typically used to match file or directory names.
- Regular expressions are typically used on commands such as `grep`, `sed`, `awk`, and so on.
- Wildcards are typically used with commands such as `cp`, `find`, `mv`, `touch`, `ls`, and so on.

### 3.1 Wildcards in GNU/Linux

---

GNU/Linux operating systems support these wildcards:

wildcards style	role
?	Matches one character of a file or directory name.
*	Matches 0 or more arbitrary characters of a file or directory name.
[ ]	Matches any single character in parentheses. For example, [one] which means to match o or n or e.
[-]	Matches any single character within the given range in parentheses. For example, [0-9] matches any single number from 0 to 9.
[^]	"logical non" matching of a single character. For example, [^a-zA-Z] represents matching a single non-letter character.
{,}	Non continuous matching of multiple single characters. Separated by commas.
{..}	Same as [-]. For example {0..9} and {a..z}

Different commands have different support for wildcard styles:

- `find` : Supports \*, ?, [ ], [-], [^]
- `ls` : All supported
- `mkdir` : Supports {}, and {..}
- `touch` : Supports {}, and {..}
- `mv` : All supported
- `cp` : All supported

For example:

```
Shell > mkdir -p /root/dir{1..3}
Shell > cd /root/dir1/
Shell > touch file{1,5,9}
Shell > cd
Shell > mv /root/dir1/file[1-9] /root/dir2/
Shell > cp /root/dir2/file{1..9} /root/dir3/
Shell > find / -iname "dir[1-9]" -a -type d
```

## 3.2 Regular expressions in GNU/Linux

---

Two major schools of regular expressions exist due to historical development:

- POSIX:
- BRE basic regular express
- ERE extend regular express
- POSIX character class
- PCRE (Perl Compatible Regular Expressions): The most common among various programming languages.

	BRE	ERE	POSIX character class	PCRE
<code>grep</code>	√	√ (Requires -E option)	√	√ (Requires -P option)
<code>sed</code>	√	√ (Requires -r option)	√	✗
<code>awk</code>	√	√	√	✗

For more on regular expressions, visit [this website](#) for more helpful information.

### 3.2.1 BRE

BRE (Basic Regular Expression) is the oldest type of regular expression, introduced by the `grep` command in UNIX systems and the `ed` text editor.

metacharacter	description	bash example
*	Matches the number of occurrences of the previous character, which can be 0 or any number of times.	
.	Matches any single character except for line breaks.	
^	Matches line beginning. For example - <code>^h</code> will match lines starting with h.	
\$	Matches End of Line. For example - <code>h\$</code> will match lines ending in h.	
[]	Matches any single character specified in parentheses. For example - <code>[who]</code> will match w or h or o; <code>[0-9]</code> will match one digit; <code>[0-9][a-z]</code> will match characters composed of one digit and a single lowercase letter.	
[^]	Matches any single character except for the characters in parentheses. For example - <code>[^0-9]</code> will match any single non-numeric character. <code>[^a-z]</code> will match any single character that is not a lowercase letter.	
\	Escape character, used to cancel the meaning represented by some special symbols.	<code>echo -e "1.2\n122" \  grep -E '\1\.\2'</code> <b>1.2</b>
\{n\}	Matches the number of occurrences of the previous single character, n represents the number of matches.	<code>echo -e "1994\n2000\n2222" \  grep "[24]\{4\}"</code> <b>2222</b>
\{n,\}	Matches the previous single character at least n times.	<code>echo -e "1994\n2000\n2222" \  grep "[29]\{2,\}"</code> <b>1994</b> <b>2222</b>
\{n,m\}	Matches the previous single character at least n times and at most m times.	<code>echo -e "abcd\n20\n300\n4444" \  grep "[0-9]\{2,4\}"</code> <b>20</b> <b>300</b> <b>4444</b>

## 3.2.2 ERE

metacharacter	description	bash example
+	Matches the number of occurrences of the previous single character, which can be 1 or more times.	echo -e "abcd\nab\nabb\ncdd" \  grep -E "ab+" <b>abcd</b> <b>ab</b> <b>abb</b>
?	Matches the number of occurrences of the previous single character, which can be 0 or 1.	echo -e "ac\nabc\nadd" \  grep -E 'a?c' <b>ac</b> <b>abc</b>
\<	Boundary character, matching the beginning of a string.	echo -e "1\n12\n123\n1234" \  grep -E "\<123" <b>123</b> <b>1234</b>
\>	Boundary character, matching the end of the string.	echo -e "export\nimport\nout" \  grep -E "port\>" <b>export</b> <b>import</b>
()	Combinatorial matching, that is, the string in parentheses as a combination, and then match.	echo -e "123abc\nabc123\na1b2c3" \  grep -E "([a-z][0-9])+" <b>a<b>1</b>b<b>2</b>c<b>3</b></b>
	The pipeline symbol represents the meaning of "or".	echo -e "port\nimport\nexport\none123" \  grep -E "port\>\ 123" <b>port</b> <b>import</b> <b>export</b> <b>one123</b>

ERE also supports characters with special meanings:

special characters	description
\w	Equivalent to <b>[a-zA-Z0-9]</b>
\W	Equivalent to <b>[^a-zA-Z0-9]</b>
\d	Equivalent to <b>[0-9]</b>
\D	Equivalent to <b>[^0-9]</b>
\b	Equivalent to \< or \>
\B	Matches non-boundary character.
\s	Matches any whitespace character. Equivalent to <b>[ \f\n\r\t\v]</b>
\S	Equivalent to <b>[^ \f\n\r\t\v]</b>

blank character	description
\f	Matches a single feed character. Equivalent to <b>\x0c</b> and <b>\cL</b>
\n	Matches individual line breaks. Equivalent to <b>\x0a</b> and <b>\cJ</b>
\r	Matches a single carriage return. Equivalent to <b>\x0d</b> and <b>\cM</b>
\t	Matches a single tab. Equivalent to <b>\x09</b> and <b>\cI</b>
\v	Matches a single vertical tab. Equivalent to <b>\x0b</b> and <b>\cK</b>

### 3.2.3 POSIX character

Sometimes, you may see "POSIX character"(also known as "POSIX character class"). Please note that the author rarely uses the "POSIX character class", but has included this section to enhance basic understanding.

POSIX character	equivalent to
[:alnum:]	[a-zA-Z0-9]
[:alpha:]	[a-zA-Z]
[:lower:]	[a-z]
[:upper:]	[A-Z]
[:digit:]	[0-9]
[:space:]	[ \f\n\r\t\v]
[:graph:]	[^\f\n\r\t\v]
[:blank:]	[ \t]
[:cntrl:]	[\x00-\x1F\x7F]
[:print:]	[\x20-\x7E]
[:punct:]	[]!#\$%&'^*+,./;<=>?@^_`{ }~-]
[:xdigit:]	[A-Fa-f0-9]

### 3.2.4 Introducing regular expressions

Many websites exist for practicing regular expression skills online, such as:

- [regex101](#)
- [oschina](#)
- [regexpr](#)
- [regelearn](#)
- [coding](#)

## 4. grep command

---

The `grep` command filters the content of single or multiple files. Some variants of this command tool exist, such as `egrep` (`grep -E`) and `fgrep` (`grep -f`). For information not covered, see [the `grep` manual](#).

The usage of the `grep` command is:

```
grep [OPTIONS] PATTERN [FILE...]
grep [OPTIONS] -e PATTERN ... [FILE...]
grep [OPTIONS] -f FILE ... [FILE...]
```

The options are mainly divided into four parts:

- match control
- output control
- content line control
- directory or file control

### match control

options	description
-E --extended-regexp	Enable ERE
-P --perl-regexp	Enable PCRE
-G --basic-regexp	Enable BRE by default
-e --regexp=PATTERN	Pattern matching, multiple -e options can be specified.
-i	Ignore case
-w	Accurately match the entire word
-f FILE	Obtain patterns from FILE, one per line
-x	Pattern matching entire line
-v	Select content lines that do not match

## output control:

<b>options</b>	<b>description</b>
-m NUM	Output the first few matching results
-n	Print line numbers on output
-H	When matching the file content of multiple files, display the file name at the beginning of the line. This is the default setting
-h	When matching the file content of multiple files, the file name is not displayed at the beginning of the line
-o	Print only matching content without outputting the entire line
-q	Not outputting normal information
-s	Do not output error messages
-r	Recursive matching for directories
-c	Prints the number of lines matched by each file based on the content of the user

## content line control:

<b>options</b>	<b>description</b>
-B NUM	Print NUM lines of leading context before matching lines
-A NUM	Print NUM lines of trailing context after matching lines
-C NUM	Print NUM lines of output context

## directory or file control:

<b>options</b>	<b>description</b>
--include=FILE_PATTERN	Searches only files that match FILE_PATTERN. Wildcard characters for file names support *, ?, [], [^], [-], {..}, {},
--exclude=FILE_PATTERN	Skip files and directories matching FILE_PATTERN. Wildcard characters for file names support *, ?, [], [^], [-], {..}, {},
--exclude-dir=PATTERN	Excludes the specified directory name. Directory name support *, ?, [], [^], [-], {..}, {},
--exclude-from=FILE	Excludes the specified directory from the file content.

## 4.1 Examples of usage

---

### 1. -f option and -o option

```
Shell > cat /root/a
abcdef
123456
338922549
```

```

24680
hello world

Shell > cat /root/b
12345
test
world
aaaaa

# Treat each line of file b as a matching pattern and output the lines that match
file a.
Shell > grep -f /root/b /root/a
123456
hello world

Shell > grep -f /root/b /root/a -o
12345
world

```

## 2. Multiple pattern matching (using the -e option)

```

Shell > echo -e "a\nab\nbc\nbcde" | grep -e 'a' -e 'cd'
a
ab
bcde

```

or:

```

Shell > echo -e "a\nab\nbc\nbcde" | grep -E "a|cd"
a
ab
bcde

```

## 3. Remove blank lines and comment lines from the configuration file

```

Shell > grep -v -E "^$|^#" /etc/chrony.conf
server ntp1.tencent.com iburst
server ntp2.tencent.com iburst
server ntp3.tencent.com iburst
server ntp4.tencent.com iburst
driftfile /var/lib/chrony/drift
makestep 1.0 3
rtcsync
keyfile /etc/chrony.keys

```

```
leapsectz right/UTC
logdir /var/log/chrony
```

#### 4. Print the top 5 results that all match

```
Shell > seq 1 20 | grep -m 5 -E "[0-9]{2}"
10
11
12
13
14
```

or:

```
Shell > seq 1 20 | grep -m 5 "[0-9]\{2\}"
10
11
12
13
14
```

#### 5. -B option and -A option

```
Shell > seq 1 20 | grep -B 2 -A 3 -m 5 -E "[0-9]{2}"
8
9
10
11
12
13
14
15
16
17
```

#### 6. -C option

```
Shell > seq 1 20 | grep -C 3 -m 5 -E "[0-9]{2}"
7
8
9
10
11
12
```

13  
14  
15  
16  
17

## 7. -c option

```
Shell > cat /etc/ssh/sshd_config | grep -n -i -E "port"
13:# If you want to change the port on a SELinux system, you have to tell
15:# semanage port -a -t ssh_port_t -p tcp #PORTNUMBER
17:#Port 22
99:# WARNING: 'UsePAM no' is not supported in RHEL and may cause several
105:#GatewayPorts no

Shell > cat /etc/ssh/sshd_config | grep -E -i "port" -c
5
```

## 8. -v option

```
Shell > cat /etc/ssh/sshd_config | grep -i -v -E "port" -c
140
```

## 9. Filter files in a directory that have lines that match the string (Exclude files in subdirectories)

```
Shell > grep -i -E "port" /etc/n*.conf -n
/etc/named.conf:11:      listen-on port 53 { 127.0.0.1; };
/etc/named.conf:12:      listen-on-v6 port 53 { ::1; };
/etc/nsswitch.conf:32:# winbind                         Use Samba winbind support
/etc/nsswitch.conf:33:# wins                            Use Samba wins support
```

## 10. Filter files in a directory that have lines that match the string (include or exclude files or directories in subdirectories)

Include syntax for multiple files:

```
Shell > grep -n -i -r -E "port" /etc/ --include={0..20}/*
/etc/grub.d/20_ppc_terminfo:26:export TEXTDOMAIN=grub
/etc/grub.d/20_ppc_terminfo:27:export TEXTDOMAINDIR=/usr/share/locale
/etc/grub.d/20_linux_xen:26:export TEXTDOMAIN=grub
/etc/grub.d/20_linux_xen:27:export TEXTDOMAINDIR="${datarootdir}/locale"
/etc/grub.d/20_linux_xen:46:# Default to disabling partition uuid support to
maintain compatibility with
```

```
/etc/grub.d/10_linux:26:export TEXTDOMAIN=grub
/etc/grub.d/10_linux:27:export TEXTDOMAINDIR="${datarootdir}/locale"
/etc/grub.d/10_linux:47:# Default to disabling partition uuid support to
maintain compatibility with

Shell > grep -n -i -r -E "port" /etc/ --include={{0..20}_*,sshd_config} -c
/etc/ssh/sshd_config:5
/etc/grub.d/20_ppc_terminfo:2
/etc/grub.d/10_reset_boot_success:0
/etc/grub.d/12_menu_auto_hide:0
/etc/grub.d/20_linux_xen:3
/etc/grub.d/10_linux:3
```

If you need to exclude a single directory, use the following syntax:

```
Shell > grep -n -i -r -E "port" /etc/ --exclude-dir=selin[u]x
```

If you need to exclude multiple directories, use the following syntax:

```
Shell > grep -n -i -r -E "port" /etc/ --exclude-dir={selin[u]x,"profile.d",
{a..z}ki,au[a-z]it}
```

If you need to exclude a single file, use the following syntax:

```
Shell > grep -n -i -r -E "port" /etc/ --exclude=sshd_config
```

If you need to exclude multiple files, use the following syntax:

```
Shell > grep -n -i -r -E "port" /etc/ --exclude={ssh[a-
z]_config,*.conf,services}
```

If you need to exclude multiple files and directories at the same time, use the following syntax:

```
Shell > grep -n -i -r -E "port" /etc/ --exclude-dir={selin[u]x,"profile.d",
{a..z}ki,au[a-z]it} --exclude={ssh[a-z]_config,*.conf,services,[0-9][0-9]*}
```

## 11. Count all IPv4 addresses of the current machine

```
Shell > ip a | grep -o -E "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}" |  
grep -v -E "127|255"  
192.168.100.3
```

## 5. `sed` command

---

`sed` : Stream Editor

**Working principle:** The `sed` command will read the currently processed row and place it in the "pattern space" for processing. After processing, the result will be output and the "pattern space" will be cleared. Next, read the next line and place it in the "pattern space" for processing, and so on, until the last line. Some documents also mention a term called "hold space" (also known as "temporary-storage space"), which can temporarily store some processed data and output it through "pattern space".

**"pattern space" and "hold space":** An area of memory where data is processed and stored.

For information not covered, review [the `sed` manual](#).

The usage of the command is:

```
sed [OPTION]... {script-only-if-no-other-script} [input-file]...
```

<b>options</b>	<b>description</b>
-n	Output text lines that will only be processed by the <code>sed</code> command to the screen
-e	Apply multiple <code>sed</code> operation commands to the input text line data
-f	Call and execute <code>sed</code> script command file
-i	Modify the original file
-r	Regular expression

<b>Operation command (sometimes called operation instruction)</b>	<b>description</b>
s/regexp/replacement/	Replacement string
p	Print the current "pattern space". Often used with the -n option, for example: <code>cat -n /etc/services \  sed -n '3,5p'</code>
d	Delete "pattern space". Start next cycle
D	Delete the first line of the "pattern space" and start next cycle
=	Print Line Number
a \text	Add one or more lines of content after the matching line. When adding multiple lines, all lines except the last line need to use "\\" to indicate that the content has not ended
i \text	Add one or more lines of content before the matching line. When adding multiple lines, all lines except the last line need to use "\\" to indicate that the content is not ended
c \text	Replace matching lines with new text
q	Immediately exit the <code>sed</code> script
r	Append text read from file
: label	Label for b and t commands
b label	Branch to label; if label is omitted, branch to end of script
t label	If "s///" is a successful replacement, then jump to the label
h H	Copy/append "pattern space" to "hold space"
g G	Copy/append "hold space" to "pattern space"
x	Exchange the contents of the hold and pattern spaces
l	List out the current line in a "visually unambiguous" form
n N	Read/append the next line of input into the "pattern space"
w FILENAME	Write the current pattern space to FILENAME
!	negation
&	Referencing a string that already matches

Addresses	description
first~step	Use "first" to specify the first line, and 'step' to specify the step size. For example, outputting odd lines of text with <code>sed -n "1~2p" /etc/services</code>
\$	Match the last line of text
/regexp/	Using regular expressions to match text lines
number	Specify line number
addr1,addr2	Use line number positioning to match all lines from "addr1" to "addr2"
addr1,+N	Use line number positioning to match addr1 and the N lines following addr1

## 5.1 Examples of usage

---

### 1. Match and print ( p )

- Print a line that begins with a netbios string

```
Shell > cat /etc/services | sed -n '/^netbios/p'
netbios-ns      137/tcp                      # NETBIOS Name Service
netbios-ns      137/udp
netbios-dgm     138/tcp                      # NETBIOS Datagram Service
netbios-dgm     138/udp
netbios-ssn     139/tcp                      # NETBIOS session service
netbios-ssn     139/udp
```

 Tip

As we all know, double and single quotation marks in a shell play different roles. The \$, `, and \ in double quotes have a special meaning. The recommendation is to use single quotes more often when using the `sed` command.

- Print the text from lines 23 to 26

```
Shell > cat -n /etc/services | sed -n '23,26p'
23  tcpmux          1/tcp                      # TCP port service
multiplexer
24  tcpmux          1/udp                      # TCP port service
multiplexer
25  rje             5/tcp                      # Remote Job Entry
26  rje             5/udp                      # Remote Job Entry
```

- Print odd lines

```
Shell > cat -n /etc/services | sed -n '1~2p'
1  # /etc/services:
```

```

3  #
5  # IANA services version: last updated 2016-07-08
7  # Note that it is presently the policy of IANA to assign a single well-known
9  # even if the protocol doesn't support UDP operations.
11 # are included, only the more common ones.
13 # The latest IANA port assignments can be gotten from
15 # The Well Known Ports are those from 0 through 1023.
17 # The Dynamic and/or Private Ports are those from 49152 through 65535
19 # Each line describes one service, and is of the form:
...

```

- Print line 10 to the last line

```

Shell > cat -n /etc/services | sed -n '10,$p'
10 # Updated from RFC 1700, ``Assigned Numbers'' (October 1994). Not all ports
11 # are included, only the more common ones.
12 #
13 # The latest IANA port assignments can be gotten from
14 # http://www.iana.org/assignments/port-numbers
15 # The Well Known Ports are those from 0 through 1023.
16 # The Registered Ports are those from 1024 through 49151
17 # The Dynamic and/or Private Ports are those from 49152 through 65535
...

```

- Lines 10 to the last do not print

```

Shell > cat -n /etc/services | sed -n '10,$!p'
1 # /etc/services:
2 # $Id: services,v 1.49 2017/08/18 12:43:23 ovasik Exp $
3 #
4 # Network services, Internet style
5 # IANA services version: last updated 2016-07-08
6 #
7 # Note that it is presently the policy of IANA to assign a single well-known
8 # port number for both TCP and UDP; hence, most entries here have two entries
9 # even if the protocol doesn't support UDP operations.

```

- Print the line number and content of the matched string

```

Shell > sed -n -e '/netbios=/ -e '/netbios/p' /etc/services
123
netbios-ns      137/tcp                                # NETBIOS Name Service
124
netbios-ns      137/udp
125

```

```

netbios-dgm      138/tcp          # NETBIOS Datagram Service
126
netbios-dgm      138/udp
127
netbios-ssn      139/tcp          # NETBIOS session service
128
netbios-ssn      139/udp

```

- Match string range and print it

Use commas to separate string ranges

```

Shell > cat /etc/services | sed -n '/^netbios/,/^imap/p'
netbios-ns      137/tcp          # NETBIOS Name Service
netbios-ns      137/udp
netbios-dgm      138/tcp          # NETBIOS Datagram Service
netbios-dgm      138/udp
netbios-ssn      139/tcp          # NETBIOS session service
netbios-ssn      139/udp
imap            143/tcp          imap2          # Interim Mail Access Proto v2

```

#### Info

**Start of range:** Match the line where the string is located, only matching the first string that appears. **End of range:** Match the line where the string is located, only matching the first string that appears.

```

Shell > grep -n ^netbios /etc/services
123:netbios-ns    137/tcp          # NETBIOS Name Service
124:netbios-ns    137/udp
125:netbios-dgm   138/tcp          # NETBIOS Datagram Service
126:netbios-dgm   138/udp
127:netbios-ssn   139/tcp          # NETBIOS session service
128:netbios-ssn   139/udp

Shell > grep -n ^imap /etc/services
129:imap          143/tcp          imap2          # Interim Mail Access Proto
v2
130:imap          143/udp          imap2
168:imap3         220/tcp          # Interactive Mail Access
169:imap3         220/udp          # Protocol v3
260:imaps         993/tcp          # IMAP over SSL
261:imaps         993/udp          # IMAP over SSL

```

In other words, the content printed above is lines 123 to 129

- Print the line where the string is located and until the last line

```
Shell > cat /etc/services | sed -n '/^netbios/, $p'
```

- Using variables in bash scripts

```
Shell > vim test1.sh
#!/bin/bash
a=10

sed -n "'${a}',$!p' /etc/services
# or
sed -n "${a},\$!p" /etc/services
```

- Regular expression

Matches only "Three Digits" + "/udp".

```
Shell > cat /etc/services | sed -r -n '/[^0-9]([1-9]{3})\/udp/p'
sunrpc      111/udp        portmapper rpcbind      # RPC 4.0 portmapper UDP
auth        113/udp        authentication tap ident
sftp         115/udp
uucp-path   117/udp
nntp         119/udp        readnews untp      # USENET News Transfer Protocol
ntp          123/udp        ntp                  # Network Time Protocol
netbios-ns   137/udp
netbios-dgm  138/udp
netbios-ssn  139/udp
...
```

## 2. Match and delete (`d`)

It is similar to printing, except that the operation command is replaced with `d` and the `-n` option is not required.

- Delete all lines that match the udp string, and delete all comment lines, and delete all Blank line

```
Shell > sed -e '/udp/d' -e '/#/d' -e '/$/d' /etc/services
tcpmux      1/tcp          # TCP port service multiplexer
rje         5/tcp          # Remote Job Entry
echo        7/tcp
discard    9/tcp          sink null
```

```

systat      11/tcp      users
daytime     13/tcp
qotd        17/tcp      quote
chargen     19/tcp      ttystt source
ftp-data    20/tcp
ftp         21/tcp
ssh          22/tcp      # The Secure Shell (SSH) Protocol
telnet      23/tcp
...

```

- Delete consecutive lines of text

```

Shell > cat -n /etc/services | sed '10,$d'
1  # /etc/services:
2  # $Id: services,v 1.49 2017/08/18 12:43:23 ovasik Exp $
3  #
4  # Network services, Internet style
5  # IANA services version: last updated 2016-07-08
6  #
7  # Note that it is presently the policy of IANA to assign a single well-known
8  # port number for both TCP and UDP; hence, most entries here have two entries
9  # even if the protocol doesn't support UDP operations.

```

- Regular expression

```

Shell > cat /etc/services | sed -r '/^(tcp)|(udp)|(^#)|(^$)/d'
http        80/sctp      # HyperText Transfer Protocol
bgp         179/sctp
https       443/sctp    # http protocol over TLS/SSL
h323hostcall 1720/sctp # H.323 Call Control
nfs          2049/sctp   nfsd shlp # Network File System
rtmp         1/ddp       # Routing Table Maintenance
Protocol
nbp          2/ddp       # Name Binding Protocol
echo         4/ddp       # AppleTalk Echo Protocol
zip          6/ddp       # Zone Information Protocol
discard      9/sctp     # Discard
discard      9/dccp     # Discard SC:DISC
...

```

### 3. Replace strings( s///g )

Syntax	Syntax description
sed 's/string/replace/g' FILENAME	<b>s:</b> All lines representing the content of the file. You can also specify the range of lines, for example: sed '20,200s/netbios/TMP/g' /etc/services <b>g (Global):</b> If there is no g, This means that when multiple matching strings appear on a single line, only the first matching string will be replaced. <b>/:</b> Delimiter style. You can also specify other styles, for example: sed '20,200s?netbios?TMP?g' /etc/services

#### 🔥 Tip

Example in the bash script:

```
Shell > vim /root/sedReplace.sh
#!/bin/bash
a="SELINUX=enforcing"
b="SELINUX=disabled"

sed -i 's/${a}'/'${b}'/g' /etc/selinux/config
# or
sed -i "s/${a}/${b}/g" /etc/selinux/config
```

- Replace and print

```
Shell > sed -n '44,45s/ssh/SSH/gp' /etc/services
SSH          22/tcp
SSH          22/udp
```

- Use the "&" symbol to reference a string

```
Shell > sed -n '44,45s/ssh/&-SSH/gp' /etc/services
ssh-SSH      22/tcp
ssh-SSH      22/udp
```

- Use a string to locate one or more lines and replace the specified string within the line range

```
Shell > grep ssh /etc/services -n
44:ssh          22/tcp                                # The Secure Shell (SSH)
Protocol
45:ssh          22/udp                                # The Secure Shell (SSH)
Protocol
551:x11-ssh-offset 6010/tcp                         # SSH X11 forwarding offset
593:ssh          22/sctp                             # SSH
1351:sshell      614/tcp                            # SSLshell
1352:sshell      614/udp                            # SSLshell
1607:netconf-ssh 830/tcp                           # NETCONF over SSH
1608:netconf-ssh 830/udp                           # NETCONF over SSH
```

```

7178:sdo-ssh      3897/tcp          # Simple Distributed Objects over
SSH
7179:sdo-ssh      3897/udp          # Simple Distributed Objects over
SSH
7791:netconf-ch-ssh 4334/tcp          # NETCONF Call Home (SSH)
8473:snmpssh      5161/tcp          # SNMP over SSH Transport Model
8474:snmpssh-trap 5162/tcp          # SNMP Notification over SSH
Transport Model
9126:tl1-ssh       6252/tcp          # TL1 over SSH
9127:tl1-ssh       6252/udp          # TL1 over SSH
10796:ssh-mgmt    17235/tcp          # SSH Tectia Manager
10797:ssh-mgmt    17235/udp          # SSH Tectia Manager

Shell > sed '/ssh/s/tcp/TCP/gp' -n /etc/services
ssh           22/TCP          # The Secure Shell (SSH) Protocol
x11-ssh-offset 6010/TCP         # SSH X11 forwarding offset
sshell        614/TCP          # SSLshell
netconf-ssh   830/TCP          # NETCONF over SSH
sdo-ssh       3897/TCP          # Simple Distributed Objects over SSH
netconf-ch-ssh 4334/TCP          # NETCONF Call Home (SSH)
snmpssh      5161/TCP          # SNMP over SSH Transport Model
snmpssh-trap 5162/TCP          # SNMP Notification over SSH Transport
Model
tl1-ssh       6252/TCP          # TL1 over SSH
ssh-mgmt     17235/TCP          # SSH Tectia Manager

```

- String replacement for consecutive lines

```
Shell > sed '10,30s/tcp/TCP/g' /etc/services
```

- Multiple matches and replacements

```
Shell > cat /etc/services | sed 's/netbios/test1/g ; s/^#//d ; s/dhcp/&t2/g'
```

- Group replacement of regular expressions

In regular expressions, a "(") is a grouping. \1 represents reference group 1, \2 represents reference group 2, and so on.

```

Shell > cat /etc/services
...
axio-disc      35100/tcp          # Axiomatic discovery protocol
axio-disc      35100/udp          # Axiomatic discovery protocol
pmwebapi      44323/tcp          # Performance Co-Pilot client HTTP API
cloudcheck-ping 45514/udp          # ASSIA CloudCheck WiFi Management

```

```

keepalive
cloudcheck      45514/tcp          # ASSIA CloudCheck WiFi Management System
spremotetablet 46998/tcp          # Capture handwritten signatures

Shell > cat /etc/services | sed -r 's/([0-9]*\tcp)/\1\tCONTENT1/g ; s/([0-9]*\udp)/\1\tADD2/g'
...
axio-disc      35100/tcp          CONTENT1          # Axiomatic discovery
protocol
axio-disc      35100/udp          ADD2            # Axiomatic discovery protocol
pmwebapi       44323/tcp          CONTENT1          # Performance Co-Pilot
client HTTP API
cloudcheck-ping 45514/udp         ADD2            # ASSIA CloudCheck WiFi
Management keepalive
cloudcheck      45514/tcp          CONTENT1          # ASSIA CloudCheck WiFi
Management System
spremotetablet 46998/tcp          CONTENT1          # Capture handwritten
signatures

```

\t: That is, a tab

- Replace all comment lines with blank space

```

Shell > cat /etc/services | sed -r 's/(^#.*)//g'
...
chargen      19/udp          ttytst source
ftp-data     20/tcp
ftp-data     20/udp

ftp          21/tcp
ftp          21/udp          fsp fspd
ssh          22/tcp          # The Secure Shell (SSH) Protocol
ssh          22/udp          # The Secure Shell (SSH) Protocol
...

```

- Replace one of the lowercase letters of a word with a capital letter

```

Shell > echo -e "hello,world\nPOSIX" | sed -r 's/(.*)w/\1W/g'
hello,World
POSIX

```

- String position swapping

```

Shell > cat /etc/services
...
```

```

cloudcheck-ping 45514/udp          # ASSIA CloudCheck WiFi Management
keepalive
cloudcheck      45514/tcp          # ASSIA CloudCheck WiFi Management System
spremotetablet 46998/tcp          # Capture handwritten signatures

```

We can divide this file into five parts:

```

cloudcheck-ping    45514        /     udp      # ASSIA CloudCheck WiFi
Management keepalive
↓           ↓           ↓           ↓           ↓
( .*)       (\<[0-9]+\>)   \/   (tcp|udp)   ( .*)
↓           ↓           ↓           ↓           ↓
\1         \2           \3           \4

```

```

Shell > cat /etc/services | sed -r 's/( .*)(\<[0-9]+\>)\/(tcp|udp)( .*)/\1\3\2\4/g'
...
edi_service    udp/34567      # dhanalakshmi.org EDI Service
axio-disc      tcp/35100      # Axiomatic discovery protocol
axio-disc      udp/35100      # Axiomatic discovery protocol
pmwebapi       tcp/44323      # Performance Co-Pilot client HTTP API
cloudcheck-ping  udp/45514    # ASSIA CloudCheck WiFi Management
keepalive
cloudcheck      tcp/45514      # ASSIA CloudCheck WiFi Management System
spremotetablet  tcp/46998      # Capture handwritten signatures

```

- Remove any blank characters

```

Shell > echo -e "abcd\t1 2 3 4\tWorld"
abcd      1 2 3 4 World
Shell > echo -e "abcd\t1 2 3 4\tWorld" | sed -r 's/(\s)*//g'
abcd1234World

```

#### 4. Execute multiple times using the -e option

The following example:

```

Shell > tail -n 10 /etc/services
aigairserver    21221/tcp      # Services for Air Server
ka-kdp          31016/udp      # Kollective Agent Kollective Delivery
ka-sddp          31016/tcp      # Kollective Agent Secure Distributed
Delivery
edi_service     34567/udp      # dhanalakshmi.org EDI Service
axio-disc       35100/tcp      # Axiomatic discovery protocol

```

```

axio-disc      35100/udp          # Axiomatic discovery protocol
pmwebapi       44323/tcp          # Performance Co-Pilot client HTTP API
cloudcheck-ping 45514/udp        # ASSIA CloudCheck WiFi Management
keepalive
cloudcheck     45514/tcp          # ASSIA CloudCheck WiFi Management System
spremotetablet 46998/tcp        # Capture handwritten signatures

Shell > tail -n 10 /etc/services | sed -e '1,3d' -e '/cloud/s/ping/PING/g'
# or
Shell > tail -n 10 /etc/services | sed '1,3d ; /cloud/s/ping/PING/g'
edi_service    34567/udp          # dhanalakshmi.org EDI Service
axio-disc      35100/tcp          # Axiomatic discovery protocol
axio-disc      35100/udp          # Axiomatic discovery protocol
pmwebapi       44323/tcp          # Performance Co-Pilot client HTTP API
cloudcheck-PING 45514/udp        # ASSIA CloudCheck WiFi Management
keepalive
cloudcheck     45514/tcp          # ASSIA CloudCheck WiFi Management System
spremotetablet 46998/tcp        # Capture handwritten signatures

```

## 5. Add content above or below a specific line (`i` and `a`)

- Add two lines of content above the specified line number

```

Shell > tail -n 10 /etc/services > /root/test.txt
Shell > cat /root/test.txt
aigairserver   21221/tcp          # Services for Air Server
ka-kdp         31016/udp          # Kollective Agent Kollective Delivery
ka-sddp         31016/tcp          # Kollective Agent Secure Distributed
Delivery
edi_service    34567/udp          # dhanalakshmi.org EDI Service
axio-disc      35100/tcp          # Axiomatic discovery protocol
axio-disc      35100/udp          # Axiomatic discovery protocol
pmwebapi       44323/tcp          # Performance Co-Pilot client HTTP API
cloudcheck-ping 45514/udp        # ASSIA CloudCheck WiFi Management
keepalive
cloudcheck     45514/tcp          # ASSIA CloudCheck WiFi Management System
spremotetablet 46998/tcp        # Capture handwritten signatures

Shell > cat /root/test.txt | sed '3i 123\
abc'
abc
aigairserver   21221/tcp          # Services for Air Server
ka-kdp         31016/udp          # Kollective Agent Kollective Delivery
123
abc
ka-sddp         31016/tcp          # Kollective Agent Secure Distributed
Delivery
edi_service    34567/udp          # dhanalakshmi.org EDI Service
axio-disc      35100/tcp          # Axiomatic discovery protocol

```

```

axio-disc      35100/udp          # Axiomatic discovery protocol
pmwebapi       44323/tcp          # Performance Co-Pilot client HTTP API
cloudcheck-ping 45514/udp        # ASSIA CloudCheck WiFi Management
keepalive
cloudcheck     45514/tcp          # ASSIA CloudCheck WiFi Management System
spremotetablet 46998/tcp        # Capture handwritten signatures

```

- Add three lines below the specified line number

```

Shell > cat /root/test.txt | sed '5a 123\
comment yes\
tcp or udp'
aigairserver   21221/tcp          # Services for Air Server
ka-kdp         31016/udp          # Kollective Agent Kollective Delivery
ka-sddp         31016/tcp          # Kollective Agent Secure Distributed
Delivery
edi_service    34567/udp          # dhanalakshmi.org EDI Service
axio-disc      35100/tcp          # Axiomatic discovery protocol
123
comment yes
tcp or udp
axio-disc      35100/udp          # Axiomatic discovery protocol
pmwebapi       44323/tcp          # Performance Co-Pilot client HTTP API
cloudcheck-ping 45514/udp        # ASSIA CloudCheck WiFi Management
keepalive
cloudcheck     45514/tcp          # ASSIA CloudCheck WiFi Management System
spremotetablet 46998/tcp        # Capture handwritten signatures

```

- Match a specific line based on a string and add 2 lines of content above it

```

Shell > cat /root/test.txt | sed '/tcp/iTCP\
UDP'
TCP
UDP
aigairserver   21221/tcp          # Services for Air Server
ka-kdp         31016/udp          # Kollective Agent Kollective Delivery
TCP
UDP
ka-sddp         31016/tcp          # Kollective Agent Secure Distributed
Delivery
edi_service    34567/udp          # dhanalakshmi.org EDI Service
TCP
UDP
axio-disc      35100/tcp          # Axiomatic discovery protocol
axio-disc      35100/udp          # Axiomatic discovery protocol
TCP

```

```

UDP
pmwebapi      44323/tcp          # Performance Co-Pilot client HTTP API
cloudcheck-ping 45514/udp        # ASSIA CloudCheck WiFi Management
keepalive
TCP
UDP
cloudcheck     45514/tcp          # ASSIA CloudCheck WiFi Management System
TCP
UDP
spremotetablet 46998/tcp        # Capture handwritten signatures

```

## 6. Replace lines ( c )

- Locate one or more lines based on a string and replace these lines of text

```

Shell > cat /root/test.txt | sed '/ser/c\TMP1 \
TMP2'
TMP1
TMP2
ka-kdp      31016/udp          # Kollective Agent Kollective Delivery
ka-sddp      31016/tcp           # Kollective Agent Secure Distributed
Delivery
TMP1
TMP2
axio-disc    35100/tcp           # Axiomatic discovery protocol
axio-disc    35100/udp           # Axiomatic discovery protocol
pmwebapi     44323/tcp           # Performance Co-Pilot client HTTP API
cloudcheck-ping 45514/udp        # ASSIA CloudCheck WiFi Management
keepalive
cloudcheck     45514/tcp          # ASSIA CloudCheck WiFi Management System
spremotetablet 46998/tcp        # Capture handwritten signatures

```

- Single line replacement

```

Shell > cat /root/test.txt | sed '7c REPLACE'
aigairserver  21221/tcp          # Services for Air Server
ka-kdp        31016/udp          # Kollective Agent Kollective Delivery
ka-sddp        31016/tcp           # Kollective Agent Secure Distributed
Delivery
edi_service   34567/udp           # dhanalakshmi.org EDI Service
axio-disc     35100/tcp           # Axiomatic discovery protocol
axio-disc     35100/udp           # Axiomatic discovery protocol
REPLACE
cloudcheck-ping 45514/udp        # ASSIA CloudCheck WiFi Management
keepalive

```

```
cloudcheck      45514/tcp          # ASSIA CloudCheck WiFi Management System
spremotetablet 46998/tcp          # Capture handwritten signatures
```

- Replace consecutive lines of text

```
Shell > cat /root/test.txt | sed '2,$c REPLACE1 \
replace2'
aigairserver    21221/tcp          # Services for Air Server
REPLACE1
replace2
```

- Replace even numbered lines

```
Shell > cat /root/test.txt | sed '2~2c replace'
aigairserver    21221/tcp          # Services for Air Server
replace
ka-sddp         31016/tcp          # Kollective Agent Secure Distributed
Delivery
replace
axio-disc       35100/tcp          # Axiomatic discovery protocol
replace
pmwebapi        44323/tcp          # Performance Co-Pilot client HTTP API
replace
cloudcheck      45514/tcp          # ASSIA CloudCheck WiFi Management System
replace
```

## 7. Read the contents of the file and append its contents below the matching line ( r )

```
Shell > cat /root/app.txt
append1
POSIX
UNIX

Shell > cat /root/test.txt | sed '/ping/r /root/app.txt'
aigairserver    21221/tcp          # Services for Air Server
ka-kdp          31016/udp          # Kollective Agent Kollective Delivery
ka-sddp         31016/tcp          # Kollective Agent Secure Distributed
Delivery
edi_service     34567/udp          # dhanalakshmi.org EDI Service
axio-disc       35100/tcp          # Axiomatic discovery protocol
axio-disc       35100/udp          # Axiomatic discovery protocol
pmwebapi        44323/tcp          # Performance Co-Pilot client HTTP API
cloudcheck-ping 45514/udp          # ASSIA CloudCheck WiFi Management
keepalive
append1
```

POSIX

UNIX

```
cloudcheck      45514/tcp          # ASSIA CloudCheck WiFi Management System
spremotetablet 46998/tcp          # Capture handwritten signatures
```

## 8. Write matching lines to other files ( w )

```
Shell > cat /root/test.txt | sed '/axio/w /root/storage.txt'
```

```
Shell > cat /root/storage.txt
axio-disc      35100/tcp          # Axiomatic discovery protocol
axio-disc      35100/udp          # Axiomatic discovery protocol
```

## 9. Read/append the next line of input into the "pattern space" ( n and N )

- Print the next line of the matching line

```
Shell > cat /root/test.txt
aigairserver   21221/tcp          # Services for Air Server
ka-kdp         31016/udp          # Kollective Agent Kollective Delivery
ka-sddp         31016/tcp          # Kollective Agent Secure Distributed
Delivery
edi_service    34567/udp          # dhanalakshmi.org EDI Service
axio-disc      35100/tcp          # Axiomatic discovery protocol
axio-disc      35100/udp          # Axiomatic discovery protocol
pmwebapi       44323/tcp          # Performance Co-Pilot client HTTP API
cloudcheck-ping 45514/udp        # ASSIA CloudCheck WiFi Management
keepalive
cloudcheck     45514/tcp          # ASSIA CloudCheck WiFi Management System
spremotetablet 46998/tcp          # Capture handwritten signatures

Shell > cat /root/test.txt | sed '/ping/{n;p}' -n
cloudcheck     45514/tcp          # ASSIA CloudCheck WiFi Management System
```



Tip

Multiple sed operation commands may affect each other, and you can use "{}" to reduce this effect.

- Print even text lines

First, read the first line, as an n command is present; the second line will be printed out, and so on.

```
Shell > cat -n /root/test.txt | sed -n '{n;p}'
# or
```

```
Shell > cat -n /root/test.txt | sed -n '2~2p'
2 ka-kdp          31016/udp      # Kollective Agent Kollective Delivery
4 edi_service     34567/udp      # dhanalakshmi.org EDI Service
6 axio-disc       35100/udp      # Axiomatic discovery protocol
8 cloudcheck-ping 45514/udp    # ASSIA CloudCheck WiFi Management
keepalive
10 spremotetablet 46998/tcp   # Capture handwritten signatures
```

- Print odd text lines

```
Shell > cat -n /root/test.txt | sed -n '{p;n}'
# or
Shell > cat -n /root/test.txt | sed -n '1~2p'
# or
Shell > cat -n /root/test.txt | sed 'n;d'
1 aigairserver   21221/tcp      # Services for Air Server
3 ka-sddp        31016/tcp      # Kollective Agent Secure Distributed
Delivery
5 axio-disc       35100/tcp      # Axiomatic discovery protocol
7 pmwebapi        44323/tcp      # Performance Co-Pilot client HTTP API
9 cloudcheck      45514/tcp      # ASSIA CloudCheck WiFi Management
System
```

- Print 3n lines

```
Shell > cat -n /root/test.txt | sed -n '{n;n;p}'
# or
Shell > cat -n /root/test.txt | sed -n '3~3p'
3 ka-sddp        31016/tcp      # Kollective Agent Secure Distributed
Delivery
6 axio-disc       35100/udp      # Axiomatic discovery protocol
9 cloudcheck      45514/tcp      # ASSIA CloudCheck WiFi Management
System
```

- N

Read the first line and append one line after encountering the `N` command. In this example, the "pattern space" is "1\n2". Finally, execute the `q` command to exit.

```
Shell > seq 1 10 | sed 'N;q'
1
2
```

Because there is no additional line after line 9, the output is as follows:

```
Shell > seq 1 9 | sed -n 'N;p'
1
2
3
4
5
6
7
8
```

When the last line is read, the `N` command is not executed and the output is as follows:

```
Shell > seq 1 9 | sed -n '$!N;p'
1
2
3
4
5
6
7
8
9
```

Merge two lines into one line. Replace the "\n" of the "pattern space" with a blank character.

```
Shell > seq 1 6 | sed 'N;{s/\n//g}'
12
34
56
```

## 10. Ignore case ( I )

There seems to be no information about ignoring case in `man 1 sed`.

```
Shell > echo -e "abc\nAbc" | sed -n 's/a/X/Igp'
Xbc
XBC
```

```
Shell > cat /etc/services | sed '/OEM/Ip' -n
oem-agent      3872/tcp          # OEM Agent
oem-agent      3872/udp          # OEM Agent
oemcacao-jmxmp 11172/tcp        # OEM cacao JMX-remoting access point
oemcacao-rmi   11174/tcp        # OEM cacao rmi registry access point
oemcacao-websvc 11175/tcp       # OEM cacao web service access point
```

```
Shell > cat /etc/services | sed -r '/(TCP)|(UDP)/Id'
```

```
Shell > cat /etc/services | sed -r '/(TCP)|(UDP)/Ic TMP'
```

## 11. Gets the total number of lines in a file

```
Shell > cat /etc/services | sed -n '$='
# or
Shell > cat /etc/services | wc -l

11473
```

## 6. awk command

---

In 1977, a programming language-level tool for processing text, named' awk', was born at Bell Labs. The name comes from the first letters of the last names of three famous people:

- Alfred **Aho**
- Peter **Weinberger**
- Brian **Kernighan**

Similar to shell (bash, csh, zsh, and ksh), `awk` has derivatives with the development of history:

- `awk` : Born in 1977 Bell Labs.
- `nawk` (new awk): It was born in 1985 and is an updated and enhanced version of `awk`. It was widely used with Unix System V Release 3.1 (1987). The old version of `awk` is called `oawk` (old awk).
- `gawk` (GNU awk): It was written by Paul Rubin in 1986. The GNU Project was born in 1984.
- `mawk` : was written in 1996 by Mike Brennan, the interpreter of the awk programming language.
- `jawk` : Implementation of `awk` in JAVA

In the GNU/Linux operating system, the usual `awk` refers to `gawk`. However, some distributions, such as Ubuntu or Debian, use `mawk` as their default `awk`.

In the Rocky Linux 8.8, `awk` refers to `gawk`.

```
Shell > whereis awk
awk: /usr/bin/awk /usr/libexec/awk /usr/share/awk /usr/share/man/man1/awk.1.gz

Shell > ls -l /usr/bin/awk
lrwxrwxrwx. 1 root root 4 4 16 2022 /usr/bin/awk -> gawk

Shell > rpm -qf /usr/bin/awk
gawk-4.2.1-4.el8.x86_64
```

For information not covered, see the [gawk manual](#).

Although `awk` is a tool for processing text, it has some programming language features:

- variable
- process control (loop)
- data type
- logical operation
- function
- array
- ...

**The working principle of `awk`** : Similar to relational databases, it supports processing fields (columns) and records (rows). By default, `awk` treats each line of a file as a record and places these records in memory for line-by-line processing, with a portion of each line treated as a field in the record. By default, delimiters to separate different fields use spaces and tabs, while numbers represent different fields in the row record. To reference multiple fields, separate them with commas or tabs.

A simple example that is easy to understand

1	2	3	4	5	6	7
8						
Filesystem	Type	Size	Used	Avail	Use%	Mounted
on	<-- 1 (first line)					
devtmpfs	devtmpfs	1.8G	0	1.8G	0%	/dev
	<-- 2					
tmpfs	tmpfs	1.8G	0	1.8G	0%	/dev/shm
	<-- 3					
tmpfs	tmpfs	1.8G	8.9M	1.8G	1%	/run
	<-- 4					
tmpfs	tmpfs	1.8G	0	1.8G	0%	/sys/fs/cgroup
	<-- 5					
/dev/nvme0n1p2	ext4	47G	2.6G	42G	6%	/
	<-- 6					
/dev/nvme0n1p1	xfs	1014M	182M	833M	18%	/boot
	<-- 7					

```
| tmpfs | tmpfs | 364M | 0 | 364M | 0% | /run/user/0
|       |<-- 8 (end line)
```

Shell > df -hT | awk '{print \$1,\$2}'

Filesystem Type

devtmpfs devtmpfs

tmpfs tmpfs

tmpfs tmpfs

/dev/nvme0n1p2 ext4

/dev/nvme0n1p1 xfs

tmpfs tmpfs

# \$0: Reference the entire text content.

Shell > df -hT | awk '{print \$0}'

Filesystem	Type	Size	Used	Avail	Use%	Mounted on
devtmpfs	devtmpfs	1.8G	0	1.8G	0%	/dev
tmpfs	tmpfs	1.8G	0	1.8G	0%	/dev/shm
tmpfs	tmpfs	1.8G	8.9M	1.8G	1%	/run
tmpfs	tmpfs	1.8G	0	1.8G	0%	/sys/fs/cgroup
/dev/nvme0n1p2	ext4	47G	2.6G	42G	6%	/
/dev/nvme0n1p1	xfs	1014M	182M	833M	18%	/boot
tmpfs	tmpfs	364M	0	364M	0%	/run/user/0

## 6.1 Instructions for using awk

---

The usage of awk is - awk option 'pattern {action}' FileName

**pattern:** Find specific content in the text **action:** Action instruction **{ }**: Group some instructions according to specific patterns

option	description
-f program-file --file program-file	Reading <code>awk</code> program source files from files
-F FS	Specify the separator for separating fields. The 'FS' here is a built-in variable in <code>awk</code> , with default values of spaces or tabs
-v var=value	variable assignment
--posix	Turn on compatibility mode
--dump-variables=[file]	Write global variables in <code>awk</code> to a file. If no file is specified, the default file is <code>awkvars.out</code>
--profile=[file]	Write performance analysis data to a specific file. If no file is specified, the default file is <code>awkprof.out</code>

pattern	description
BEGIN{ }	An action that is performed before all row records are read
END{ }	An action that is performed after all row records are read
/regular expression/	Match the regular expression for each input line record
pattern && pattern	Logic and operation
pattern    pattern	Logic or operation
!pattern	Logical negation operation
pattern1,pattern2	Specify the pattern range to match all row records within that range

`awk` is powerful and involves a lot of knowledge, so some of the content will be explained later.

### 6.1.1 `printf` commands

Before formally learning `awk`, beginners need to understand the command `printf`.

`printf` format and print data. Its usage is - `printf FORMAT [ARGUMENT]...`

**FORMAT** Used to control the content of the output. The following common interpretation sequences are supported

- **\a** - alert (BEL)
- **\b** - backspace
- **\f** - form feed
- **\n** - new line
- **\r** - carriage return
- **\t** - horizontal tab
- **\v** - vertical tab
- **%Ns** - The output string. The N represents the number of strings, for example:

```
%s %s %s
```

- **%Ni** - Output integers. The N represents the number of integers of the output, for example: `%i %i`
- **%m.nf** - Output Floating Point Number. The m represents the total number of digits output, and the n represents the number of digits after the decimal point. For example: `%8.5f`

**ARGUMENT:** If it is a file, you need to do some preprocessing to output correctly.

```
Shell > cat /tmp/printf.txt
ID      Name    Age     Class
1       Frank   20      3
2       Jack    25      5
3       Django  16      6
4       Tom     19      7

# Example of incorrect syntax:
Shell > printf '%s %s $s\n' /tmp/printf.txt
/tmp/printf.txt

# Change the format of the text
Shell > printf '%s' $(cat /tmp/printf.txt)
IDNameAgeClass1Frank2032Jack2553Django1664Tom197
# Change the format of the text
Shell > printf '%s\t%s\t%s\n' $(cat /tmp/printf.txt)
ID      Name    Age
Class  1       Frank
20     3       2
```

```

Jack    25      5
3       Django  16
6       4       Tom
19      7

Shell > printf "%s\t%s\t%s\t%s\n" a b c d 1 2 3 4
a      b      c      d
1      2      3      4

```

No `print` command exists in RockyLinux OS. You can only use `print` in `awk`, and its difference from `printf` is that it automatically adds a newline at the end of each line. For example:

```

Shell > awk '{printf $1 "\t" $2"\n"}' /tmp/printf.txt
ID      Name
1       Frank
2       Jack
3       Django
4       Tom

Shell > awk '{print $1 "\t" $2}' /tmp/printf.txt
ID      Name
1       Frank
2       Jack
3       Django
4       Tom

```

## 6.2 Basic usage example

---

### 1. Reading `awk` program source files from files

```

Shell > vim /tmp/read-print.awk
#!/bin/awk
{print $6}

Shell > df -hT | awk -f /tmp/read-print.awk
Use%
0%
0%
1%
0%
6%
18%
0%

```

## 2. Specify delimiter

```
Shell > awk -F ":" '{print $1}' /etc/passwd
root
bin
daemon
adm
lp
sync
...
Shell > tail -n 5 /etc/services | awk -F "\/" '{print $2}'
awk: warning: escape sequence `\/' treated as plain `/'
axio-disc      35100
pmwebapi       44323
cloudcheck-ping 45514
cloudcheck     45514
spremotetablet 46998
```

You can also use words as delimiters. Parentheses indicate this is an overall delimiter, and "|" means or.

```
Shell > tail -n 5 /etc/services | awk -F "(tcp)|(udp)" '{print $1}'
axio-disc      35100/
pmwebapi       44323/
cloudcheck-ping 45514/
cloudcheck     45514/
spremotetablet 46998/
```

## 3. Variable assignment

```
Shell > tail -n 5 /etc/services | awk -v a=123 'BEGIN{print a}{print $1}'
123
axio-disc
pmwebapi
cloudcheck-ping
cloudcheck
spremotetablet
```

Assign the value of user-defined variables in bash to awk's variables.

```
Shell > ab=123
Shell > echo ${ab}
123
```

```
Shell > tail -n 5 /etc/services | awk -v a=${ab} 'BEGIN{print a}{print $1}'
123
axio-disc
pmwebapi
cloudcheck-ping
cloudcheck
spremotetablet
```

#### 4. Write awk's global variables to a file

```
Shell > seq 1 6 | awk --dump-variables '{print $0}'
1
2
3
4
5
6

Shell > cat /root/awkvars.out
ARGC: 1
ARGIND: 0
ARGV: array, 1 elements
BINMODE: 0
CONVFMT: "%.6g"
ENVIRON: array, 27 elements
ERRNO: ""
FIELDWIDTHS: ""
FILENAME: "-"
FNR: 6
FPAT: "[^[:space:]]+"
FS: " "
FUNCTAB: array, 41 elements
IGNORECASE: 0
LINT: 0
NF: 1
NR: 6
OFMT: "%.6g"
OFS: " "
ORS: "\n"
PREC: 53
PROCINFO: array, 20 elements
RLENGTH: 0
ROUNDMODE: "N"
RS: "\n"
RSTART: 0
RT: "\n"
SUBSEP: "\034"
```

```
SYMTAB: array, 28 elements
TEXTDOMAIN: "messages"
```

Later, we will introduce what these variables mean. To review them now, [jump to variables](#).

## 5. BEGIN{ } and END{ }

```
Shell > head -n 5 /etc/passwd | awk 'BEGIN{print
"UserName:PasswordIdentification:UID:InitGID"}{print $0}END{print "one\ntwo"}'
UserName:PasswordIdentification:UID:InitGID
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
one
two
```

## 6. --profile option

```
Shell > df -hT | awk --profile
'BEGIN{print "start line"}{print $0}END{print "end line"}'
start line
Filesystem      Type      Size  Used Avail Use% Mounted on
devtmpfs        devtmpfs  1.8G   0  1.8G  0% /dev
tmpfs           tmpfs    1.8G   0  1.8G  0% /dev/shm
tmpfs           tmpfs    1.8G  8.9M  1.8G  1% /run
tmpfs           tmpfs    1.8G   0  1.8G  0% /sys/fs/cgroup
/dev/nvme0n1p2 ext4     47G  2.7G  42G  6% /
/dev/nvme0n1p1 xfs    1014M 181M  834M 18% /boot
tmpfs           tmpfs    363M   0  363M  0% /run/user/0
end line
```

```
Shell > cat /root/awkprof.out
# gawk profile, created Fri Dec  8 15:12:56 2023
```

```
# BEGIN rule(s)

BEGIN {
1      print "start line"
}

# Rule(s)

8 {
```

```

8          print $0
}

# END rule(s)

END {
1      print "end line"
}

```

Modify the awkprof.out file.

```

Shell > vim /root/awkprof.out
BEGIN {
    print "start line"
}

{
    print $0
}

END {
    print "end line"
}

Shell > df -hT | awk -f /root/awkprof.out
start line
Filesystem      Type  Size  Used Avail Use% Mounted on
devtmpfs        devtmpfs 1.8G   0  1.8G  0% /dev
tmpfs           tmpfs   1.8G   0  1.8G  0% /dev/shm
tmpfs           tmpfs   1.8G  8.9M  1.8G  1% /run
tmpfs           tmpfs   1.8G   0  1.8G  0% /sys/fs/cgroup
/dev/nvme0n1p2 ext4    47G  2.7G  42G  6% /
/dev/nvme0n1p1 xfs    1014M 181M  834M 18% /boot
tmpfs           tmpfs   363M   0  363M  0% /run/user/0
end line

```

## 7. Match rows (records) through regular expressions

```

Shell > cat /etc/services | awk '/[^0-9a-zA-Z]1[1-9]{2}\tcp/ {print $0}'
sunrpc      111/tcp      portmapper rpcbind      # RPC 4.0 portmapper TCP
auth        113/tcp      authentication tap ident
sftp         115/tcp
uucp-path   117/tcp
nntp        119/tcp      readnews untp      # USENET News Transfer Protocol
ntp          123/tcp
netbios-ns  137/tcp      # NETBIOS Name Service

```

```
netbios-dgm      138/tcp          # NETBIOS Datagram Service
netbios-ssn      139/tcp          # NETBIOS session service
...
...
```

## 8. Logical operations (logical and, logical OR, reverse)

logical and: && logical OR: || reverse: !

```
Shell > cat /etc/services | awk '/[^0-9a-zA-Z]1[1-9]{2}\tcp/ && /175/ {print $0}'
vmnet           175/tcp          # VMNET
```

```
Shell > cat /etc/services | awk '/[^0-9a-zA-Z]9[1-9]{2}\tcp/ || /91{2}\tcp/ {print $0}'
telnets         992/tcp          #
imaps           993/tcp          # IMAP over SSL
pop3s           995/tcp          # POP-3 over SSL
mtp              1911/tcp         #
rndc             953/tcp          # rndc control sockets (BIND 9)
xact-backup     911/tcp          # xact-backup
apex-mesh        912/tcp          # APEX relay-relay service
apex-edge        913/tcp          # APEX endpoint-relay service
ftps-data        989/tcp          # ftp protocol, data, over TLS/SSL
nas               991/tcp          # Netnews Administration System
vsinet            996/tcp          # vsinet
maitrd            997/tcp          #
busboy            998/tcp          #
garcon            999/tcp          #
#puprouter        999/tcp          #
blockade          2911/tcp          # Blockade
prnstatus         3911/tcp          # Printer Status Port
cpdlc             5911/tcp          # Controller Pilot Data Link
Communication
manyone-xml       8911/tcp          # manyone-xml
sype-transport    9911/tcp          # SYPECom Transport Protocol
```

```
Shell > cat /etc/services | awk '!/(tcp)|(udp)/ {print $0}'
discard          9/sctp          # Discard
discard          9/dccp          # Discard SC:DISC
ftp-data          20/sctp          # FTP
ftp               21/sctp          # FTP
ssh               22/sctp          # SSH
exp1              1021/sctp        # RFC3692-style Experiment 1
(*)                [RFC4727]
exp1              1021/dccp        # RFC3692-style Experiment 1
(*)                [RFC4727]
```

```

exp2          1022/sctp           # RFC3692-style Experiment 2
(*)          [RFC4727]
exp2          1022/dccp           # RFC3692-style Experiment 2
(*)          [RFC4727]
ltp-deepspace 1113/dccp         # Licklider Transmission Protocol
cisco-ipsla   1167/sctp          # Cisco IP SLAs Control Protocol
rcip-itu      2225/sctp          # Resource Connection Initiation Protocol
m2ua          2904/sctp          # M2UA
m3ua          2905/sctp          # M3UA
megaco-h248   2944/sctp          # Megaco-H.248 text
...

```

## 9. Locates consecutive lines by string and prints them

```

Shell > cat /etc/services | awk '/^ntp/,/^netbios/ {print $0}'
ntp          123/tcp             # Network Time Protocol
ntp          123/udp             # NETBIOS Name Service
netbios-ns   137/tcp             # NETBIOS Name Service

```

 **Info**

Start range: stop matching when the first match is encountered. End range: stop matching when the first match is encountered.

## 6.3 Built-in variable

---

<b>Variable name</b>	<b>Description</b>
FS	The delimiter of the input field. The default is space or tab
OFS	The delimiter of the output field. The default is space
RS	The delimiter of the input row record. The default is a newline character (\n)
ORS	The delimiter of output row record. The default is a newline character (\n)
NF	Count the number of fields in the current row record
NR	Count the number of row records. After each line of text is processed, the value of this variable will be +1
FNR	Count the number of row records. When the second file is processed, the NR variable continues to add up, but the FNR variable is recounted
ARGC	The number of command line arguments
ARGV	An array of command line arguments, with subscript starting at 0 and ARGV[0] representing awk
ARGIND	The index value of the file currently being processed. The first file is 1, the second file is 2, and so on
ENVIRON	Environment variables of the current system
FILENAME	Output the currently processed file name
IGNORECASE	Ignore case
SUBSEP	The delimiter of the subscript in the array, which defaults to "\034"

## 1. FS and OFS

```
Shell > cat /etc/passwd | awk 'BEGIN{FS=":"}{print $1}'
root
bin
daemon
adm
lp
sync
```

You can also use the `-v` option to assign values to variables.

```
Shell > cat /etc/passwd | awk -v FS=: '{print $1}'
root
bin
daemon
adm
lp
sync
```

The default output delimiter is a space when using commas to reference multiple fields. You can, however, specify the output delimiter separately.

```
Shell > cat /etc/passwd | awk 'BEGIN{FS=":"}{print $1,$2}'
root x
bin x
daemon x
adm x
lp x
```

```
Shell > cat /etc/passwd | awk 'BEGIN{FS=":";OFS="\t"}{print $1,$2}'
# or
Shell > cat /etc/passwd | awk -v FS=: -v OFS="\t" '{print $1,$2}'
root    x
bin    x
daemon x
adm    x
lp     x
```

## 2. RS and ORS

By default, `awk` uses newline characters to distinguish each line record

```
Shell > echo -e "https://example.com/books/index.html\n\ttitle//tcp"
https://example.com/books/index.html
title//tcp

Shell > echo -e "https://example.com/books/index.html\n\ttitle//tcp" | awk
'BEGIN{RS="\n";ORS="%%"}{print $0}'
awk: cmd. line:1: warning: escape sequence `\/' treated as plain `/'
https:%%example.com/books/index.html
title%%tcp
%%           ← Why? Because "print"
```

### 3. NF

Count the number of fields per line in the current text

```
Shell > head -n 5 /etc/passwd | awk -F ":" 'BEGIN{RS="\n";ORS="\n"} {print NF}'
7
7
7
7
7
```

Print the fifth field

```
Shell > head -n 5 /etc/passwd | awk -F ":" 'BEGIN{RS="\n";ORS="\n"} {print $(NF-2)}'
root
bin
daemon
adm
lp
```

Print the last field

```
Shell > head -n 5 /etc/passwd | awk -F ":" 'BEGIN{RS="\n";ORS="\n"} {print $NF}'
/bin/bash
/sbin/nologin
/sbin/nologin
/sbin/nologin
/sbin/nologin
```

Exclude the last two fields

```
Shell > head -n 5 /etc/passwd | awk -F ":" '$NF-1=" ";print $0'
root x 0 0 root
bin x 1 1 bin
daemon x 2 2 daemon
adm x 3 4 adm
lp x 4 7 lp
```

## Exclude the first field

```
Shell > head -n 5 /etc/passwd | awk -F ":" '$NF-1=" ";print $0' | sed -r 's/(^ )//g'
x 0 0 root /root /bin/bash
x 1 1 bin /bin /sbin/nologin
x 2 2 daemon /sbin /sbin/nologin
x 3 4 adm /var/adm /sbin/nologin
x 4 7 lp /var/spool/lpd /sbin/nologin
```

## 4. NR and FNR

```
Shell > tail -n 5 /etc/services | awk '{print NR,$0}'
1 axio-disc      35100/udp          # Axiomatic discovery protocol
2 pmwebapi       44323/tcp          # Performance Co-Pilot client HTTP API
3 cloudcheck-ping 45514/udp        # ASSIA CloudCheck WiFi Management
keepalive
4 cloudcheck     45514/tcp          # ASSIA CloudCheck WiFi Management
System
5 spremotetablet 46998/tcp        # Capture handwritten signatures
```

## Print the total number of lines in the file content

```
Shell > cat /etc/services | awk 'END{print NR}'
11473
```

## Print the content of line 200

```
Shell > cat /etc/services | awk 'NR==200'
microsoft-ds    445/tcp
```

## Print the second field on line 200

```
Shell > cat /etc/services | awk 'BEGIN{RS="\n";ORS="\n"} NR==200 {print $2}'
445/tcp
```

## Print content within a specific range

```
Shell > cat /etc/services | awk 'BEGIN{RS="\n";ORS="\n"} NR<=10 {print NR,$0}'
1 # /etc/services:
2 # $Id: services,v 1.49 2017/08/18 12:43:23 ovasik Exp $
3 #
4 # Network services, Internet style
5 # IANA services version: last updated 2016-07-08
6 #
7 # Note that it is presently the policy of IANA to assign a single well-known
8 # port number for both TCP and UDP; hence, most entries here have two entries
9 # even if the protocol doesn't support UDP operations.
10 # Updated from RFC 1700, ``Assigned Numbers'' (October 1994). Not all ports
```

## Comparison between NR and FNR

```
Shell > head -n 3 /etc/services > /tmp/a.txt

Shell > cat /tmp/a.txt
# /etc/services:
# $Id: services,v 1.49 2017/08/18 12:43:23 ovasik Exp $
#

Shell > cat /etc/resolv.conf
# Generated by NetworkManager
nameserver 8.8.8.8
nameserver 114.114.114.114

Shell > awk '{print NR,$0}' /tmp/a.txt /etc/resolv.conf
1 # /etc/services:
2 # $Id: services,v 1.49 2017/08/18 12:43:23 ovasik Exp $
3 #
4 # Generated by NetworkManager
5 nameserver 8.8.8.8
6 nameserver 114.114.114.114

Shell > awk '{print FNR,$0}' /tmp/a.txt /etc/resolv.conf
1 # /etc/services:
2 # $Id: services,v 1.49 2017/08/18 12:43:23 ovasik Exp $
3 #
4 # Generated by NetworkManager
```

```
2 nameserver 8.8.8.8
3 nameserver 114.114.114.114
```

## 5. ARGC and ARGV

```
Shell > awk 'BEGIN{print ARGC}' log dump long
4
Shell > awk 'BEGIN{print ARGV[0]}' log dump long
awk
Shell > awk 'BEGIN{print ARGV[1]}' log dump long
log
Shell > awk 'BEGIN{print ARGV[2]}' log dump long
dump
```

## 6. ARGIND

This variable is mainly used to determine the file the `awk` program is working on.

```
Shell > awk '{print ARGIND,$0}' /etc/hostname /etc/resolv.conf
1 Master
2 # Generated by NetworkManager
2 nameserver 8.8.8.8
2 nameserver 114.114.114.114
```

## 7. ENVIRON

You can reference operating systems or user-defined variables in `awk` programs.

```
Shell > echo ${SSH_CLIENT}
192.168.100.2 6969 22

Shell > awk 'BEGIN{print ENVIRON["SSH_CLIENT"]}'
192.168.100.2 6969 22

Shell > export a=123
Shell > env | grep -w a
a=123
Shell > awk 'BEGIN{print ENVIRON["a"]}'
123
Shell > unset a
```

## 8. FILENAME

```
Shell > awk 'BEGIN{RS="\n";ORS="\n"} NR=FNR {print ARGIND,FILENAME---"$0}' /etc/
hostname /etc/resolv.conf /etc/rocky-release
1 /etc/hostname---Master
2 /etc/resolv.conf---# Generated by NetworkManager
2 /etc/resolv.conf---nameserver 8.8.8.8
2 /etc/resolv.conf---nameserver 114.114.114.114
3 /etc/rocky-release---Rocky Linux release 8.9 (Green Obsidian)
```

## 9. IGNORECASE

This variable is useful if you want to use regular expressions in `awk` and ignore case.

```
Shell > awk 'BEGIN{IGNORECASE=1;RS="\n";ORS="\n"} /^(\SSH)|^(ftp)/ {print $0}' /
etc/services
ftp-data      20/tcp
ftp-data      20/udp
ftp          21/tcp
ftp          21/udp      fsp fspd
ssh           22/tcp      # The Secure Shell (SSH) Protocol
ssh           22/udp      # The Secure Shell (SSH) Protocol
ftp-data      20/sctp     # FTP
ftp          21/sctp     # FTP
ssh           22/sctp     # SSH
ftp-agent    574/tcp      # FTP Software Agent System
ftp-agent    574/udp      # FTP Software Agent System
sshshell     614/tcp      # SSLshell
sshshell     614/udp      #       SSLshell
ftps-data    989/tcp      # ftp protocol, data, over TLS/SSL
ftps-data    989/udp      # ftp protocol, data, over TLS/SSL
ftps         990/tcp      # ftp protocol, control, over TLS/SSL
ftps         990/udp      # ftp protocol, control, over TLS/SSL
ssh-mgmt    17235/tcp     # SSH Tectia Manager
ssh-mgmt    17235/udp     # SSH Tectia Manager
```

```
Shell > awk 'BEGIN{IGNORECASE=1;RS="\n";ORS="\n"} /^(SMTP)\s/,/^(\TFTP)\s/ {print
$0}' /etc/services
smtp          25/tcp      mail
smtp          25/udp      mail
time          37/tcp      timserver
time          37/udp      timserver
rlp           39/tcp      resource      # resource location
rlp           39/udp      resource      # resource location
nameserver   42/tcp      name        # IEN 116
nameserver   42/udp      name        # IEN 116
nicname      43/tcp      whois
nicname      43/udp      whois
```

tacacs	<b>49/tcp</b>	# Login Host Protocol (TACACS)
tacacs	<b>49/udp</b>	# Login Host Protocol (TACACS)
re-mail-ck	<b>50/tcp</b>	# Remote Mail Checking Protocol
re-mail-ck	<b>50/udp</b>	# Remote Mail Checking Protocol
domain	<b>53/tcp</b>	# name-domain server
domain	<b>53/udp</b>	
whois++	<b>63/tcp</b>	whoispp
whois++	<b>63/udp</b>	whoispp
bootps	<b>67/tcp</b>	# BOOTP server
bootps	<b>67/udp</b>	
bootpc	<b>68/tcp</b>	dhcpclient
bootpc	<b>68/udp</b>	dhcpclient
tftp	<b>69/tcp</b>	

## 6.4 Operator

---

Operator	Description
(...)	Grouping
\$n	Field reference
++	Incremental
--	Decreasing
+	Mathematical plus sign
-	Mathematical minus sign
!	Negation
*	Mathematical multiplication sign
/	Mathematical division sign
%	Modulo operation
in	Elements in an array
&&	Logic and Operations
	Logical OR operation
?:	Abbreviation of conditional expressions
~	Another representation of regular expressions
!~	Reverse Regular Expression

 **Note**

In the `awk` program, the following expressions will be judged as **false**:

- The number is 0;
- Empty string;
- Undefined value.

```
Shell > awk 'BEGIN{n=0;if(n) print "Ture";else print "False"}'  
False  
Shell > awk 'BEGIN{s="";if(s) print "True";else print "False"}'  
False  
Shell > awk 'BEGIN{if(t) print "True";else print "Flase"}'  
False
```

## 1. Exclamation point

Print odd rows:

```
Shell > seq 1 10 | awk 'i!=i {print $0}'
1
3
5
7
9
```

### Question

**Why? Read the first line:** Because "i" is not assigned a value, so "i!=i" indicates TRUE. **Read the second line:** At this point, "i!=i" indicates FALSE. And so on, the final printed line is an odd number.

Print even rows:

```
Shell > seq 1 10 | awk '! (i!=i)'
# or
Shell > seq 1 10 | awk '! (i!=i) {print $0}'
2
4
6
8
10
```

### Note

As you can see, sometimes you can ignore the syntax for the "action" part, which by default is equivalent to "{print \$0}".

## 2. Reversal

```
Shell > cat /etc/services | awk '!/(tcp)|(udp)|(^#)|(^$)/ {print $0}'
http          80/sctp                      # HyperText Transfer Protocol
bgp           179/sctp
https         443/sctp                     # http protocol over TLS/SSL
h323hostcall  1720/sctp                   # H.323 Call Control
nfs           2049/sctp        nfsd shlp   # Network File System
rtmp          1/ddp                         # Routing Table Maintenance
Protocol
nbp           2/ddp                         # Name Binding Protocol
echo          4/ddp                         # AppleTalk Echo Protocol
zip           6/ddp                         # Zone Information Protocol
discard       9/sctp                        # Discard
```

```
discard          9/dccp           # Discard SC:DISC
```

```
...
```

### 3. Basic operations in mathematics

```
Shell > echo -e "36\n40\n50" | awk '{print $0+1}'
37
41

Shell > echo -e "30\t5\t8\n11\t20\t34"
30      5      8
11      20     34
Shell > echo -e "30\t5\t8\n11\t20\t34" | awk '{print $2*2+1}'
11
41
```

It can also be used in the "pattern":

```
Shell > cat -n /etc/services | awk '/^[1-9]*/ && $1%2==0 {print $0}'
...
24  tcpmux          1/udp           # TCP port service
multiplexer
26  rje             5/udp           # Remote Job Entry
28  echo            7/udp
30  discard          9/udp           sink null
32  systat          11/udp          users
34  daytime          13/udp
36  qotd            17/udp          quote
...

Shell > cat -n /etc/services | awk '/^[1-9]*/ && $1%2!=0 {print $0}'
...
23  tcpmux          1/tcp            # TCP port service
multiplexer
25  rje             5/tcp            # Remote Job Entry
27  echo            7/tcp
29  discard          9/tcp           sink null
31  systat          11/tcp          users
...
```

### 4. Pipe symbol

You can use the bash command in the awk program, for example:

```
Shell > echo -e "6\n3\n9\n8" | awk '{print $0 | "sort"}'
3
6
8
9
```

**i Info**

Please pay attention! You must use double quotes to include the command.

## 5. Regular expression

Here, we cover basic examples of regular expressions. You can use regular expressions on row records.

```
Shell > cat /etc/services | awk '/[^0-9a-zA-Z]1[1-9]{2}\tcp/ {print $0}'

# Be equivalent to:

Shell > cat /etc/services | awk '$0~/[^0-9a-zA-Z]1[1-9]{2}\tcp/ {print $0}'
```

If the file has a large amount of text, regular expressions can also be used for fields, which will help improve processing efficiency. The usage example is as follows:

```
Shell > cat /etc/services | awk '$0~/^(ssh)/ && $2~/tcp/ {print $0}'
ssh          22/tcp                                # The Secure Shell (SSH) Protocol
sshell        614/tcp                               # SSLshell
ssh-mgmt     17235/tcp                            # SSH Tectia Manager

Shell > cat /etc/services | grep -v -E "(^#)|(^$)" | awk '$2~/^(tcp)|(udp)/
{print $0}'
http         80/sctp                             # HyperText Transfer Protocol
bgp          179/sctp                            # http protocol over TLS/SSL
https        443/sctp                            # H.323 Call Control
h323hostcall 1720/sctp                           nfsd shlp      # Network File System
nfs          2049/sctp                           rtmp          1/ddp       # Routing Table Maintenance
Protocol
nbp          2/ddp                                # Name Binding Protocol
...
```

## 6.5 Flow control

---

### 1. if statement

The basic syntax format is - `if (condition) statement [ else statement ]`

Example of a single branch use of an if statement:

```
Shell > cat /etc/services | awk '{if(NR==110) print $0}'
pop3          110/udp          pop-3
```

The condition is determined as a regular expression:

```
Shell > cat /etc/services | awk '{if(/^(ftp)\s|^ssh\s/) print $0}'
ftp          21/tcp
ftp          21/udp          fsp  fspd
ssh          22/tcp          # The Secure Shell (SSH) Protocol
ssh          22/udp          # The Secure Shell (SSH) Protocol
ftp          21/sctp         # FTP
ssh          22/sctp         # SSH
```

Double branch:

```
Shell > seq 1 10 | awk '{if($0==10) print $0 ; else print "False"}'
False
False
False
False
False
False
False
False
False
10
```

Multiple branches:

```
Shell > cat /etc/services | awk '{ \
if($1~/netbios/)
    {print $0}
else if($2~/175/)
    {print "175"}
else if($2~/137/)
    {print "137"}}
```

```
else {print "no"
}'
```

## 2. while statement

The basic syntax format is - `while (condition) statement`

Traverse and print out the fields of all row records.

```
Shell > tail -n 2 /etc/services
cloudcheck      45514/tcp          # ASSIA CloudCheck WiFi Management System
spremotetablet 46998/tcp          # Capture handwritten signatures

Shell > tail -n 2 /etc/services | awk '{ \
i=1;
while(i<=NF){print $i;i++}
}'

cloudcheck
45514/tcp
#
ASSIA
CloudCheck
WiFi
Management
System
spremotetablet
46998/tcp
#
Capture
handwritten
signatures
```

## 3. for statement

The basic syntax format is - `for (expr1; expr2; expr3) statement`

Traverse and print out the fields of all row records.

```
Shell > tail -n 2 /etc/services | awk '{ \
for(i=1;i<=NF;i++) print $i
}'
```

Print the fields for each row of records in reverse order.

```
Shell > tail -n 2 /etc/services | awk '{ \
for(i=NF;i>=1;i--) print $i
}'
```

```
System
Management
WiFi
CloudCheck
ASSIA
#
45514/tcp
cloudcheck
signatures
handwritten
Capture
#
46998/tcp
spremotetablet
```

Print each line of records in the opposite direction.

```
Shell > tail -n 2 /etc/services | awk '{ \
for(i=NF;i>=1;i--) {printf $i" "}; \
print "" \
}'
```

```
System Management WiFi CloudCheck ASSIA # 45514/tcp cloudcheck
signatures handwritten Capture # 46998/tcp spremotetablet
```

#### 4. **break** statement and **continue** statement

The comparison between the two is as follows:

```
Shell > awk 'BEGIN{ \
for(i=1;i<=10;i++) \
{ \
    if(i==3) {break}; \
    print i \
} \
}'
```

```
1
2
```

```
Shell > awk 'BEGIN{ \
for(i=1;i<=10;i++)
{
    if(i==3) {continue};
    print i
}
}'
```

```
1
2
4
5
6
7
8
9
10
```

## 5. exit statement

You can specify a return value in the range of [0,255]

The basic syntax format is - `exit [expression]`

```
Shell > seq 1 10 | awk '{
    if($0~/5/) exit "135"
}'
```

```
Shell > echo $?
135
```

## 6.6 Array

---

**array:** A collection of data with the same data type arranged in a certain order. Each data in an array is called an element.

Like most programming languages, `awk` also supports arrays, which are divided into **indexed arrays (with numbers as subscripts)** and **associative arrays (with strings as subscripts)**.

`awk` has a lot of functions, and the functions related to arrays are:

- **length(Array\_Name)** - Get the length of the array.
- Custom array

Format - `Array_Name[Index]=Value`

```
Shell > awk 'BEGIN{a1[0]="test0" ; a1[1]="s1"; print a1[0]}'  
test0
```

Get the length of the array:

```
Shell > awk 'BEGIN{name[-1]="jimcat8" ; name[3]="jack" ; print length(name)}'  
2
```

Store all GNU/Linux users in an array:

```
Shell > cat /etc/passwd | awk -F ":" '{username[NR]=$1}END{print username[2]}'  
bin  
Shell > cat /etc/passwd | awk -F ":" '{username[NR]=$1}END{print username[1]}'  
root
```

#### Info

The numeric subscript of an `awk` array can be a positive integer, a negative integer, a string, or 0, so the numeric subscript of an `awk` array has no concept of an initial value. This is not the same as arrays in `bash`.

```
Shell > arr1=(2 10 30 string1)  
Shell > echo "${arr1[0]}"  
2  
Shell > unset arr1
```

- Delete array

Format - `delete Array_Name`

- Delete an element from an array

Format - `delete Array_Name[Index]`

- Traversal array

You can use the **for** statement, which is suitable for cases where the array subscript is unknown:

```
Shell > head -n 5 /etc/passwd | awk -F ":" '\  
{  
    username[NR]=$1
```

```

}
END {
    for(i in username)
        print username[i],i
}
'

root 1
bin 2
daemon 3
adm 4
lp 5

```

If the subscript of an array is regular, you can use this form of the **for** statement:

```

Shell > cat /etc/passwd | awk -F ":" '\
{
    username[NR]=$1
}
END{
    for(i=1;i<=NR;i++)
        print username[i],i
}
'

root 1
bin 2
daemon 3
adm 4
lp 5
sync 6
shutdown 7
halt 8
...

```

- Use "++" as the subscript of the array

```

Shell > tail -n 5 /etc/group | awk -F ":" '\
{
    a[x++]=$1
}
END{
    for(i in a)
        print a[i],i
}
'

```

```
slocate 0
unbound 1
docker 2
cgred 3
redis 4
```

- Use a field as the subscript of an array

```
Shell > tail -n 5 /etc/group | awk -F ":" '\
{
    a[$1] = $3
}
END{
    for(i in a)
        print a[i], i
}
'

991 docker
21 slocate
989 redis
992 unbound
990 cgred
```

- Count the number of occurrences of the same field

Count the number of occurrences of the same IPv4 address. Basic idea:

- First use the `grep` command to filter out all IPv4 addresses
- Then hand it over to the `awk` program for processing

```
Shell > cat /var/log/secure | egrep -o "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}" | awk ' \
{
    a[$1]++;
}
END{
    for(v in a) print a[v], v
}
'

4 0.0.0.0
4 192.168.100.2
```

### Info

`a[$1]++` is equivalent to `a[$1]+=1`

Count the number of occurrences of words regardless of case. Basic idea:

- Split all fields into multiple rows of records
- Then hand it over to the `awk` program for processing

```
Shell > cat /etc/services | awk -F " " '{for(i=1;i<=NF;i++) print $i}' | awk 'BEGIN{IGNORECASE=1;OFS="\t"} /netbios$/ || /ftp$/ {a[$1]++} END{for(v in a) print a[v],v}' | awk '3' | awk '18' | awk '7'

3      NETBIOS
18     FTP
7      ftp

Shell > cat /etc/services | awk -F " " '{ for(i=1;i<=NF;i++) print $i }' | awk 'BEGIN{IGNORECASE=1;OFS="\t"} /netbios$/ || /ftp$/ {a[$1]++} END{for(v in a) \ if(a[v]>=5) print a[v],v}' | awk '18' | awk '7'

18     FTP
7      ftp
```

You can first filter specific row records and then perform statistics, such as:

```
Shell > ss -tulnp | awk -F " " '/tcp/ {a[$2]++} END{for(i in a) print a[i],i}' | awk '2'
2 LISTEN
```

- Print lines based on the number of occurrences of a specific field

```
Shell > tail /etc/services
aigairserver    21221/tcp          # Services for Air Server
ka-kdp         31016/udp          # Kollective Agent Kollective Delivery
ka-sddp         31016/tcp          # Kollective Agent Secure Distributed
Delivery
edi_service     34567/udp          # dhanalakshmi.org EDI Service
```

```

axio-disc      35100/tcp          # Axiomatic discovery protocol
axio-disc      35100/udp          # Axiomatic discovery protocol
pmwebapi       44323/tcp          # Performance Co-Pilot client HTTP API
cloudcheck-ping 45514/udp        # ASSIA CloudCheck WiFi Management
keepalive
cloudcheck     45514/tcp          # ASSIA CloudCheck WiFi Management
System
spremotetablet 46998/tcp        # Capture handwritten signatures

Shell > tail /etc/services | awk 'a[$1]++ {print $0}'
axio-disc      35100/udp          # Axiomatic discovery protocol

```

Reverse:

```

Shell > tail /etc/services | awk '!a[$1]++ {print $0}'
aigairserver   21221/tcp          # Services for Air Server
ka-kdp         31016/udp          # Kollective Agent Kollective Delivery
ka-sddp         31016/tcp          # Kollective Agent Secure Distributed
Delivery
edi_service    34567/udp          # dhanalakshmi.org EDI Service
axio-disc       35100/tcp          # Axiomatic discovery protocol
pmwebapi        44323/tcp          # Performance Co-Pilot client HTTP API
cloudcheck-ping 45514/udp        # ASSIA CloudCheck WiFi Management
keepalive
cloudcheck     45514/tcp          # ASSIA CloudCheck WiFi Management
System
spremotetablet 46998/tcp        # Capture handwritten signatures

```

- Multidimensional array

The `awk` program does not support multi-dimensional arrays, but support for multi-dimensional arrays is achievable through simulation. By default, "\034" is the delimiter for the subscript of a multidimensional array.

Please note the following differences when using multidimensional arrays:

```

Shell > awk 'BEGIN{ a["1,0"]=100 ; a[2,0]=200 ; a["3","0"]=300 ; for(i in a)
print a[i],i }'
200 20
300 30
100 1,0

```

Redefine the delimiter:

```
Shell > awk 'BEGIN{ SUBSEP="----" ; a["1,0"]=100 ; a[2,0]=200 ;
a["3","0"]=300 ; for(i in a) print a[i],i }'
300 3----0
200 2----0
100 1,0
```

## Reorder

```
Shell > awk 'BEGIN{ SUBSEP="----" ; a["1,0"]=100 ; a[2,0]=200 ;
a["3","0"]=300 ; for(i in a) print a[i],i | "sort" }'
100 1,0
200 2----0
300 3----0
```

Count the number of times the field appears:

```
Shell > cat c.txt
A 192.168.1.1 HTTP
B 192.168.1.2 HTTP
B 192.168.1.2 MYSQL
C 192.168.1.1 MYSQL
C 192.168.1.1 MQ
D 192.168.1.4 NGINX
```

```
Shell > cat c.txt | awk 'BEGIN{SUBSEP="----"} {a[$1,$2]++} END{for(i in a)
print a[i],i}'
1 A----192.168.1.1
2 B----192.168.1.2
2 C----192.168.1.1
1 D----192.168.1.4
```

## 6.7 Built-in function

Function name	Description
int(expr)	Truncate as an integer
sqrt(expr)	Square root
rand()	Returns a random number N with a range of (0,1). The result is not that every run is a random number, but that it remains the same.
srand([expr])	Use "expr" to generate random numbers. If "expr" is not specified, the current time is used as the seed by default, and if there is a seed, the generated random number is used.
asort(a,b)	The elements of the array "a" are reordered (lexicographically) and stored in the new array "b", with the subscript in the array "b" starting at 1. This function returns the number of elements in the array.
asorti(a,b)	Reorder the subscript of the array "a" and store the sorted subscript in the new array "b" as an element, with the subscript of the array "b" starting at 1.
sub(r,s[,t])	Use the "r" regular expression to match the input records, and replace the matching result with "s". "t" is optional, indicating a replacement for a certain field. The function returns the number of replacements - 0 or 1. Similar to <code>sed s//</code>
gsub(r,s[,t])	Global replacement. "t" is optional, indicating the replacement of a certain field. If "t" is ignored, it indicates global replacement. Similar to <code>sed s///g</code>
gensub(r,s,h[,t])	The "r" regular expression matches the input records and replaces the matching result with "s". "t" is optional, indicating a replacement for a certain field. "h" represents replacing the specified index position
index(s,t)	Returns the index position of the string "t" in the string "s" (the string index starts from 1). If the function returns 0, it means it does not exist
length([s])	Returns the length of "s"
match(s,r[,a])	Test whether the string "s" contains the string "r". If included, return the index position of "r" within it (string index starting from 1). If not, return 0
split(s,a[,r[,seps]])	Split string "s" into an array "a" based on the delimiter "seps". The subscript of the array starts with 1.
substr(s,i[,n])	Intercept the string. "s" represents the string to be processed; "i" indicates the index position of the string; "n" is the length. If you do not specify "n", it means to intercept all remaining parts
tolower(str)	Converts all strings to lowercase
toupper(str)	Converts all strings to uppercase
systime()	Current timestamp
strftime([format[,timestamp[,utc-flag]]])	Format the output time. Converts the timestamp to a string

## 1. **int** function

```
Shell > echo -e "qwer123\n123\nabc\n123abc123\n100.55\n-155.27"
qwer123
123
abc
123abc123
100.55
-155.27

Shell > echo -e "qwer123\n123\nabc\n123abc123\n100.55\n-155.27" | awk '{print
int($1)}'
0
123
0
123
100
-155
```

As you can see, the int function only works for numbers, and when encountering a string, converts it to 0. When encountering a string starting with a number, truncates it.

## 2. **sqrt** function

```
Shell > awk 'BEGIN{print sqrt(9)}'
3
```

## 3. **rand** function and **srand** function

The example of using the rand function is as follows:

```
Shell > awk 'BEGIN{print rand()}'
0.924046
Shell > awk 'BEGIN{print rand()}'
0.924046
Shell > awk 'BEGIN{print rand()}'
0.924046
```

The example of using the srand function is as follows:

```
Shell > awk 'BEGIN{srand() ; print rand()}'
0.975495
Shell > awk 'BEGIN{srand() ; print rand()}'
```

```
0.99187
Shell > awk 'BEGIN{srand() ; print rand()}'
0.069002
```

Generate an integer within the range of (0,100):

```
Shell > awk 'BEGIN{srand() ; print int(rand()*100)}'
56
Shell > awk 'BEGIN{srand() ; print int(rand()*100)}'
33
Shell > awk 'BEGIN{srand() ; print int(rand()*100)}'
42
```

#### 4. **asort** function and **asorti** function

```
Shell > cat /etc/passwd | awk -F ":" '{a[NR]=$1} END{anu=asort(a,b) ;
for(i=1;i<=anu;i++) print i,b[i]}'
1 adm
2 bin
3 chrony
4 daemon
5 dbus
6 ftp
7 games
8 halt
9 lp
10 mail
11 nobody
12 operator
13 polkitd
14 redis
15 root
16 shutdown
17 sshd
18 sssd
19 sync
20 systemd-coredump
21 systemd-resolve
22 tss
23 unbound

Shell > awk
'BEGIN{a[1]=1000 ; a[2]=200 ; a[3]=30 ; a[4]="admin" ; a[5]="Admin" ; \
a[6]="12string" ; a[7]=-1 ; a[8]=-10 ; a[9]=-20 ; a[10]=-21 ;nu=asort(a,b) ;
for(i=1;i<=nu;i++) print i,b[i]}'
1 -21
```

```

2 -20
3 -10
4 -1
5 30
6 200
7 1000
8 12string
9 Admin
10 admin

```

### Info

Sorting rules:

- Numbers have higher priority than strings and are arranged in ascending order.
- Arrange strings in ascending dictionary order

If you are using the **asorti** function, the example is as follows:

```

Shell > awk 'BEGIN{ a[-11]=1000 ; a[-2]=200 ; a[-10]=30 ; a[-21]="admin" ;
a[41]="Admin" ; \
a[30]="12string" ; a["root"]="rootstr" ; a["Root"]="r1" ; nu=asorti(a,b) ; for(i
in b) print i,b[i] }'
1 -10
2 -11
3 -2
4 -21
5 30
6 41
7 Root
8 root

```

### Info

Sorting rules:

- Numbers have priority over strings
- If a negative number is encountered, the first digit from the left will be compared. If it is the same, the second digit will be compared, and so on
- If a positive number is encountered, it will be arranged in ascending order
- Arrange strings in ascending dictionary order

## 5. **sub** function and **gsub** function

```

Shell > cat /etc/services | awk '/netbios/ {sub(/tcp/,"test") ; print $0 }'
netbios-ns      137/test                                # NETBIOS Name Service
netbios-ns      137/udp
netbios-dgm     138/test                                # NETBIOS Datagram Service

```

```

netbios-dgm      138/udp
netbios-ssn      139/test                         # NETBIOS session service
netbios-ssn      139/udp

Shell > cat /etc/services | awk '/^ftp/ && /21\/tcp/ {print $0}'
ftp              21/tcp
↑               ↑
Shell > cat /etc/services | awk 'BEGIN{OFS="\t"} /^ftp/ && /21\/tcp/ {gsub(/p/, "P", $2) ; print $0}'
ftp              21/tcP
↑
Shell > cat /etc/services | awk 'BEGIN{OFS="\t"} /^ftp/ && /21\/tcp/ {gsub(/p/, "P") ; print $0}'
ftP              21/tcP
↑               ↑

```

Just like the `sed` command, you can also use the "&" symbol to reference already matched strings.

[Review that here.](#)

```

Shell > vim /tmp/tmp-file1.txt
A 192.168.1.1 HTTP
B 192.168.1.2 HTTP
B 192.168.1.2 MYSQL
C 192.168.1.1 MYSQL
C 192.168.1.1 MQ
D 192.168.1.4 NGINX

# Add a line of text before the second line
Shell > cat /tmp/tmp-file1.txt | awk 'NR==2 {gsub(/.*/, "add a line\n&")} {print $0}'
A 192.168.1.1 HTTP
add a line
B 192.168.1.2 HTTP
B 192.168.1.2 MYSQL
C 192.168.1.1 MYSQL
C 192.168.1.1 MQ
D 192.168.1.4 NGINX

# Add a string after the IP address in the second line
Shell > cat /tmp/tmp-file1.txt | awk 'NR==2 {gsub(/[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/, "&\tSTRING")} {print $0}'
A 192.168.1.1 HTTP
B 192.168.1.2 STRING HTTP
B 192.168.1.2 MYSQL
C 192.168.1.1 MYSQL

```

```
C 192.168.1.1 MQ
D 192.168.1.4 NGINX
```

## 6. index function

```
Shell > tail -n 5 /etc/services
axio-disc      35100/udp          # Axiomatic discovery protocol
pmwebapi       44323/tcp          # Performance Co-Pilot client HTTP API
cloudcheck-ping 45514/udp         # ASSIA CloudCheck WiFi Management
keepalive
cloudcheck      45514/tcp          # ASSIA CloudCheck WiFi Management System
spremotetablet 46998/tcp          # Capture handwritten signatures

Shell > tail -n 5 /etc/services | awk '{print index($2,"tcp")}'
0
7
0
7
7
```

## 7. length function

```
# The length of the output field
Shell > tail -n 5 /etc/services | awk '{print length($1)}'
9
8
15
10
14

# The length of the output array
Shell > cat /etc/passwd | awk -F ":" 'a[NR]="$1" END{print length(a)}'
22
```

## 8. match function

```
Shell > echo -e "1592abc144qszd\n144bc\nbn"
1592abc144qszd
144bc
bn

Shell > echo -e "1592abc144qszd\n144bc\nbn" | awk '{print match($1,144)}'
8
1
0
```

## 9. **split** function

```
Shell > echo "365%tmp%dir%number" | awk '{split($1,a1,"%") ; for(i in a1) print i,a1[i]}'
1 365
2 tmp
3 dir
4 number
```

## 10. **substr** function

```
Shell > head -n 5 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

# I need this part of the content - "emon:/sbin:/sbin/nologin"
Shell > head -n 5 /etc/passwd | awk '/daemon/ {print substr($0,16)}'
emon:/sbin:/sbin/nologin

Shell > tail -n 5 /etc/services
axio-disc      35100/udp          # Axiomatic discovery protocol
pmwebapi       44323/tcp          # Performance Co-Pilot client HTTP API
cloudcheck-ping 45514/udp         # ASSIA CloudCheck WiFi Management
keepalive
cloudcheck      45514/tcp          # ASSIA CloudCheck WiFi Management System
spremotetablet 46998/tcp          # Capture handwritten signatures

# I need this part of the content - "tablet"
Shell > tail -n 5 /etc/services | awk '/^sp/ {print substr($1,9)}'
tablet
```

## 11. **tolower** function and **toupper** function

```
Shell > echo -e "AbcD123\nqweR" | awk '{print tolower($0)}'
abcd123
qwer

Shell > tail -n 5 /etc/services | awk '{print toupper($0)}'
AXIO-DISC      35100/UDP          # AXIOMATIC DISCOVERY PROTOCOL
PMWEBAPI       44323/TCP          # PERFORMANCE CO-PILOT CLIENT HTTP API
CLOUDCHECK-PING 45514/UDP         # ASSIA CLOUDCHECK WIFI MANAGEMENT
KEEPALIVE
```

CLOUDCHECK	<b>45514</b> /TCP	# ASSIA CLOUDCHECK WIFI MANAGEMENT SYSTEM
SPREMOTETABLET	<b>46998</b> /TCP	# CAPTURE HANDWRITTEN SIGNATURES

## 12. Functions that deal with time and date

**What is a UNIX timestamp?** According to the development history of GNU/Linux, UNIX V1 was born in 1971, and the book "UNIX Programmer's Manual" was published on November 3 of the same year, which defines 1970-01-01 as the reference date of the start of UNIX.

The conversion between a timestamp and a natural date time in days:

```
Shell > echo "$(( $(date --date="2024/01/06" +%s)/86400 + 1 ))"
19728

Shell > date -d "1970-01-01 19728days"
Sat Jan  6 00:00:00 CST 2024
```

The conversion between a timestamp and a natural date time in seconds:

```
Shell > echo "$(date --date="2024/01/06 17:12:00" +%s)"
1704532320

Shell > echo "$(date --date='@1704532320')"
Sat Jan  6 17:12:00 CST 2024
```

The conversion between natural date time and UNIX timestamp in `awk` program:

```
Shell > awk 'BEGIN{print systime()}'
1704532597

Shell > echo "1704532597" | awk '{print strftime("%Y-%m-%d %H:%M:%S", $0)}'
2024-01-06 17:16:37
```

## 6.8 I/O statement

---

<b>Statement</b>	<b>Description</b>
getline	Read the next matching row record and assign it to "\$0". The return value is 1: Indicates that relevant row records have been read. The return value is 0: Indicates that the last line has been read The return value is negative: Indicates encountering an error
getline var	Read the next matching row record and assign it to the variable "var"
command   getline [var]	Assign the result to "\$0" or the variable "var"
next	Stop the current input record and perform the following actions
print	Print the result
printf	See <a href="#">here</a>
system(cmd-line)	Execute the command and return the status code. 0 indicates that the command was executed successfully; non-0 indicates that the execution failed
print ... >> file	Output redirection
print ...   command	Print the output and use it as input to the command

## 1. getline

```
Shell > seq 1 10 | awk '/3/ || /6/ {getline ; print $0}'
4
7

Shell > seq 1 10 | awk '/3/ || /6/ {print $0 ; getline ; print $0}'
3
4
6
7
```

Using the functions we learned earlier and the "&" symbol, we can:

```
Shell > tail -n 5 /etc/services | awk '/45514\/tcp/ {getline ; gsub(/.*/, "&\tSTRING1") ; print $0}'  
spremotetablet 46998/tcp # Capture handwritten signatures  
STRING1

Shell > tail -n 5 /etc/services | awk '/45514\/tcp/ {print $0 ; getline;  
gsub(/.*/, "&\tSTRING2") } {print $0}'  
axio-disc 35100/udp # Axiomatic discovery protocol  
pmwebapi 44323/tcp # Performance Co-Pilot client HTTP API  
cloudcheck-ping 45514/udp # ASSIA CloudCheck WiFi Management  
keepalive  
cloudcheck 45514/tcp # ASSIA CloudCheck WiFi Management System  
spremotetablet 46998/tcp # Capture handwritten signatures  
STRING2
```

Print even and odd lines:

```
Shell > tail -n 10 /etc/services | cat -n | awk '{ if( (getline) <= 1) print $0}'  
2 ka-kdp 31016/udp # Kollective Agent Kollective Delivery  
4 edi_service 34567/udp # dhanalakshmi.org EDI Service  
6 axio-disc 35100/udp # Axiomatic discovery protocol  
8 cloudcheck-ping 45514/udp # ASSIA CloudCheck WiFi Management  
keepalive  
10 spremotetablet 46998/tcp # Capture handwritten signatures

Shell > tail -n 10 /etc/services | cat -n | awk '{if(NR==1) print $0}  
{ if(NR%2==0) {if(getline > 0) print $0} }'  
1 aigairserver 21221/tcp # Services for Air Server  
3 ka-sddp 31016/tcp # Kollective Agent Secure Distributed  
Delivery  
5 axio-disc 35100/tcp # Axiomatic discovery protocol  
7 pmwebapi 44323/tcp # Performance Co-Pilot client HTTP API
```

```
9 cloudcheck      45514/tcp          # ASSIA CloudCheck WiFi Management
System
```

## 2. getline var

Add each line of the b file to the end of each line of the C file:

```
Shell > cat /tmp/b.txt
b1
b2
b3
b4
b5
b6

Shell > cat /tmp/c.txt
A 192.168.1.1 HTTP
B 192.168.1.2 HTTP
B 192.168.1.2 MYSQL
C 192.168.1.1 MYSQL
C 192.168.1.1 MQ
D 192.168.1.4 NGINX

Shell > awk '{getline var1 <"/tmp/b.txt" ; print $0 , var1}' /tmp/c.txt
A 192.168.1.1 HTTP b1
B 192.168.1.2 HTTP b2
B 192.168.1.2 MYSQL b3
C 192.168.1.1 MYSQL b4
C 192.168.1.1 MQ b5
D 192.168.1.4 NGINX b6
```

Replace the specified field of the c file with the content line of the b file:

```
Shell > awk '{ getline var2 < "/tmp/b.txt" ; gsub($2 , var2 , $2) ; print $0 }' /
tmp/c.txt
A b1 HTTP
B b2 HTTP
B b3 MYSQL
C b4 MYSQL
C b5 MQ
D b6 NGINX
```

## 3. command | getline [var]

```
Shell > awk 'BEGIN{ "date +%Y%m%d" | getline datenow ; print datenow }'
20240107
```

### Tip

Use double quotes to include Shell command.

## 4. next

Earlier, we introduced the **break** statement and the **continue** statement, the former used to terminate the loop, and the latter used to jump out of the current loop. See [here](#). For **next**, when the conditions are met, it will stop the input recording that meets the conditions and continue with subsequent actions.

```
Shell > seq 1 5 | awk '{if(NR==3) {next} print $0}'
1
2
4
5

# equivalent to
Shell > seq 1 5 | awk '{if($1!=3) print $0}'
```

Skip eligible line records:

```
Shell > cat /etc/passwd | awk -F ":" 'NR>5 {next} {print $0}'
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

# equivalent to
Shell > cat /etc/passwd | awk -F ":" 'NR>=1 && NR<=5 {print $0}'
```

### Tip

"**next**" cannot be used in "BEGIN{}" and "END{}".

## 5. system function

You can use this function to call commands in the Shell, such as:

```
Shell > awk 'BEGIN{ system("echo nginx http") }'
nginx http
```

### 🔥 Tip

Please note to add double quotes when using the `system` function. If not added, the `awk` program will consider it a variable of the `awk` program.

```
Shell > awk 'BEGIN{ cmd1="date +%Y" ; system(cmd1)}'
2024
```

**What if the Shell command itself contains double quotes?** Using escape characters - "\", such as:

```
Shell > egrep "^\root|^\nobody" /etc/passwd
Shell > awk 'BEGIN{ system("egrep \"^\root|^\nobody\" /etc/passwd") }'
root:x:0:0:root:/root:/bin/bash
nobody:x:65534:65534:Kernel Overflow User:::/sbin/nologin
```

Another example:

```
Shell > awk 'BEGIN{ if ( system("xmind > /dev/null") == 0 ) print "True"; else
print "False" }'
False
```

6. Write the output of the `awk` program to a file

```
Shell > head -n 5 /etc/passwd | awk -F ":" 'BEGIN{OFS="\t"} {print $1,$2 > "/tmp/user.txt"}'
Shell > cat /tmp/user.txt
root      x
bin       x
daemon    x
adm       x
lp        x
```

### 🔥 Tip

">" indicates writing to the file as an overlay. If you want to write to the file as an append, please use ">>". Reminder again, you should use double quotation marks to include the file path.

7. pipe character

See [here](#)

## 8. Custom functions

syntax - `function NAME(parameter list) { function body }`. Such as:

```
Shell > awk 'function mysum(a,b) {return a+b} BEGIN{print mysum(1,6)}'  
7
```

## 6.9 Concluding remarks

---

If you have specialized programming language skills, `awk` is relatively easy to learn. However, for most sysadmins with weak programming language skills (including the author), `awk` can be very complicated to learn. For information not covered, please refer to [here](#).

Thank you again for reading.

<https://docs.rockylinux.org/>