## 三剑客

### grep, Global Regular Expression Print

SYNOPSIS

grep [OPTIONS] PATTERN [FILE...]

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

grep [-abcEFGhHilLnqrsvVwxy][-A<显示列数>][-B<显示列数>][-C<显示列数>][-d<进行动作>][-e<范本样式>][-f<范本文件>][--help][范本样式][文件或目录...]

DESCRIPTION grep searches the named input FILEs (or standard input if no files are named, or if a single hyphen-minus (-) is given as file name) for lines containing a match to the given PATTERN. By default, grep prints the matching lines.

grep指令用于查找内容包含指定的范本样式的文件，如果发现某文件的内容符合所指定的范本样式，预设grep指令会把含有范本样式的那一列显示出来。若不指定任何文件名称，或是所给予的文件名为"-"，则grep指令会从标准输入设备读取数据。筛取自己想要的东西

$ seq 50|xargs -n 5 >att.txt

$ vi att.txt

$ cat att.txt

1 2 3 4 5 oldboy

6 7 8 9 10 oldgirl

11 12 13 14 15 littergirl

16 17 18 19 20 sandowboy

21 22 23 24 25 litterboy

26 27 28 29 30 aaa

31 32 33 34 35 aaa

36 37 38 39 40 magical

41 42 43 44 45 dream

46 47 48 49 50 mag

#### Context Line Control

* -A NUM, --after-context=NUM Print NUM lines of trailing context after matching lines. Places a line containing a group separator (described under --group-separator) between contiguous groups of matches.

$ grep -A2 '18' att.txt

16 17 18 19 20 sandowboy

21 22 23 24 25 litterboy

26 27 28 29 30 aaa

* -B NUM, --before-context=NUM Print NUM lines of leading context before matching lines. Places a line containing a group separator (described under --group-separator) between contiguous groups of matches.

$ grep -B2 '18' att.txt

6 7 8 9 10 oldgirl

11 12 13 14 15 littergirl

16 17 18 19 20 sandowboy

* -C NUM, -NUM, --context=NUM Print NUM lines of output context. Places a line containing a

separator (described under --group-separator) between contiguous groups of matches.

$ grep --context=1 '18' att.txt

11 12 13 14 15 littergirl

16 17 18 19 20 sandowboy

21 22 23 24 25 litterboy

* -group-separator=SEP Use SEP as a group separator. By default SEP is double hyphen (--).
* --no-group-separator Use empty string as a group separator.

#### Matcher Selection

* -E, --extended-regexp Interpret PATTERN as an extended regular expression (ERE, see below). (-E is specified by POSIX.)

$ grep -E '18|38' att.txt

16 17 18 19 20 sandowboy

36 37 38 39 40 magical

* -F, --fixed-strings, --fixed-regexp Interpret PATTERN as a list of fixed strings, separated by newlines, any of which is to be matched. (-F is specified by POSIX, --fixed-regexp is an obsoleted alias,please do not use it new scripts.)
* -G, --basic-regexp Interpret PATTERN as a basic regular expression (BRE, see below). This is the default.
* -P, --perl-regexp Interpret PATTERN as a Perl regular expression. This is highly experimental and grep -P may warn of unimplemented features.

#### Matching Control

* -e PATTERN, --regexp=PATTERN Use PATTERN as the pattern. This can be used to specify multiple search patterns, or to protect a pattern beginning with a hyphen (-). (-e is specified by POSIX.)
* -f FILE, --file=FILE Obtain patterns from FILE, one per line. The empty file contains zero patterns, and therefore matches nothing. (-f is specified by POSIX.)
* -i, --ignore-case Ignore case distinctions in both the PATTERN and the input files. (-i is specified by POSIX.)
* -v, --invert-match Invert the sense of matching, to select non-matching lines. (-v is specified by POSIX.)
* -w, --word-regexp Select only those lines containing matches that form whole words. The test is that the matching substring must either be at the beginning of the line, or preceded by a non-word constituent character. Similarly, it must be either at the end of the line or followed by a non-word constituent character. Word-constituent characters are letters, digits, and the underscore.-x, --line-regexp Select only those matches that exactly match the whole line. (-x is specified by POSIX.)Obsolete synonym for -i.

#### Output Line Prefix Control

* -n, --line-number Prefix each line of output with the 1-based line number within its input file. (-n is specified by POSIX.)

$ grep -n '18' att.txt

4:16 17 18 19 20 sandowboy

-a或--text 不要忽略二进制的数据。

-b或--byte-offset Print the 0-based byte offset within the input file before each line of output在显示符合范本样式的那一行之前，标示出该行第一个字符的**位**编号。

$ grep -b '18' att.txt

36:16 17 18 19 20

-c或--count 计算符合范本样式的列数。

$ grep -c '18' att.txt

1

$ grep -v oldboy att.txt

**-v 排除**

$ grep oldboy att.txt

只选取oldboy

$ grep -E '3|4|7' att.txt

3

4

7

$ grep -vE '1|2|4|5|7' ett.txt3

**find  /oldboy -type f -name "\*.sh" |xargs sed -i 's#oldboy#oldgirl#g'**

**find  /oldboy -type f -name "\*.sh" |xargs   cat**

**sed -i 's#oldgirl#oldboy#g' `find /oldboy -type f -name "\*.sh"`**

[root@oldboy data]# ifconfig eth0|awk -F "[ :]+" 'NR==2 {print $4}'  
10.0.0.8  
[root@oldboy data]# ifconfig eth0|sed -nr 's#^.\*dr:(.\*) Bc.\*$#\1#gp'

10.0.0.8

ifconfig eth0 | sed -nr 's#^.#dr#g

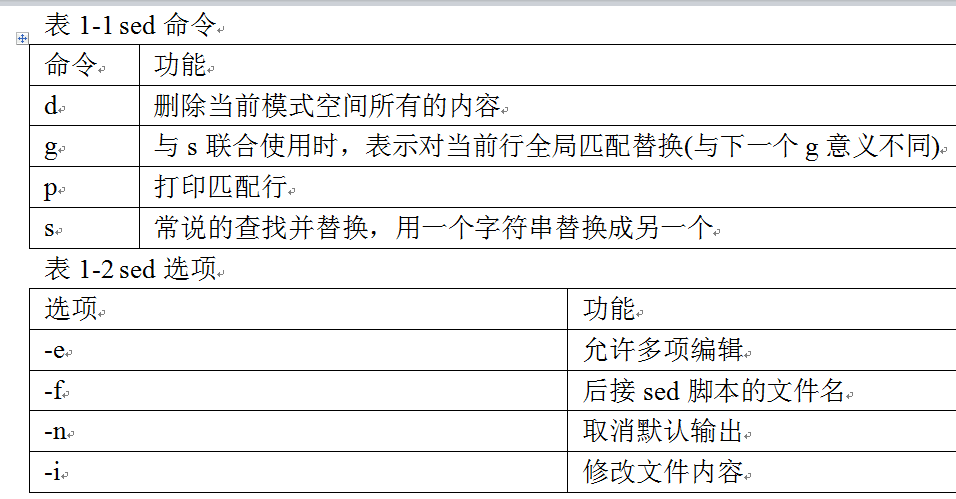
### sed

**sed  '/oldboy/d'  text.txt      %把oldboy整行删除    //之间过滤的内容 ，  d 是删除**

**替换： sed -e 's#oldboy#xubushi#g" a.txt   % -e 不改原文件  -i  修改原文件**

**具**

**sed -n '20,30p' ett.txt** %-n  取消默认输出， p 打印输出



### **gawk** - pattern scanning and processing language

SYNOPSIS

gawk [ POSIX or GNU style options ] -f program-file [ -- ] file ...

gawk [ POSIX or GNU style options ] [ -- ] program-text file ...

pgawk [ POSIX or GNU style options ] -f program-file [ -- ] file ...

pgawk [ POSIX or GNU style options ] [ -- ] program-text file ...

#### Arrays

Arrays are subscripted with an expression between square brackets ([ and ]). If the expression is an expression list (expr, expr ...) then the array subscript is a string consisting of the concatenation of the (string) value of each expression, separated by the value of the SUBSEP variable. This facility is used to simulate multiply dimensioned arrays. For example:

i = "A"; j = "B"; k = "C"

x[i, j, k] = "hello, world\n"

assigns the string "hello, world\n" to the element of the array x which is indexed by the string "A\034B\034C". All arrays in AWK are associative, i.e. indexed by string values. The special operator in may be used to test if an array has an index consisting of a particular value.

if (val in array)

print array[val]

If the array has multiple subscripts, use (i, j) in array.The in construct may also be used in a for loop to iterate over all the elements of an array. An element may be deleted from an array using the delete statement. The delete statement may also be used to delete the entire contents of an array, just by specifying the array name without a subscript.

#### Numeric Functions

AWK has the following built-in arithmetic functions:

atan2(y, x) Returns the arctangent of y/x in radians.

cos(expr) Returns the cosine of expr, which is in radians.

exp(expr) The exponential function.

int(expr) Truncates to integer.

log(expr) The natural logarithm function.

rand() Returns a random number N, between 0 and 1, such that 0 â‰¤ N < 1.

sin(expr) Returns the sine of expr, which is in radians.

sqrt(expr) The square root function.

srand([expr]) Uses expr as a new seed for the random number generator. If no expr is pro-

vided, the time of day is used. The return value is the previous seed for the

random number generator.

#### String Functions

Gawk has the following built-in string functions:

asort(s [, d]) Returns the number of elements in the source array s. The contents

of s are sorted using gawkâ™s normal rules for comparing values, and

the indices of the sorted values of s are replaced with sequential

integers starting with 1. If the optional destination array d is

specified, then s is first duplicated into d, and then d is sorted,

leaving the indices of the source array s unchanged.

asorti(s [, d]) Returns the number of elements in the source array s. The behavior

is the same as that of asort(), except that the array indices are

used for sorting, not the array values. When done, the array is

indexed numerically, and the values are those of the original

indices. The original values are lost; thus provide a second array

if you wish to preserve the original.

gensub(r, s, h [, t]) Search the target string t for matches of the regular expression r.

If h is a string beginning with g or G, then replace all matches of

r with s. Otherwise, h is a number indicating which match of r to

replace. If t is not supplied, $0 is used instead. Within the

replacement text s, the sequence \n, where n is a digit from 1 to 9,

may be used to indicate just the text that matched the nâ™th paren-

thesized subexpression. The sequence \0 represents the entire

matched text, as does the character &. Unlike sub() and gsub(), the

modified string is returned as the result of the function, and the

original target string is not changed.

gsub(r, s [, t]) For each substring matching the regular expression r in the string

t, substitute the string s, and return the number of substitutions.

If t is not supplied, use $0. An & in the replacement text is

replaced with the text that was actually matched. Use \& to get a

literal &. (This must be typed as "\\&"; see GAWK: Effective AWK

Programming for a fuller discussion of the rules for &â™s and back-

slashes in the replacement text of sub(), gsub(), and gensub().)

index(s, t) Returns the index of the string t in the string s, or 0 if t is not

present. (This implies that character indices start at one.)

length([s]) Returns the length of the string s, or the length of $0 if s is not

supplied. Starting with version 3.1.5, as a non-standard extension,

with an array argument, length() returns the number of elements in

the array.

match(s, r [, a]) Returns the position in s where the regular expression r occurs, or

0 if r is not present, and sets the values of RSTART and RLENGTH.

Note that the argument order is the same as for the ~ operator: str

~ re. If array a is provided, a is cleared and then elements 1

through n are filled with the portions of s that match the corre-

sponding parenthesized subexpression in r. The 0â™th element of a

contains the portion of s matched by the entire regular expression

r. Subscripts a[n, "start"], and a[n, "length"] provide the start-

ing index in the string and length respectively, of each matching

substring.

split(s, a [, r]) Splits the string s into the array a on the regular expression r,

and returns the number of fields. If r is omitted, FS is used

instead. The array a is cleared first. Splitting behaves identi-

cally to field splitting, described above.

sprintf(fmt, expr-list) Prints expr-list according to fmt, and returns the resulting string.

strtonum(str) Examines str, and returns its numeric value. If str begins with a

leading 0, strtonum() assumes that str is an octal number. If str

begins with a leading 0x or 0X, strtonum() assumes that str is a

hexadecimal number.

sub(r, s [, t]) Just like gsub(), but only the first matching substring is replaced.

substr(s, i [, n]) Returns the at most n-character substring of s starting at i. If n

is omitted, the rest of s is used.

tolower(str) Returns a copy of the string str, with all the upper-case characters

in str translated to their corresponding lower-case counterparts.

Non-alphabetic characters are left unchanged.

toupper(str) Returns a copy of the string str, with all the lower-case characters

in str translated to their corresponding upper-case counterparts.

Non-alphabetic characters are left unchanged.

As of version 3.1.5, gawk is multibyte aware. This means that index(), length(), substr()

and match() all work in terms of characters, not bytes.

#### Time Functions

Since one of the primary uses of AWK programs is processing log files that contain time

stamp information, gawk provides the following functions for obtaining time stamps and for-

matting them.

mktime(datespec)

Turns datespec into a time stamp of the same form as returned by systime(). The

datespec is a string of the form YYYY MM DD HH MM SS[ DST]. The contents of the

string are six or seven numbers representing respectively the full year including

century, the month from 1 to 12, the day of the month from 1 to 31, the hour of

the day from 0 to 23, the minute from 0 to 59, and the second from 0 to 60, and an

optional daylight saving flag. The values of these numbers need not be within the

ranges specified; for example, an hour of -1 means 1 hour before midnight. The

origin-zero Gregorian calendar is assumed, with year 0 preceding year 1 and year

-1 preceding year 0. The time is assumed to be in the local timezone. If the

daylight saving flag is positive, the time is assumed to be daylight saving time;

if zero, the time is assumed to be standard time; and if negative (the default),

mktime() attempts to determine whether daylight saving time is in effect for the

specified time. If datespec does not contain enough elements or if the resulting

time is out of range, mktime() returns -1.

strftime([format [, timestamp[, utc-flag]]])

Formats timestamp according to the specification in format. If utc-flag is

present and is non-zero or non-null, the result is in UTC, otherwise the result is

in local time. The timestamp should be of the same form as returned by systime().

If timestamp is missing, the current time of day is used. If format is missing, a

default format equivalent to the output of date(1) is used. See the specification

for the strftime() function in ANSI C for the format conversions that are guaran-

teed to be available.

systime() Returns the current time of day as the number of seconds since the Epoch

(1970-01-01 00:00:00 UTC on POSIX systems).

#### Bit Manipulations Functions

Starting with version 3.1 of gawk, the following bit manipulation functions are available.

They work by converting double-precision floating point values to uintmax\_t integers, doing

the operation, and then converting the result back to floating point. The functions are:

and(v1, v2) Return the bitwise AND of the values provided by v1 and v2.

compl(val) Return the bitwise complement of val.

lshift(val, count) Return the value of val, shifted left by count bits.

or(v1, v2) Return the bitwise OR of the values provided by v1 and v2.

rshift(val, count) Return the value of val, shifted right by count bits.

xor(v1, v2) Return the bitwise XOR of the values provided by v1 and v2.

Internationalization Functions

Starting with version 3.1 of gawk, the following functions may be used from within your AWK

program for translating strings at run-time. For full details, see GAWK: Effective AWK Pro-

gramming.

bindtextdomain(directory [, domain])

Specifies the directory where gawk looks for the .mo files, in case they will not or

cannot be placed in the â˜â˜standardâ™â™ locations (e.g., during testing). It returns

the directory where domain is â˜â˜bound.â™â™

The default domain is the value of TEXTDOMAIN. If directory is the null string (""),

then bindtextdomain() returns the current binding for the given domain.

dcgettext(string [, domain [, category]])

Returns the translation of string in text domain domain for locale category category.

The default value for domain is the current value of TEXTDOMAIN. The default value

for category is "LC\_MESSAGES".

If you supply a value for category, it must be a string equal to one of the known

locale categories described in GAWK: Effective AWK Programming. You must also supply

a text domain. Use TEXTDOMAIN if you want to use the current domain.

dcngettext(string1 , string2 , number [, domain [, category]])

Returns the plural form used for number of the translation of string1 and string2 in

text domain domain for locale category category. The default value for domain is the

current value of TEXTDOMAIN. The default value for category is "LC\_MESSAGES".

If you supply a value for category, it must be a string equal to one of the known

locale categories described in GAWK: Effective AWK Programming. You must also supply

a text domain. Use TEXTDOMAIN if you want to use the current domain.

USER-DEFINED FUNCTIONS

Functions in AWK are defined as follows:

function name(parameter list) { statements }

Functions are executed when they are called from within expressions in either patterns or

actions. Actual parameters supplied in the function call are used to instantiate the formal

parameters declared in the function. Arrays are passed by reference, other variables are

passed by value.

Since functions were not originally part of the AWK language, the provision for local vari-

ables is rather clumsy: They are declared as extra parameters in the parameter list. The

convention is to separate local variables from real parameters by extra spaces in the param-

eter list. For example:

function f(p, q, a, b) # a and b are local

{

...

}

/abc/ { ... ; f(1, 2) ; ... }

The left parenthesis in a function call is required to immediately follow the function name,

without any intervening white space. This avoids a syntactic ambiguity with the concatena-

tion operator. This restriction does not apply to the built-in functions listed above.

Functions may call each other and may be recursive. Function parameters used as local vari-

ables are initialized to the null string and the number zero upon function invocation.

Use return expr to return a value from a function. The return value is undefined if no

value is provided, or if the function returns by âœfalling offâ the end.

If --lint has been provided, gawk warns about calls to undefined functions at parse time,

instead of at run time. Calling an undefined function at run time is a fatal error.

The word func may be used in place of function.

#### DYNAMICALLY LOADING NEW FUNCTIONS

Beginning with version 3.1 of gawk, you can dynamically add new built-in functions to the

running gawk interpreter. The full details are beyond the scope of this manual page; see

GAWK: Effective AWK Programming for the details.

extension(object, function)

Dynamically link the shared object file named by object, and invoke function in that

object, to perform initialization. These should both be provided as strings.

Returns the value returned by function.

This function is provided and documented in GAWK: Effective AWK Programming, but everything

about this feature is likely to change eventually. We STRONGLY recommend that you do not

use this feature for anything that you arenâ™t willing to redo.

SIGNALS

pgawk accepts two signals. SIGUSR1 causes it to dump a profile and function call stack to

the profile file, which is either awkprof.out, or whatever file was named with the --profile

option. It then continues to run. SIGHUP causes pgawk to dump the profile and function

call stack and then exit.

**awk 一门语言，擅长取列，取行，过滤 （grep)**

**$ awk -F ":" '{print $1}' /etc/passward**

**%打印第一列$1 第二列 全部 $0 -F 指定分隔符 行号NR 最后一列$NF**

**$ awk -F ":" '{print $1,$2,$3}' /etc/passward**

**$ awk -F ":" 'NR==2 {print $1,$2,$3}' /etc/passward**

$ awk '19<NR && NR<31' att.txt

% NR行号，