GREP family

Σύνταξη: grep [options] pattern filename(s)

fgrep [options] string filename(s)

egrep [options] pattern filename(s)

Η εντολή grep χρησιμοποιείται για αναζήτηση μιας κανονικής έκφρασης.

Η εντολή egrep χρησιμοποιείται για αναζήτηση μιας εκτεταμένης κανονικής έκφρασης.

Η εντολή fgrep χρησιμοποιείται για αναζήτηση ενός σταθερού string.

Αν δεν οριστούν τα αρχεία, τότε κάνει αναζήτηση στην πρότυπη είσοδο (πληκτρολόγιο).

Σημαντικές επιλογές εντολών grep:

n: εμφανίζει τους αριθμούς γραμμών.

i: αγνοεί τη διάκριση μεταξύ πεζών και κεφαλαίων χαρακτήρων.

l: εμφανίζει τα ονόματα αρχείων που περιέχουν το πρότυπο.

v: εμφανίζει τις γραμμές κειμένου που δεν ταιριάζουν το πρότυπο.

w: περιορίζει την αναζήτηση σε ολόκληρες λέξεις μόνο

Παραδειγμα

grep pdpuser passwd1

Εμφανίζει όλες τις γραμμές του αρχείου passwd1 που περιέχουν το πρότυπο pdpuser.

grep -wn bash passwd1

Εμφανίζει όλες τις γραμμές του αρχείου passwd1 αριθμημένες που περιέχουν το πρότυπο bash σε ολόκληρη λέξη μόνο.

Grep και Κανονικες Εκφρασεις

Οι κανονικές εκφράσεις αποτελούνται από κανονικούς χαρακτήρες και μετα-χαρακτήρες.

Οι κανονικοί χαρακτήρες είναι ένα οποιοδήποτε χαρακτήρα εκτός τον χαρακτήρα νέας γραμμής.

Οι μετα-χαρακτήρες είναι ειδικοί χαρακτήρες που έχουν ειδική σημασία.

Οι μετα-χαρακτήρες καθορίζουν διάφορα χαρακτηριστικά των χαρακτήρων ή της αναζήτησης.

Γενικά οι μετα-χαρακτήρες Ομαδοποιούνται σε (http://regexr.com/)

Καθοριστές ομάδων χαρακτήρων (πχ (acgt))

Καθοριστές κλάσεων χαρακτήρων (πχ [A-Z})

Χαρακτήρες μπαλαντέρ (πχ *)

Άγκυρες (πχ τέλος γραμμής $)

Καθοριστές γειτνίασης

String υποκατάστασης

Χαρακτήρες αναστολής μετα-χαρακτήρα (escaped)

Η τελεία (.) ταυτίζει έναν οποιοδήποτε απλό χαρακτήρα εκτός από τον χαρακτήρα νέας γραμμής.

α.c ταυτίζει abc, adc, a&c και a;c.

u..x ταυτίζει unix, uvax και u3(x.

Ο αστερίσκος (*) ταυτίζει μηδέν ή περισσότερες εμφανίσεις του χαρακτήρα που προηγείται.

ab*c ταυτίζει ac, abc, abbc και abbbc.

.* ταυτίζει οποιοδήποτε αλφαριθμητικό.

Κλάση χαρακτήρων ([]) ταυτίζει οποιοδήποτε από τους χαρακτήρες που βρίσκονται μέσα στις αγκύλες.

Η παύλα (-) δηλώνει μια περιοχή χαρακτήρων. Π.χ. [α-e]

Ο χαρακτήρας (^) ταιριάζει κάθε χαρακτήρα που δεν περικλείεται μέσα στις αγκύλες.

Παραδείγματα:

[fF]un ταυτίζει fun και Fun.
b[aeiou]g ταυτίζει bag, beg, big, bog και bug.
[A-Z].* ταυτίζει ένα αλφαριθμητικό που αρχίζει με ένα κεφαλαίο γράμμα.
[^abc].* ταυτίζει ένα οποιοδήποτε αλφαριθμητικό που να μην αρχίζει με τους χαρακτήρες a, b ή c.

Το σύμβολο (^) ταυτίζει μόνο την αρχή της γραμμής.

^D.* ταυτίζει μια γραμμή που στην αρχή της ξεκινά με D.

Το sύμβολο δολάριο ($) ταυτίζει μόνο το τέλος της γραμμής.

.*d$ ταυτίζει μια γραμμή που τελειώνει με d.

Το σύμβολο (\) αναστέλει την σημασία κάθε μετα-χαρακτήρα.

file\.txt ταυτίζει file.txt αλλά όχι file_txt.

Regular Expressions

If there’s one thing that humans do well, it’s pattern matching. You can categorize the numbers in

the following list with barely any thought:

```
321-40-0909
302-555-8754
3-15-66
95124-0448
```

You can tell at a glance which of the following words can't possibly be valid English words by the pattern of consonants and vowels:

```
grunion vortenal pskov trebular elibm talus
```

Regular expressions are `grep`'s method of letting you look for patterns in a file:

- A fraction is a series of digits followed by a slash, followed by another series of digits.
- A valid name consists of a series of letters, a comma followed by zero or more spaces, followed by another series of letters.
- A simple condition consists of a variable name or number, followed by one of < <= > >= == or !=, followed by another variable name or number.
- A valid number consists of an optional minus sign, a run of digits, and an optional decimal point which is possibly followed by more digits.

The Simplest Patterns

The simplest pattern to look for is a word or words. If you want to see if a file `data.txt` contains the words `Joe Smith`, for example, you can use this command:

```
grep 'Joe Smith' data.txt
```

Notice the quotemarks around `Joe Smith` to prevent the shell from thinking the blank separates command options or parameters.

Matching any single character

Let's make a pattern that will match the letter `e` followed by *any character at all*, followed by the letter `t`. To say "any character at all", you use a dot. Here's the pattern:

```
grep 'e.t' data.txt
```

This will match *better*, *either*, and *best* (the dot will match the *t*, *i*, and *s* in those words). It will not match *beast* (two letters between the *e* and *t*), *ketch* (no letters between the *e* and *t*), or *crease* (no letter *t* at all!).

Matching classes of characters

Now let's find out how to narrow down the field a bit. We'd like to be able to find a pattern consisting of the letter *b*, any vowel (*a*, *e*, *i*, *o*, or *u*), followed by the letter *t*. To say "any one of a certain series of characters", you enclose them in square brackets:

```
grep 'b[aeiou]t' data.txt/
```

This matches lines with words like *bat*, *bet*, *rab**bit***, *ro**bot**ic*, and *a**but**ment*. It won't match *boot*,

because there are two letters between the *b* and *t*, and the class matches only a single character. (We'll see how to check for multiple vowels later.)

There are abbreviations for establishing a series of letters: `[a-f]` is the same as `[abcdef]`; `[A-Gm-p]` is the same as `[ABCDEFGmnop]`; `[0-9]` matches a single digit (same as `[0123456789]`).

You may also complement (negate) a class; you can look for the letter *e* followed by anything **except** a vowel, followed by the letter *t*; or any character **except** a capital letter:

```
grep 'e[^aeiou]t' data.txt
grep '[^A-Z]' data.txt
```

There are some classes that are so useful that the POSIX standard supplies quick and easy abbrevations, among them:

Abbreviation	Means
`[:digit:]`	a digit
`[:alpha:]`	Alphabetic characters
`[:space:]`	a "whitespace" character
`[:upper:]`	Uppercase letters

The square brackets are part of the abbreviation, so when you use them inside a character class specification, you will end up with two sets of brackets. Thus, this pattern matches three alphabetic characters (we'll see a better way later on)..

```
grep '[[:alpha:]][[:alpha:]][[:alpha:]]' data.txt
```

Anchors

All the patterns we've seen so far will find a match anywhere within a line, which is usually - but not always - what we want. For example, we might insist on a capital letter, but only as the very first character in the string. Or, we might say that an employee ID number has to end with a digit. Or, we might want to find the word *go* only if it is at the beginning of a word, so that we will find it in *You met another, and pfft you was **go**ne.*, but we won't mistakenly find it in *I for**go**t my umbrella*. This is the purpose of an anchor; to make sure that we are at a certain boundary before we continue the match. Unlike character classes, which match individual characters in a string, these anchors do not match any character; they simply establish that we are on the correct boundaries.

The up-arrow `^` matches the beginning of a line, and the dollar sign `$` matches the end of a line. Thus, `^[A-Z]` matches a capital letter at the beginning of the line. Note that if we put the `^` *inside* the square brackets, that would mean something entirely different!

A pattern `[0-9]$` matches a digit at the end of a line. These are the boundaries you will use most often.

The other anchor is **\b**, which stands for a "word boundary". For example, if we want to find the word *met* at the beginning of a word, we write the pattern **`'\bmet'`**, which will match *The metal plate* and *The metropolitan lifestyle*, but not *Wear your bike helmet*. The pattern **`'ing\b'`** will

match *Hiking is fun* and *Reading, writing, and arithmetic*, but not *Gold ingots are heavy*. Finally,the pattern **`'\bhat\b'`** matches only the *The hat is red* but not *That is the question* or *she hates anchovies* or *the shattered glass*.

Repetition

All of these classes match only one character; what if we want to match three digits in a row, or an arbitrary number of vowels? You can follow any class or character by a repetition count. (From here on, we will leave off the quote marks around the patterns. When you put them into your `grep` command, you should put quotes around the pattern.)

Pattern	Matches
`b[aeiou]\{2\}t`	`b` followed by two vowels, followed by `t`
`[[:alpha:]]\{3\}`	Three alphabetic characters
`A[0-9]\{3,\}`	The letter `A` followed by 3 or more digits
`[A-Z]\{0,5\}`	Zero to five capital letters
`[[:alpha:]]\{3,7\}`	Three to seven word characters

Notice that you need a `\` (backslash) before the beginning and ending braces when using `grep`. When using `egrep` or `grep -E`, you do not need the backslashes.

This lets us write our social security number pattern match as **`\[0-9]\{3\}-[0-9]\{2\}-[0-9]\{4\}/`**.

There are three repetitions that are so common that Perl has special symbols for them: `*` means "zero or more," `\+` means "one or more," and `\?` means "zero or one". Thus, if you want to look for lines consisting of last names followed by a first initial, you'd use this pattern:

```
^[A-Za-z]\+,[[:space:]]*[A-Z]$
```

This matches, starting at the beginning of the line, a word of one or more alphabetic characters followed by an optional comma, zero or more spaces, and a single capital letter, which must be at the end of the line.

Note: In `egrep`, you do not need the backslash before the plus sign or question mark.

Grouping

So far so good, but what if we want to scan for a last name, followed by an optional comma-whitespace-initial; thus matching only a last name like "Smith" or a full "Smith, J"? We need to put the comma, whitespace, and initial into a unit with parentheses, preceded by backslashes in `grep` but not in `egrep`:

```
^[A-Za-z]\+\(,[[:space:]]*[A-Z]\)\?$/
```

There's a side effect of grouping - whenever we use parentheses to group something, the match operation stores the matched area in a buffer which we can access later on in the match. For example, let's say you want to find all lines with repeated words on them. You type this:

```
\([A-Za-z]\+\)[[:space:]]\1
```

This says to look for:

- `\([A-Za-z]\+\)` one or more letters, and remember it in buffer 1
- `[[:space:]]` one whitespace character
- `\1` whatever was in buffer 1

Thus, this will find lines with repeated words like:

```
Paris in the the spring.
This is very very important.
```

You can see exactly what `grep` matched by using the `--only-matching` option. Try putting this in a file named `data.txt`

```
No duplicate words here.
This does not have too too much on it.
Paris in the the spring.
A sentence with all different words.
I sang it again and again.
```

Run these commands to see what the `--only-matching` does.

```
grep '\([A-Za-z]\+\)[[:space:]]\1' data.txt
grep --only-matching '\([A-Za-z]\+\)[[:space:]]\1' data.txt
grep --color '\([A-Za-z]\+\)[[:space:]]\1' data.txt
```

Παραδείγματα

grep '.n' textfile

Εμφανίζει όλες τις γραμμές του αρχείου textfile που ταυτίζουν ένα οποιοδήποτε χαρακτήρα ακολουθούμενο από τον χαρακτήρα n.

grep 'bio*' textfile

Εμφανίζει όλες τις γραμμές του αρχείου textfile που περιέχουν το πρότυπο bio ακολοθούμενο από μια οποιαδήποτε ακολουθία χαρακτήρων.

grep '199[1-5]' textfile

Εμφανίζει όλες τις γραμμές του αρχείου textfile που περιέχουν τα έτη 1991 έως 1995.

grep '^[a-z]' textfile

Εμφανίζει όλες τις γραμμές του αρχείου textfile που ξεκινούν με έναν οποιοδήποτε πεζό αγγλικό χαρακτήρα.

grep '22$' textfile

Εμφανίζει όλες τις γραμμές του αρχείου textfile που τελειώνουν σε 22.

grep '\$1\.20' textfile

Εμφανίζει όλες τις γραμμές του αρχείου textfile που περιέχουν την τιμή $1.20.

grep '^[A-Z]' textfile

Εμφανίζει όλες τις γραμμές του αρχείου textfile που ξεκινούν με ένα κεφαλαίο αγγλικό χαρακτήρα.

grep '^..$' textfile

Εμφανίζει όλες τις γραμμές του αρχείου textfile που περιέχουν 2 χαρακτήρες.

grep '^[^0-9]*$' textfile

Εμφανίζει όλες τις γραμμές του αρχείου textfile που δεν περιέχουν αριθμούς.

grep '[0-9]*[.][0-9]*' textfile

Εμφανίζει όλες τις γραμμές του αρχείου textfile που περιέχουν πραγματικούς αριθμούς.

grep '1*[012]*[1-9]*:[0-5][0-9]' textfile

Εμφανίζει όλες τις γραμμές του αρχείου textfile που περιέχουν ώρα σε 12ωρη μορφή.

Επεκτασεις του egrep

Το σύμβολο (+) ταυτίζει μια ή περισσότερες εμφανίσεις του χαρακτήρα που προηγείται.

ab+c ταυτίζει abc, abbc, abbbc άλλα όχι ac.

Το ερωτηματικό (?) ταυτίζει μηδέν ή μια εμφάνιση του χαρακτήρα που προηγείται.

ab?c ταυτίζει ac ή abc αλλά όχι abbc.

Το λογικό Η' (|) ταυτίζει μια από τις δύο κανονικές εκφράσεις.

abc|def ταυτίζει abc ή def.

Οι παρενθέσεις ( ) τοποθετούνται στην κανονική έκφραση ώστε οι μετα-χαρακτήρες *, + ή ?, να δράσουν στο σύνολο της έκφρασης αντί μόνο σε χαρακτήρα.

a(bc)* ταυτίζει a, abc, abcbc, abcbcbc.

(foot|base)ball ταυτίζει football ή baseball.

Οι αγκύλες { } δηλώνουν τον αριθμό των επαναλήψεων που μπορεί να επαναληφθεί η κανονική έκφραση.

[a-z]{3} ταιριάζει τρεις πεζούς χαρακτήρες.

m.{2,4} ταιριάζει αλφαριθμητικά που ξεκινούν με m και ακολουθούν από 2 μέχρι και 4 χαρακτήρες.

Παραδείγματα

egrep '[A-Z][A-Z]+' textfile

Εμφανίζει όλες τις γραμμές του αρχείου textfile που περιέχουν μια οποιαδήποτε ακολουθία μόνο κεφαλαίων αγγλικών χαρακτήρων.

egrep '[α-z]?' textfile

Εμφανίζει όλες τις γραμμές του αρχείου textfile που περιέχουν ένα ή κανένα πεζό αγγλικό χαρακτήρα.

egrep '239(4|6)-(0|1)' textfile

Εμφανίζει όλες τις γραμμές του αρχείου textfile που περιέχουν τηλέφωνα που το 4 ψηφίο είναι 4 ή 6 και το 6 ψηφίο είναι 0 ή 1.

egrep '[0-9]{2}' textfile

Εμφανίζει όλες τις γραμμές του αρχείου textfile που περιέχουν αριθμούς με 2 ψηφία.

egrep '[\!*]' textfile

Εμφανίζει όλες τις γραμμές του αρχείου textfile που περιέχουν τους χαρακτήρες ! και *.

Yet anoteher grep exercise

Obtain the file grepdata.txt.

Once you have the file, write a series of grep statements that do the following:

- Print all lines that contain a phone number with an extension (the letter x or X followed by four digits).
- Print all lines that begin with three digits followed by a blank. Your answer *must* use the \{ and \} repetition specifier.
- Print all lines that contain a date. Hint: this is a *very* simple pattern. It does not have to work for any year before 2000.
- Print all lines containing a vowel (a, e, i, o, or u) followed by a single character followed by the same vowel again. Thus, it will find “eve” or “adam” but not “vera”. Hint: \(and \)
- Print all lines that do not begin with a capital S.

Write grep statements that use command-line options along with the pattern to do the following:

- Print all lines that contain CA in either uppercase or lowercase.
- Print all lines that contain an email address (they have an @ in them), preceded by the line number.
- Print all lines that do *not* contain the word Sep. (including the period).
- Print all lines that contain the word de as a whole word.