Expressions regulars

PID_00270620

Guillem Lluch Moll

Temps mínim de dedicació recomanat: 3 hores

Universitat Oberta de Catalunya

Guillem Lluch Moll

Llicenciat en Matemàtiques per la Universitat Autònoma de Barcelona (UAB), màster en Programari Lliure per la Universitat Oberta de Catalunya (UOC) i màster en Llenguatges i Sistemes Informàtics (UNED). De llarga trajectòria docent, en diferents nivells i assignatures, actualment treballa com a coordinador TIC en un institut de secundària.

L'encàrrec i la creació d'aquest recurs d'aprenentatge UOC han estat coordinats pel professor: Julià Minguillón Alfonso (2020)

Primera edició: febrer 2020

Av. Tibidabo, 39-43, 08035 Barcelona
Realització editorial: FUOC

Índex

1. Introducció

Durant el tractament de les diverses dades sorgeixen moltes situacions en què cal trobar, comprovar o substituir fragments que tenen una certa regularitat en la seva representació. Pensem, per exemple, en situacions com ara:

- **Nom de fitxers similars**: compra1, compra2, ..., compra18, etc.
- **Identificadors**: com el DNI, format per vuit dígits i una lletra. A vegades la lletra està separada per un guió, a vegades no hi ha cap separació.
- **Números de telèfon**: segueixen un patró fàcil de reconèixer per a les persones. Però el programari s'ha d'afrontar a les diferents formes d'expressarlos, com ara en grups de tres dígits separats per espai o guió, o per un grup inicial de tres i llavors de dos, etc. A més, s'hi pot incloure el prefix del país.
- **Adreces postals**: formades pel carrer, el número, el codi postal, la ciutat i el país.
- **Dates**: les diferències poden ser molt grans. Els EUA empren un ordre diferent que el castellà o català. A més, hi ha detalls que compliquen el seu tractament automàtic, com ara la separació, típicament amb guions o amb barres, si es posa l'any complert o només les dues últimes xifres, etc.
- **Tractament de fitxers estructurats**: com csv, tsv, o fins i tot, json o xml.
- Etc.

La manipulació d'aquests tipus de fragments, teòricament, es podria fer emprant una programació de propòsit general, però de forma molt complexa (depenent del que cerquem). Així, el que s'utilitza per aquests tipus de problemes són les **expressions regulars** o **RegEx**.

Per a entendre-les, tornem a l'exemple del nom dels fitxers: compra1, compra2, ..., compra18, etc. Informalment, podríem referir-nos a aquests com «compraX» o «compraN». La idea que hi ha al darrera dels RegEx és formalitzar i millorar aquest procés de generalització. Fixem-nos en la «X» o la «N» de l'expressió «compraX» o «compraN». Aquestes lletres perden el seu significat habitual i passen a representar, en aquest cas, un nombre natural.

Això és el nucli de les expressions regulars: fer servir uns determinats símbols per a representar un **conjunt de caràcters** o per a senyalar la **posició** en la frase o per a denotar el **nombre d'aparicions** d'un o més caràcters.

Aquests caràcters especials són triats de manera que siguin uns símbols poc emprats i, conseqüentment, converteixen les expressions regulars en una seqüència de caràcters estranys, almenys al principi del seu aprenentatge.

2. La sintaxi

Les expressions regulars són una seqüència de símbols que o representen un conjunt de caràcters o fan una funció especial. Per exemple, «.» representa qualsevol caràcter, com ara un «9», una «n», etc. En canvi, el símbol `$` s'empra per a identificar el final de línia.

Desafortunadament, no tots els programes utilitzen el mateix joc de símbols per a les expressions regulars, encara que sí que utilitzen una sintaxi i una forma de construcció similars. Nosaltres, en aquest mòdul, emprarem els estàndards propis de **POSIX**, concretament **BRE**[1] amb Grep i Sed. POSIX també defineix **ERE**[2] molt similar i que és el que empren Bash (a partir de la versió 3) i AWK.

(1)Acrònim de *Basic Regular Expressions*.

(2)Acrònim de *Extended Regular Expressions*.

Famílies de RegEx

Cal tenir ben present, especialment quan naveguem per internet, que hi ha altres famílies de RegEx, com ara les de JavaScript (molt útils per a validar formularis a la web) o les de Perl que no sempre funcionen com a RegEx POSIX.

2.1. Comandes Grep

Per a provar les pròximes expressions ho farem amb Grep.

La comanda `grep` mostra només aquelles línies que compleixen el patró RegEx subministrat i n'amaga la resta.

Anirem il·lustrant les explicacions amb el fitxer de text pla, «deudas.txt», amb el contingut següent (vigileu els espais):

```
cat deudas.txt

amic, 25, euros
Carina, -, eurooos
Juan López, 3, dolars
Juan Pérz, 34.8, dolar
Amorós López, 110.000, euros
Lourdes,-,dolars
```

Posant `grep d deudas.txt` veureu que surten les línies 3a., 4a. i l'última, on hi ha una «d». Si poseu `grep j deudas.txt`, veureu que no surt cap línia ja que les j's que hi ha són majúscules i es diferencien les minúscules de les majúscules.

Diferencia minúscules de majúscules

En anglès, aquest comportament es diu *case sentive*.

```
# Cerca línies amb la lletra d.
grep d deudas.txt

Juan López, 3, dolars
Juan Pérz, 34.8, dolar
Lourdes,-,dolars
```

```
# Cerca línies amb la lletra j. Com que no n'hi cap, no retorna res.
grep j deudas.txt
```

Les comandes `grep` també són molt usades per a filtrar els resultats d'altres comandes, emprant *pipe* (`|`). Per exemple, `ls -ltr` llista els fitxers d'un directori. Si la sortida la redirigim a `grep` podrem aconseguir veure només els noms del fitxers que coincideixen amb el patró subministrat:

```
# Filtra la sortida de ls, mostrant només els fitxers tipus csv.
ls -ltr | grep .csv$

-rw-rw-r-- 1 gandalf comunidad 1236531 de ma 26 14:10 IncomeLeft.csv
-rw-rw-r-- 1 gandalf comunidad 3922 de ju 21 19:35 comp.csv
```

En aquest tema, provarem els patrons tant sobre un fitxer amb Grep patró fitxer com redirigint la sortida de la comanda `echo`.

```
# Quan el text de l'echo coincideix amb el patró, es mostra la línia.
echo "Martín" | grep Mar

Martín
```

```
# Si el text de l'echo no coincideix, no es mostra res.
echo "Martín" | grep " "
```

2.2. Literals i la seqüència d'escapament

Si volem trobar un determinat caràcter exactament o una seqüència de caràcters, en la majoria de casos, hem d'escriure la seqüència directament:

```
# Busca línies amb els caràcters ar (seguits).
grep ar deudas.txt

Carina, -, eurooos
Juan López, 3, dolars
Juan Pérz, 34.8, dolar
Lourdes,-,dolars
```

```
# Busca línies amb almenys un espai en blanc.
grep " " deudas.txt

amic, 25, euros
Carina, -, eurooos
Juan López, 3, dolars
Juan Pérz, 34.8, dolar
Amorós López, 110.000, euros
```

```
# Busca línies amb un nom concret.
grep "Juan López" deudas.txt

Juan López, 3, dolars
```

Atès que el símbol «.» serveix per representar qualsevol caràcter, cal fer servir algun tipus de truc quan el que volem trobar és el caràcter «.» literalment. Per a canviar el significat per defecte d'un símbol s'empra una **seqüència d'escapament**, compartida amb molts llenguatges de programació, que és la barra invertida (`\`). Així:

```
# Busca línies amb un punt.
grep "\." deudas.txt

Juan Pérz, 34.8, dolar
Amorós López, 110.000, euros
```

2.3. Classes de caràcters

Les **classes** representa un grup de caràcters. Les classes POSIX es denoten posant un text que les defineix entre dos punts i entre dos claudàtors. Per exemple `[[:digit:]]`, que representa qualsevol dígit o `[[:upper:]]`, que representa qualsevol majúscula:

```
# Cerca línies amb algun dígit.
grep [[:digit:]] deudas.txt

amic, 25, euros
Juan López, 3, dolars
Juan Pérz, 34.8, dolar
Amorós López, 110.000, euros
```

```
# Cerca línies amb alguna majúscula.
grep [[:upper:]] deudas.txt

Carina, -, eurooos
Juan López, 3, dolars
Juan Pérz, 34.8, dolar
Amorós López, 110.000, euros
Lourdes,-,dolars
```

Òbviament, els literals i les classes es poden (i se solen) combinar per a obtenir patrons més precisos:

```
# Línies amb una majúscula seguida d'una o accentuada.
grep "[[:upper:]]ó" deudas.txt

Juan López, 3, dolars
Amorós López, 110.000, euros
```

```
# Línies amb un espai en blanc i amb un dígit.
grep " [[:digit:]]" deudas.txt

amic, 25, euros
Juan López, 3, dolars
Juan Pérz, 34.8, dolar
Amorós López, 110.000, euros
```

Les classes de caràcters poden ser diferents segons el *locale* actual. Per tant, és important definir-lo en consonància amb les dades i textos que volem tractar.

Més informació

Per a més informació, vegeu el manual de *locale* a Linux.

2.4. Quantificadors

Només amb els literals i les classes estaríem molt limitats respecte a les possibilitats de precisar la cerca. Imagineu que volem que les línies tinguin dos o més «o»'s seguits. En el cas de voler-ne dos exactament, el que es podria pensar fer és:

```
# Intent d'otenir les línies amb exactament 2 «o»'s.
grep "oo" deudas.txt

Carina, -, eurooos
```

El resultat però inclou la cadena amb 3 «o»'s. Això és a causa que amb POSIX **es retornen els fragments més grans** que compleixen el patró. Si som estrictes, on hi ha tres «o»'s, realment també n'hi ha dues.

Comportament avariciós

En anglès, a aquest comportament li diuen *greedy*.

Si el que volem és poder fer cerques que puguin contenir valors amb un nombre concret de caràcters, dígits o símbols, resulta evident que aquesta forma de procedir, repetint el símbol les vegades que calgui, **no és escalable**. Per això, les expressions regulars permeten emprar **quantificadors** que regulen el nombre de vegades que pot aparèixer un caràcter o un conjunt de caràcters. Són molt comuns:

Quantificadors	Descripció
`\?`	Per un element opcional.
`\+`	Per a denotar que pot aparèixer una o més vegades.
`\*`	Per cap o moltes vegades.

Si cal més precisió hi ha `\{n}`, `\{n,m}`, `\{,m}` o `\{n,}` on *n* és el mínim nombre d'ocurrència i *m* el màxim. Per exemple:

```
# Línies que contenen una e, tres caràcters qualssevol i una s.
grep "e.\{3\}s" deudas.txt

amic, 25, euros
Amorós López, 110.000, euros
```

```
# Línies que contenen una e i, més endavant, una s.
grep "e.\+s" deudas.txt

amic, 25, euros
Carina, -, eurooos
Juan López, 3, dolars
Amorós López, 110.000, euros
Lourdes,-,dolars
```

Així i tot, si intentem tornar a reescriure l'obtenció d'exactament dues «o»'s, ens trobem que seguim sense aconseguir-ho.

```
# Intent d'obtenir les línies amb exactament 2 «o»'s.
grep "o\{2\}" deudas.txt

Carina, -, eurooos
```

Podria semblar impossible, però es pot fer, amb una mica d'enginy i amb la negació, que veurem més endavant.

```
# Línies que contenen un dígit, un punt i un dígit.
grep "[[:digit:]]\.[[:digit:]]" deudas.txt

Juan Pérz, 34.8, dolar
Amorós López, 110.000, euros
```

```
# Línies que contenen almenys un dígit, un punt i un o més dígits.
grep "[[:digit:]]\+\.[[:digit:]]\+" deudas.txt

Juan Pérz, 34.8, dolar
```

```
Amorós López, 110.000, euros
```

Aquests dos últims exemples són molt similars. Per als fragments que coincideixen amb la primera expressió són «4.8» i «0.0» i amb la segona «34.8» i «110.000». En principi, pot semblar millor la segona expressió, però això depèn de la intenció i ús que en vulguem fer.

```
# Línies que contenen almenys dos dígit seguits.
grep "[[:digit:]]\{2\}" deudas.txt

amic, 25, euros
Juan Pérz, 34.8, dolar
Amorós López, 110.000, euros
```

2.5. Delimitadors (*anchors*)

Un altre element molt utilitzat en les expressions regulars són el **delimitadors**, amb els quals podem cercar patrons que estiguin en una **determinada posició**.

Així, hi ha el de final i principi de línia `$` i `^` respectivament.

Per exemple:

```
# Línies que comencen per minúscula.
grep "^[[:lower:]]" deudas.txt

amic, 25, euros
```

```
# Línies que acaben en os.
grep "os$" deudas.txt

amic, 25, euros
Carina, -, eurooos
```

```
# Línies que acaben en os o amb os i un espai.
grep "os \?$" deudas.txt

amic, 25, euros
Carina, -, eurooos
Amorós López, 110.000, euros
```

Cal fixar-se en la diferència dels dos últims exemples. Inapreciablement, la línia d'Amorós López conté un espai en blanc al final i això fa que no compleixi amb el patró `"os$"`.

Els espais en blanc i, en general, els caràcters no visibles poden suposar grans problemes si no es consideren a temps o passen desapercebuts.

2.6. Caràcters optatius, rangs i negació

Els **caràcters optatius** i els **rangs** ofereixen molta flexibilitat a l'hora de construir les expressions regulars.

Ambdós es denoten en claudàtors (`[]`). Vegem-ne un quants exemples:

```
# Línies que comencen per J o L.
grep "^[JL]" deudas.txt

Juan López, 3, dolars
Juan Pérz, 34.8, dolar
Lourdes,-,dolars
```

```
# Línies que contenen una e, fins i tot, accentuada.
grep "[eèé]" deudas.txt

amic, 25, euros
Carina, -, eurooos
Juan López, 3, dolars
Juan Pérz, 34.8, dolar
Amorós López, 110.000, euros
Lourdes,-,dolars
```

L'altra possibilitat que hi ha és la de fer servir **rangs**. Així:

Rangs	Descripció
`[a-z]`	Rang de totes les minúscules.
`[c-n]`	Rang de totes les minúscules des de la c fins a l'n.
`[a-zA-Z]`	Podem barrejar opcions i rangs. Encaixa amb minúscules i majúscules.

```
# Línies que comencen per una majúscula entre A i D.
grep "^[A-D]" deudas.txt

Carina, -, eurooos
Amorós López, 110.000, euros
```

El manual de Grep assegura que els rangs tenen en compte les particularitats del joc de caràcters emprats.

És a dir, si tenim els *locale* de Linux configurats d'acord amb les dades, es tindran en compte les particularitats de la llengua en ús. Així i tot, en cas que els rangs no es comportin com volem, hi ha les classes, que ja hem vist, o podem emprar la «força bruta» i **escriure tots els caràcters possibles un a un**, i així tenir independència del *locale* (però no de la codificació).

Vocal minúscula en castellà

Per exemple, si volem assegurar que detectem qualsevol vocal minúscula en castellà hem d'emprar `[aáeéiíoóuú]`.

Una altra casuística que ens pot ocorre és **cercar la negació d'un patró**. La negació es denota amb ^ com a primer element dins dels claudàtors. No s'ha de confondre amb ^ que denota el principi de línia i que va fora dels claudàtors.

```
# Línies que no comencen per una majúscula entre A i D.
grep "^[^A-D]" deudas.txt
# Cal no confondre's amb ^. El primer, fora dels claudàtors, és el de principi de línia.
# El ^ de dins és la negació.

amic, 25, euros
Juan López, 3, dolars
Juan Pérz, 34.8, dolar
Lourdes,-,dolars
```

```
# Línies que no contenen la J ni la C (incorrecte).
grep "[^JC]" deudas.txt
# El que hem de fer realment són línies que tenen algun caràcter que no sigui ni J ni C.

amic, 25, euros
Carina, -, eurooos
Juan López, 3, dolars
Juan Pérz, 34.8, dolar
Amorós López, 110.000, euros
Lourdes,-,dolars
```

Una vegada més, cal recordar que Grep funciona *Greedy* i, per tant, en totes les línies anteriors hi ha caràcters que no són J ni C. Observeu:

```
# Mostrem si hi ha algun caràcter diferent de J i de C.
echo "J" | grep "[^JC]"

echo "V" | grep "[^JC]"

V

echo "JC" | grep "[^JC]"

echo "JCV" | grep "[^JC]"

JCV

echo "JJJJJJCCCC" | grep "[^JC]"
```

Amb la negació, podem ajustar per buscar exactament un número concret de vegades:

```
# Obtenir les línies amb exactament 2 «o»'s.
# Per a fer-ho, demanem que no hi hagi una o ni abans ni després.
echo "koole" | grep "[^o]oo[^o]"

koole
```

Cal pensar en el patró anterior `"[^o]oo[^o]"`. Es pot assegurar que cobrirà totes les situacions en què puguin aparèixer dues «o» seguides? Abans de continuar llegint, convé pensar-hi.

Fixeu-vos amb les situacions següents que no encaixen amb el patró anterior, analitzeu-les i feu un hipòtesi del que està fallant.

```
echo "noom" | grep "[^o]oo[^o]"

noom
```

```
echo "oom" | grep "[^o]oo[^o]"

echo "noo" | grep "[^o]oo[^o]"

echo "oo" | grep "[^o]oo[^o]"
```

Realment, el patró troba dues «o» seguides sempre que no estiguin a principi o final de línia.

2.7. Alternatives

Per a poder acabar el patró de les dues «o»s consecutives, podem emprar les alternatives. Les alternatives serveixen per a trobar les ocurrències de dues o més expressions regulars. Les diferents expressions regulars se separen per `|`.

Qualsevol línia que coincideixi amb qualssevol dels patrons, serà retornada.

Ho veurem primer amb un exemple simple i després amb les dues «o»s.

```
# Línies que comencen per A o acaben per r i espai.
grep "^A\|r $" deudas.txt

Juan Pérz, 34.8, dolar
Amorós López, 110.000, euros
```

```
# Línies que tenen un espai i guió o un espai i 3.
grep "-\| 3" deudas.txt

Carina, -, eurooos
Juan López, 3, dolars
Juan Pérz, 34.8, dolar
```

Tornem així al patró de les dues «o»s. El que hem de fer és crear un patró per quan hi ha les oo al principi, al final, o al principi i al final i ajuntar-ho amb l'expressió regular que ja tenim `[^o]oo[^o]`.

```
# oo al principi.
echo "noom" | grep "^oo[^o]"
echo "oom" | grep "^oo[^o]"
echo "noo" | grep "^oo[^o]"
echo "oo" | grep "^oo[^o]"

oom
```

```
# oo al final.
echo "noom" | grep "[^o]oo$"
echo "oom" | grep "[^o]oo$"
echo "noo" | grep "[^o]oo$"
echo "oo" | grep "[^o]oo$"

noo
```

```
# oo al principi i al final.
echo "noom" | grep "^oo$"
echo "oom" | grep "^oo$"
echo "noo" | grep "^oo$"
echo "oo" | grep "^oo$"
```

```
oo
```

```
Ara, per a tenir el patró, hem d'unir-les totes:
# Obtenir les línies amb exactament 2 «o»'s.
# Per a fer-ho, demanem que no hi hagi una o ni abans ni després.
echo "noom" | grep "^oo[^o]\|[^o]oo$\|^oo$\|[^o]oo[^o]"
echo "oom" | grep "^oo[^o]\|[^o]oo$\|^oo$\|[^o]oo[^o]"
echo "noo" | grep "^oo[^o]\|[^o]oo$\|^oo$\|[^o]oo[^o]"
echo "oo" | grep "^oo[^o]\|[^o]oo$\|^oo$\|[^o]oo[^o]"

noom
oom
noo
oo
```

2.8. Agrupaments i referència

De la forma que hem emprat els quantificadors, només podem assignar repeticions a un caràcter, un grup d'opcions o un rang. Però també hi ha situacions en què cal comptar o fer altres operacions amb un conjunt arbitrari de símbols. En aquests casos, per a crear grups, s'empren els parèntesis (amb el símbol d'escapament).

```
# Cerquem dos oh repetits.
echo "ohohaahjijiooohf" | grep "\(oh\)\{2\}"

ohohaahjijiooohf

# Cerquem tres oh repetits.
echo "ohohaahjijiooohf" | grep "\(oh\)\{3\}"
```

Les **agrupacions** tenen un paper molt important en les substitucions. En l'editor Sed podem cercar un patró i manipular tot o una part del que s'ha trobat per a crear una nova expressió.

Vegeu també

Vegeu l'apartat «L'editor Sed.»

2.9. Tipus d'expressions regulars POSIX

Hi ha dos tipus d'expressions regulars POSIX: **BRE** (*basic regular expressions*) i **ERE** (*extended regular expressions*). En «sed, a stream editor», es pot veure que les diferències es donen en els símbols `?`, `+`, `{.}`, `|`, encara que també caldria esmentar `(,)`. Tots aquests símbols, amb BRE, tenen un significat literal, excepte quan van precedits pel caràcter d'escapament. Amb ERE passa al revés, i els símbols esmentats són especials, llevat que vagin precedits pel caràcter d'escapament. Pot ser una mica confús, ja que els humans tenim una alta capacitat per arribar a acords... i també per mantenir discrepàncies.

Aquí tenim un llistat de les **classes POSIX**:

Classe POSIX	Similar a	Significat
`[:upper:]`	`[A-Z]`	Majúscules
`[:lower:]`	`[a-z]`	Minúscules

Classe POSIX	Similar a	Significat
`[:alpha:]`	`[A-Za-z]`	Majúscules i minúscules
`[:digit:]`	`[0-9]`	Dígits
`[:xdigit:]`	`[0-9A-Fa-f]`	Dígits en hexadecimal
`[:alnum:]`	`[A-Za-z0-9]`	Dígits, majúscules i minúscules
`[:punct:]`		Puntuació (caràcters imprimibles, excepte nombres i lletres)
`[:blank:]`	`[ \t]`	Espais i tabuladors
`[:space:]`	`[ \t\n\r\f\v]`	Espai tabulador, final de línia, etc.
`[:cntrl:]`		Caràcters de control
`[:graph:]`	`[^ [:cntrl:]]`	Tots els caràcters imprimibles
`[:print:]`	`[[:graph] ]`	Caràcters gràfics i espais

I a continuació, els caràcters especials o **metacaràcters**:

Metacaràcter	Descripció
`.`	Qualsevol caràcter individual. Això sí, si és dins una expressió amb claudàtors, el punt coincideix amb el punt literal. Per exemple, `a.c` coincideix amb «aac», «abc», «acc», etc., mentre `[a.c]` coincideix amb qualsevol conjunt de caràcters que inclogui un o més dels caràcters «a», «.» o «c».
`[ ]`	Una expressió entre claudàtors coincideix amb un sol caràcter que està dins dels claudàtors. Per exemple, `[abc]` coincideix amb «a», «b» o «c» i `[a-z]` especifica un interval que coincideix amb qualsevol lletra minúscula d'«a» a «z». Aquestes formes es poden barrejar: `[abcx-z]` coincideix amb «a», «b», «c», «x», «y» o «z», igual que `[a-cx-z]`. El caràcter – es tracta com a caràcter literal si és el darrer o el primer caràcter entre els claudàtors: `[abc-]`, `[-abc]`. El caràcter `]` es pot incloure en una expressió entre claudàtors si és el primer caràcter: `[]abc]`. L'expressió entre claudàtors també pot contenir classes.
`[^ ]`	Coincideix amb un caràcter que no està dins dels claudàtors. Per exemple, `[^abc]` coincideix amb qualsevol caràcter que no sigui ni «a», ni «b» ni «c» i `[^a-z]` coincideix amb qualsevol caràcter que no sigui una lletra minúscula d'«a» a «z». Aquestes formes es poden barrejar: `[^abcx-z]` coincideix amb qualsevol caràcter que no sigui ni «a», ni «b», ni «c», ni «x», ni «y» ni «z». El caràcter – es tracta com a caràcter literal si és l'últim caràcter o el primer caràcter després de `^`: `[^abc-]`, `[^-abc]`. El caràcter `]` es tracta com a caràcter literal si és el primer caràcter després de `^`: `[^]abc]`. L'expressió també pot contenir classes.
`^`	Coincideix amb la posició inicial dins de la cadena, si és el primer caràcter de l'expressió regular.
`$`	Coincideix amb la posició final de la cadena, si és l'últim caràcter de l'expressió regular.
BRE: `\?` ERE: `?`	Converteix l'element anterior en optatiu. Per exemple, `a\?` coincideix amb «» o «a».
`*`	Coincideix amb l'element anterior o amb el buit. Per exemple, `ab*c` coincideix amb «ac», «abc», «abbbc», etc. `[xyz]*` coincideix amb «», «x», «y», «z», «zx», «zyx», «xyzzy», i així successivament.
BRE: `\{m\}` ERE: `{m}`	Coincideix exactament m vegades amb l'ítem anterior. Per exemple, `a\{3\}` coincideix amb «aaa».
BRE: `\{m,\}` ERE: `{m,}`	Coincideix amb l'element anterior almenys m vegades. Per exemple, `a\{3,\}` coincideix amb «aaa», «aaaa», «aaaaa», «aaaaaa», «aaaaaaa», etc.
BRE: `\{m,n\}` ERE: `{m,n}`	Coincideix amb l'element anterior com a mínim m vegades i no més d'n vegades. Per exemple, `a\{3,5\}` coincideix amb «aaa», «aaaa» i «aaaaa».

Metacaràcter	Descripció
BRE: `\( \)` ERE: `( )`	Defineix una **subexpressió**. Es tracta com si fos un sol element. Per exemple, `ab*` coincideix amb «a», «ab», «abb» i així successivament, mentre que `(ab*)` coincideix «», «ab», «abab», «ababab», etc. La cadena corresponent dins dels parèntesis es pot recuperar (amb `\n`). Una subexpressió també s'anomena *subexpressió marcada*, un *bloc* o un *grup de captura*.
només a BRE: `\n`	Recupera el fragment entre parèntesi en lloc d'n, on n és un dígit d'1 a 9. No s'ha adoptat a la sintaxi de POSIX ERE.
només a BRE: `&`	Recuperar tota l'expressió marcada. No s'ha adoptat a la sintaxi de POSIX ERE.

Ja hem vist com provar les expressions regulars BRE amb Grep. Com amb totes les comandes, convé consultar el manual de Grep, en aquest cas, per a explorar les possibilitats que ofereixen. Per exemple `grep -i` cerca coincidències, independentment de majúscules/minúscules o `grep -E` que fa servir les expressions ERE en lloc de BRE.

```
# Línies que comencen per C o J.
grep "^C\|^J" deudas.txt

Carina, -, eurooos
Juan López, 3, dolars
Juan Pérz, 34.8, dolar

# Línies que comencen per C o J. ERE.
grep -E "^C|^J" deudas.txt

Carina, -, eurooos
Juan López, 3, dolars
Juan Pérz, 34.8, dolar

# Línies que comencen per C, J, j o c. ERE.
grep -iE "^c|^j" deudas.txt

Carina, -, eurooos
Juan López, 3, dolars
Juan Pérz, 34.8, dolar
```

2.10. Les expressions regulars amb Bash

Bash, des de la versió 3, permet fer comparacions, emprant la variant ERE, amb l'operador =~. La comparació torna 0 si hi ha coincidència o 1 si ha fallat. Cada coincidència es guarda a l'*array* `BASH_REMATCH`. `$BASH_REMATCH` és tota la cadena, mentre que `BASH_REMATCH[1]` és la primera coincidència, `BASH_REMATCH[2]` la segona, etc.

Es poden provar expressions regulars emprant Bash:

Referència

Aquest exemple està fet a partir de l'*script* de l'article «Bash Regular Expressions» de Linuxjournal.com.

Si el fitxer se l'anomena «bashre.sh» i se li dona permisos d'execució, es poden provar les expressions regulars i diverses cadenes per si coincideixen:

```
cat bashre.sh

#!/bin/bash


if [[ $# -lt 2 ]]; then
```

```
 echo "Usage: $0 PATTERN STRINGS..."
 exit 1
fi
regex=$1
shift
echo "regex: $regex"
echo

while [[ $1 ]]
do
 if [[ $1 =~ $regex ]]; then
 echo "$1 -> coincideix"
 i=1
 n=${#BASH_REMATCH[*]}
 while [[ $i -lt $n ]]
 do
 echo " capture[$i]: ${BASH_REMATCH[$i]}"
 let i++
 done
 else
 echo "$1 -> no coincideix"
 fi
 shift
done

# Es prova el patró per a cadascun dels arguments (3, en aquest cas).
# ^A|d$ implica expressions que comencin per A o que acabin en d.
./bashre.sh '^A|d$' Asociación Cariño "Visita Madrid"

regex: ^A|d$

Asociación -> coincideix
Cariño -> no coincideix
Visita Madrid -> coincideix
```

Un altre exemple d'ús d'expressions regulars amb Bash podria ser per a comprovar si un correu electrònic és vàlid:

```
cat checkmail.sh

#!/bin/bash

# Comprova si hi ha prou arguments
if [[ $# -lt 1 ]]; then
 echo "Usage: $0 Email"
 exit 1
fi
if [[ "$1" =~ ^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}$ ]]
then
 echo "$1 could be a valid email address"
else
 echo "$1 can't be a valid email"
fi

./checkmail.sh "hola"

hola can't be a valid email

./checkmail.sh "hola@algundominio.res"

hola@algundominio.res could be a valid email address
```

3. L'editor Sed

Sed és un editor no interactiu orientat a treballar línia a línia. El que el fa realment interessant i còmode és poder fer canvis en fitxers de text de forma automàtica. Permet **cercar**, **substituir**, **esborrar**, **inserir**, etc. en un o més fitxers alhora.

És l'eina ideal per a convertir grups de fitxers de forma automàtica. L'editor Sed es pot emprar directament en el *prompt* o es pot crear un fitxer amb les accions a executar i cridar-les amb `-f`.

Per a il·lustrar les explicacions, s'emprarà el fitxer següent, anomenat «deudas.txt»:

```
cat deudas.txt # Mostra el fitxer d'exemple.

amic, 25, euros
Carina, -, eurooos
Juan López, 3, dolars
Juan Pérz, 34.8, dolar
Amorós López, 110.000, euros
Lourdes,-,dolars
```

3.1. Substitució

La comanda més important i més emprada de Sed és la **substitució**. Consisteix en «[adreça] s/patró BRE/substitució/[modificadors]».

Opcionals

L'*adreça* i els *modificadors* són opcionals.

- L'**adreça** és el nombre de línia o una expressió regular BRE. La substitució es provarà de dur a terme només en les línies que compleixin l'adreça. Si no hi és, el canvi es farà a totes les línies.

- L'**adreça** també pot ser un rang, on el principi i el final se separen per coma. Tant l'inici com el final poden ser un número de línia o una expressió regular. L'operació només es durà a terme entre les línies compreses entre les adreces.

- **s** serveix per a denotar que s'ha de fer una substitució, ja que Sed té més comandes, com veurem més endavant.

- El **patró BRE** és el patró que es vol substituir.

- La part que coincideixi amb el patró BRE serà canviat pel que hi hagi a **substitució**. Aquest valor de substitució pot ser un valor fix o estar en funció de tota la part coincident, emprant `&` o un grup amb `\n`.

- Els **modificadors** canvien la forma d'actuar per defecte de Sed. Per exemple, `g` fa que se substitueixin totes les aparicions del patró en lloc de només la primera, que és el comportament per defecte.

3.1.1. Adreçament

Les **adreces** poden ser un número de línia, un rang de línies o una expressió regular:

```
# Adreçament per rang.
# Apliquem el patró a les línies 2 a 4.
# Canviem la coma pel símbol del pipe.
sed '2,4 s/,/|/' deudas.txt

amic, 25, euros
Carina| -, eurooos
Juan López| 3, dolars
Juan Pérz| 34.8, dolar
Amorós López, 110.000, euros
Lourdes,-,dolars


# L'última línia es denota amb $.
sed '3,$ s/,/|/' deudas.txt

amic, 25, euros
Carina, -, eurooos
Juan López| 3, dolars
Juan Pérz| 34.8, dolar
Amorós López| 110.000, euros
Lourdes|-,dolars


# Adreçament per número.
# Apliquem el patró a la línia 2.
sed '2 s/,/|/' deudas.txt

amic, 25, euros
Carina| -, eurooos
Juan López, 3, dolars
Juan Pérz, 34.8, dolar
Amorós López, 110.000, euros
Lourdes,-,dolars


# Adreçament per RegEx.
# Canvia la primera coma pel tabulador només de les línies que tenen el símbol -
sed '/-/s/,/\t/' deudas.txt

amic, 25, euros
Carina     -, eurooos
Juan López, 3, dolars
Juan Pérz, 34.8, dolar
Amorós López, 110.000, euros
Lourdes    -,dolars
```

```
# Adreçament per rang RegEx.
# Canvia la primera coma pel tabulador del rang de línies des de la que comença per C fins
la que comença per A.
sed '/^C/,/^A/s/,/\t/' deudas.txt

amic, 25, euros
Carina     -, eurooos
Juan López     3, dolars
Juan Pérz     34.8, dolar
Amorós López     110.000, euros
Lourdes,-,dolars
```

3.1.2. Modificadors

Per defecte Sed només fa la substitució en la primera coincidència. Això es pot canviar amb els modificadors `/g` i `/n`.

```
# Canvia totes les comes per tabuladors.
sed 's/,/\t/g' deudas.txt

amic     25     euros
Carina     -     eurooos
Juan López     3     dolars
Juan Pérz     34.8     dolar
Amorós López     110.000     euros
Lourdes    -    dolars


# Canvia la segona ocurrència de la coma per un tabulador.
sed 's/,/\t/2' deudas.txt

amic, 25     euros
Carina, -     eurooos
Juan López, 3     dolars
Juan Pérz, 34.8     dolar
Amorós López, 110.000     euros
Lourdes,-    dolars
```

Un altre modificador d'interès és `p`, que se sol emprar amb l'opció `-n` de Sed. Així fem que Sed actuï de forma similar a Grep i ens retorni només les línies que coincideixen amb l'adreça. Això és a causa que `sed -n` fa que no s'imprimeixi cap línia (per defecte Sed imprimeix totes les línies que li arriben) i amb `p` imprimeix la línia que ha coincidit amb l'adreça.

```
# S'imprimeix cada línia que conté - (amb la modificació feta).
sed -n '/-/s/,/\t/p' deudas.txt

Carina     -, eurooos
Lourdes    -,dolars
```

3.1.3. Referències

Els canvis poden estar en funció del que es canvia emprant les **referències**. El símbol `&` serveix per tornar a mostrar el patró trobat:

```
# Substitueix línia (que tingui contingut) per "deute, línia".
sed 's/./deute, &/' deudas.txt

deute, amic, 25, euros
deute, Carina, -, eurooos
deute, Juan López, 3, dolars
deute, Juan Pérz, 34.8, dolar
```

```
deute, Amorós López, 110.000, euros
deute, Lourdes,-,dolars
```

A part de la referència del total, amb els grups podem fer referència a una part. Així, `\1` és el primer grup, `\2` el segon, etc.

```
# Canvia l'últim camp per euros. Manté les altres parts iguals.
sed 's/\(.*\),\(.*\),\(.*\)/\1,\2, euros/' deudas.txt

amic, 25, euros
Carina, -, euros
Juan López, 3, euros
Juan Pérz, 34.8, euros
Amorós López, 110.000, euros
Lourdes,-, euros
```

3.1.4. Sed i ERE

```
# Amb sed -r podem emprar expressions regulars extenses.
sed -r 's/(.*),(.*),(.*)/\1\2, pendent/' deudas.txt

amic 25, pendent
Carina -, pendent
Juan López 3, pendent
Juan Pérz 34.8, pendent
Amorós López 110.000, pendent
Lourdes-, pendent
```

3.2. Esborrar, afegir, inserir, canviar

Les comandes per a fer aquestes accions són les següents:

Comanda	Acció
`d`	Esborrar.
`a`	Afegir. Afegeix una línia després de l'adreça.
`i`	Inserir. Consisteix a inserir una línia abans de l'adreça.
`c`	Canviar. Canvia tota la línia.

```
# Esborrar les línies amb -
sed '/-/ d' deudas.txt

amic, 25, euros
Juan López, 3, dolars
Juan Pérz, 34.8, dolar
Amorós López, 110.000, euros
```

```
# Esborrar les 3 primeres línies.
sed '1,3 d' deudas.txt

Juan Pérz, 34.8, dolar
Amorós López, 110.000, euros
Lourdes,-,dolars
```

```
# Afegeix una línia després de l'adreça.
sed '1,3 a Pendent' deudas.txt

amic, 25, euros
Pendent
```

```
Carina, -, eurooos
Pendent
Juan López, 3, dolars
Pendent
Juan Pérz, 34.8, dolar
Amorós López, 110.000, euros
Lourdes,-,dolars
```

```
# Afegeix una línia abans de l'adreça.
sed '1,3 i Debt:' deudas.txt

Debt:
amic, 25, euros
Debt:
Carina, -, eurooos
Debt:
Juan López, 3, dolars
Juan Pérz, 34.8, dolar
Amorós López, 110.000, euros
Lourdes,-,dolars
```

```
# Canvia una línia.
sed '4 c Paid' deudas.txt

amic, 25, euros
Carina, -, eurooos
Juan López, 3, dolars
Paid
Amorós López, 110.000, euros
Lourdes,-,dolars
```

```
# Canvia totes les línies del rang per una sola línia.
sed '1,3 c Paid' deudas.txt

Paid
Juan Pérz, 34.8, dolar
Amorós López, 110.000, euros
Lourdes,-,dolars
```

Hi ha vegades que volem que l'acció es dugui a terme en totes les línies, excepte les que compleixen el patró. El símbol per a fer-ho, la negació, és `!`.

```
# Esborrar les línies que no tinguin -
sed '/-/ !d' deudas.txt

Carina, -, eurooos
Lourdes,-,dolars
```

3.3. *Scripts* Sed

Fins aquest moment, hem vist com aplicar una comanda línia a línia sobre un fitxer. Però és possible que ens interessi aplicar no una sinó unes quantes ordres una rere l'altra. Per a fer-ho tenim dues opcions:

1) Agrupar les comandes sed entre {}.
2) Crear un fitxer amb totes les ordres i cridar-lo per a aplicar-les.

```
# Agrupació de comandes.
# Col·lapsa «o»'s i treu l'última coma.
sed '{
 s/o\+/o/g
 s/\(.*\),\(.*\),\(.*\)/\1,\2\3/
}' deudas.txt

amic, 25 euros
```

```
Carina, - euros
Juan López, 3 dolars
Juan Pérz, 34.8 dolar
Amorós López, 110.000 euros
Lourdes,-dolars


# Comandes des d'un fitxer.
# Contingut del fitxer.
cat demo.sed

#En els scripts Sed es poden posar comentaris.
s/o\+/o/g
s/\(.*\),\(.*\),\(.*\)/\1,\2\3/

# Aplicació de les comandes del fitxer demo.sed a deudas.txt
sed -f demo.sed deudas.txt

amic, 25 euros
Carina, - euros
Juan López, 3 dolars
Juan Pérz, 34.8 dolar
Amorós López, 110.000 euros
Lourdes,-dolars
```

Per a treure profit dels *scripts* Sed, convé conèixer com funciona:

- **Sed actua línia a línia**: agafa una línia i li aplica totes les comandes de l'*script*. Llavors agafa la línia següent i li aplica totes les comandes de l'*script* i continua així fins que acaba el fitxer a tractar.

- **Les modificacions no es fan en el fitxer original sinó en una còpia temporal de cada línia** (per tant, consumeix poca memòria). Si volem guardar els canvis, podem emprar la redirecció de Bash, per exemple, però mai en el mateix fitxer.

- **L'ordre de les comandes importa**. Després dels canvis fets per una comanda, les següents actuen sobre la línia modificada, no sobre l'original.

- **Una vegada tractada una línia del fitxer, es mostra la modificació a la sortida estàndard** (excepte si s'invoca Sed amb l'opció `-n`).

4. Construcció d'expressions regulars

Dominar les expressions regulars té similituds amb aprendre la notació matemàtica. Per entendre les fórmules, cal conèixer el significat de cada un dels símbols que hi apareixen.

En expressions regulars, l'avantatge, si es compara amb l'àlgebra, és que els símbols mantenen el significat constantment.

Però aprendre RegEx, igual que per la notació matemàtica, exigeix posar-hi atenció i pràctica. Al principi, l'esforç necessari és alt i pot semblar que no val la pena aprendre-ho, però a la llarga surt a compte.

És imprescindible practicar. Poder entendre i saber construir RegEx no solament consisteix a saber-se el llistat de tots els símbols sinó emprar-los i conjugar-los per a obtenir patrons útils. Una forma d'anar avançant és analitzant i tractant de millorar els RegEx construïts per altres, identificant les idees que han servit per a crear-los. Acostumar-se a entendre les expressions regulars serveix per estar preparat per si les dades en què s'apliquen canviessin.

Els patrons es construeixen per un conjunt de dades, que encara que pot ser molt ampli, té un abast.

Per exemple, es poden fer expressions regular per a trobar el DNI, números de la seguretat social (que depèn del país), números de targetes de crèdit, dates (amb format divers), preus, etc. Així, si pensem en les dates, 01-02-04 pot voler dir, segons el format, l'1 de febrer de 2004 o de 1904, o, fins i tot, dia 4 de febrer de 2001 o de 1901. És a dir, el patró a construir servirà per un o alguns d'aquest casos però potser no pels altres.

Tampoc no hem de pretendre crear el patró perfecte d'un cop. La tasca d'anar trobant el patró adequat és incremental i ens serveix per anar estudiant les dades. Cada patró que anem fent és com una hipòtesi que s'ha de validar amb les dades i molts cops també implica suposicions sobre les pròpies dades.

Les eines que fan servir les expressions regulars són molt potents i, per tant, molt complexes. Com diu, però, Barnett (2001) aquesta complexitat no pot ser una excusa per no aprofitar la part més simple. I és que amb les expressions regulars podem fer-nos una idea ràpida, en la línia de comandes, directament i immediatament, del nombre d'aparicions d'un determinat patró, per exemple. Aquestes primeres proves poden ser la llavor de la construcció de patrons més complexos i refinats que seran els que acabaran en producció.

L'objectiu que tinguem per emprar les expressions regulars també marcarà l'estratègia a seguir. Si el que volem és saber si un patró apareix en un conjunt de dades, és a dir, volem saber si hi és almenys un cop, ens bastarà una expres-

sió simple en què ja veiem que el patró és present, encara que deixi moltes aparicions sense trobar. Però, si el que ens interessa és trobar totes les ocurrències, haurem de construir una expressió molt precisa.

Finalment, cal destacar la importància d'elegir l'eina adequada en cada cas. Per a fer una exploració ràpida *grep*, *cut*, *head*, *tail*, etc. han de ser la primera opció. Si volem modificar (substituir, inserir, esborrar) determinades parts d'un o molts fitxers, Sed o AWK poden ser adequats. Si les dades no responen a expressions regulars, no estan prou estructurades, s'haurà de recórrer a altres eines de tractament de dades, de processament del llenguatge natural o de programació.

4.1. Exemples de RegEx

4.1.1. Filtrar, trobar i validar

1) Amb **Ubuntu** podem mostrar totes les carpetes a «/usr/share/locale» que continguin «es»:

```
# ERE
# Comença per d, ja que és un directori, després hi ha diversos caràcters i "es".
ls -ltr /usr/share/locale | grep -E '^d.*es'

drwxr-xr-x 3 root root 4096 d'abr 16 2018 es
drwxr-xr-x 3 root root 4096 d'abr 26 2018 es_VE
drwxr-xr-x 3 root root 4096 d'abr 26 2018 es_MX
drwxr-xr-x 3 root root 4096 d'abr 26 2018 es_CO
drwxr-xr-x 3 root root 4096 d'abr 26 2018 es_CL
drwxr-xr-x 3 root root 4096 d'abr 26 2018 es_AR
```

```
# BRE
ls -ltr /usr/share/locale | grep '^d.*es'

drwxr-xr-x 3 root root 4096 d'abr 16 2018 es
drwxr-xr-x 3 root root 4096 d'abr 26 2018 es_VE
drwxr-xr-x 3 root root 4096 d'abr 26 2018 es_MX
drwxr-xr-x 3 root root 4096 d'abr 26 2018 es_CO
drwxr-xr-x 3 root root 4096 d'abr 26 2018 es_CL
drwxr-xr-x 3 root root 4096 d'abr 26 2018 es_AR
```

2) Comprovar que un nom d'usuari només té els caràcters permesos (lletres minúscules, números, guió i guió baix) i que té una longitud d'entre 3 i 16:

Referència

J. Fox (2019). «Regex cookbook - Top 10 Most wanted regex» [en línia]. Factory Mind.

- ERE: `/^[a-z0-9_-]{3,16}$/`
- BRE: `/^[a-z0-9_-]{3,16}$/`

Aquest *regex* obliga a començar per una lletra minúscula, un número, un guió o guió baix i que hi hagi entre 3 i 16 símbols.

```
# BRE
echo "petito" | grep "^[a-z0-9_-]\{3,16\}$"
echo "petito y juan" | grep "^[a-z0-9_-]\{3,16\}$"
echo "p" | grep "^[a-z0-9_-]\{3,16\}$"

petito
```

```
# ERE
echo "petito" | grep -E "^[a-z0-9_-]{3,16}$"
echo "petito y juan" | grep -E "^[a-z0-9_-]{3,16}$"

petito
```

3) Comprovar que un número de DNI és vàlid. La lletra pot estar unida al número, separada per un espai o per un guió.

- ERE: `/[0-9]{8}( |-)?[a-zA-Z]/`
- BRE: `/[0-9]{8}( |-)\?[a-zA-Z]/`

Aquest *regex* coincideix amb les cadenes amb vuit dígits `[0-9]{8}`, un espai, un guió o cap símbol `( |-)?`, i una lletra minúscula o majúscula `[a-zA-Z]`.

```
# BRE
echo "el dni és 41000000e." | grep "[0-9]\{8\}\( |-\)\?[a-zA-Z]"
echo "el dni és 41000000--e." | grep "[0-9]\{8\}\( |-\)\?[a-zA-Z]"

el dni és 41000000e.
```

```
# ERE
echo "41000000e" | grep -E "[0-9]{8}( |-)?[a-zA-Z]"
echo "41000000--e" | grep -E "[0-9]{8}( |-)?[a-zA-Z]"

41000000e
```

4.1.2. Substituir

En el fitxer que hem estat utilitzant per a fer proves, podem eliminar les línies sense un valor numèric, unir la moneda amb la quantitat i substituir el nom de la moneda pel seu símbol.

```
sed '{
/-/ d # Esborrem les línies amb guió.
s/, euro*s/€/ # Substituïm euro pel seu símbol.
s/, dolar.\?/\$/ # Substituïm dòlar pel seu símbol.
}' deudas.txt

amic, 25€
Juan López, 3$
Juan Pérz, 34.8$
Amorós López, 110.000€
```

4.1.3. Reordenar

Amb Sed i les referències, podem reordenar o fusionar camps del fitxers estructurats.

```
# Reordena els camps posant la quantitat en primer lloc.
# Cal fixar-se en els espais.
sed 's/\(.*\), \(.*\), \(.*\)/\2, \3, \1 /' deudas.txt

25, euros, amic
-, eurooos, Carina
3, dolars, Juan López
34.8, dolar , Juan Pérz
110.000, euros , Amorós López
Lourdes,-,dolars
```

```
# Si cal ordenar les línies entre elles, ho podem fer redirigint-les a sort.
sed 's/\(.*\), \(.*\), \(.*\)/\2, \3, \1 /' deudas.txt | sort -rn -k1

110.000, euros , Amorós López
34.8, dolar , Juan Pérz
25, euros, amic
3, dolars, Juan López
Lourdes,-,dolars
-, eurooos, Carina
```

4.2. Construcció incremental d'expressions regulars

A l'hora de construir una expressió regular complexa, és recomanable començar per una de més simple i anar-la refinant fins que s'ajusti al que es vol aconseguir.

Per a il·lustrar-ho treballarem amb dates. Concretament emprarem el fitxer «fechas.txt» següent:

```
cat fechas.txt

Día 1 marzo
El 8 Marzo es el día Internacional de la Mujer
Algún día de Marzo es el día de los amigos
Día Nacional del Trabajador de Construcción
Día Mundial del Consumidor
Marzo Día del Psicoorientador
El 22 de Marzo no es el día del optómetra
24-03-2020 Día del Locutor
27-3-2020 Día Internacional del Teatro
El 31-3-20 es el Día internacional contra el cáncer de Colon
1-4-2020 Abril Día del Controlador Técnico de Audio
8-05-20 Día de la Madre
01/06/20 Día Internacional del Niño
1/06/2020 Día del Campesino
05/6/20 Día del medio ambiente
14 de 06 del 2020 es el día mundial del donante de Sangre
17 del 6 de 2020 es el día del Higienista Dental
```

Volem tractar les línies que tinguin el dia del mes, el mes i l'any i deixar la resta. I volem posar totes les dates en un format comú, separat per un guió. Podem començar trobant on hi ha números:

```
# Cerca dígits.
grep '[0-9]' fechas.txt

Día 1 marzo
El 8 Marzo es el día Internacional de la Mujer
El 22 de Marzo no es el día del optómetra
24-03-2020 Día del Locutor
27-3-2020 Día Internacional del Teatro
El 31-3-20 es el Día internacional contra el cáncer de Colon
1-4-2020 Abril Día del Controlador Técnico de Audio
8-05-20 Día de la Madre
01/06/20 Día Internacional del Niño
1/06/2020 Día del Campesino
05/6/20 Día del medio ambiente
14 de 06 del 2020 es el día mundial del donante de Sangre
17 del 6 de 2020 es el día del Higienista Dental
```

Ara refinem una mica més, ja que les tres primeres línies no les volem modificar. Observeu que les dades que volem estan formades per un o dos números, elements al mig, un o dos números, més elements al mig i dos o quatre números.

```
# Identifiquem grups d'un o dos dígits.
grep '\([0-9]\{1,2\}\)' fechas.txt

Día 1 marzo
El 8 Marzo es el día Internacional de la Mujer
El 22 de Marzo no es el día del optómetra
24-03-2020 Día del Locutor
27-3-2020 Día Internacional del Teatro
El 31-3-20 es el Día internacional contra el cáncer de Colon
1-4-2020 Abril Día del Controlador Técnico de Audio
8-05-20 Día de la Madre
01/06/20 Día Internacional del Niño
1/06/2020 Día del Campesino
05/6/20 Día del medio ambiente
14 de 06 del 2020 es el día mundial del donante de Sangre
17 del 6 de 2020 es el día del Higienista Dental
```

```
# Identifiquem grups d'un o dos dígits i que després tenen un espai, un guió, de o del.
grep '\([0-9]\{1,2\}\)\( \|-\|/\| de \| del \)' fechas.txt

Día 1 marzo
El 8 Marzo es el día Internacional de la Mujer
El 22 de Marzo no es el día del optómetra
24-03-2020 Día del Locutor
27-3-2020 Día Internacional del Teatro
El 31-3-20 es el Día internacional contra el cáncer de Colon
1-4-2020 Abril Día del Controlador Técnico de Audio
8-05-20 Día de la Madre
01/06/20 Día Internacional del Niño
1/06/2020 Día del Campesino
05/6/20 Día del medio ambiente
14 de 06 del 2020 es el día mundial del donante de Sangre
17 del 6 de 2020 es el día del Higienista Dental
```

```
# Repetim el patró anterior dos cops.
grep '\([0-9]\{1,2\}\)\( \|-\|/\| de \| del \)\([0-9]\{1,2\}\)\( \|-\|/\| de \| del \)' fechas.txt

24-03-2020 Día del Locutor
27-3-2020 Día Internacional del Teatro
El 31-3-20 es el Día internacional contra el cáncer de Colon
1-4-2020 Abril Día del Controlador Técnico de Audio
8-05-20 Día de la Madre
01/06/20 Día Internacional del Niño
1/06/2020 Día del Campesino
05/6/20 Día del medio ambiente
14 de 06 del 2020 es el día mundial del donante de Sangre
17 del 6 de 2020 es el día del Higienista Dental
```

```
# Hi afegim l'any, que és \([0-9]\{2,4\}\), és a dir , pot ser de dos o quatre dígits.
grep '\([0-9]\{1,2\}\)\( \|-\|/\| de \| del \)\([0-9]\{1,2\}\)\( \|-\|/\| de \| del \)\
([0-9]\{2,4\}\)' fechas.txt

24-03-2020 Día del Locutor
27-3-2020 Día Internacional del Teatro
El 31-3-20 es el Día internacional contra el cáncer de Colon
1-4-2020 Abril Día del Controlador Técnico de Audio
8-05-20 Día de la Madre
01/06/20 Día Internacional del Niño
1/06/2020 Día del Campesino
05/6/20 Día del medio ambiente
14 de 06 del 2020 es el día mundial del donante de Sangre
17 del 6 de 2020 es el día del Higienista Dental
```

Ara tenim cinc grups. En el primer hi ha el dia (`[0-9]{1,2}`), en el segon i quart el separador (`|-|/| de | del`), en el tercer el mes (`[0-9]{1,2}`), i en el cinquè l'any (`[0-9]{2,4}`). Apliquem-hi Sed.

Abans però, hi ha un aspecte de Sed que ens convé fer servir per a aquest cas: en lloc d'emprar la barra / com a separador podem emprar el símbol que més ens convingui.

Atès que ja estem fent servir aquesta barra dins l'expressió regular i, com que no fem servir l'arrova, podem emprar-la com a separador.

```
sed 's@\([0-9]\{1,2\}\)\( \|-\|/\| de \| del \)\([0-9]\{1,2\}\)\( \|-\|/\| de \| del \)\
([0-9]\{2,4\}\)@\1-\3-\5@' fechas.txt

Día 1 marzo
El 8 Marzo es el día Internacional de la Mujer
Algún día de Marzo es el día de los amigos
Día Nacional del Trabajador de Construcción
Día Mundial del Consumidor
Marzo Día del Psicoorientador
El 22 de Marzo no es el día del optómetra
24-03-2020 Día del Locutor
27-3-2020 Día Internacional del Teatro
El 31-3-20 es el Día internacional contra el cáncer de Colon
1-4-2020 Abril Día del Controlador Técnico de Audio
8-05-20 Día de la Madre
01-06-20 Día Internacional del Niño
1-06-2020 Día del Campesino
05-6-20 Día del medio ambiente
14-06-2020 es el día mundial del donante de Sangre
17-6-2020 es el día del Higienista Dental
```

4.3. Avantatges i limitacions

Emprar les expressions regulars POSIX fa que la **primera presa de contacte i exploració de les dades sigui immediata, directa i portable**. No cal instal·lar res ja que les eines que hem vist hi són presents per defecte o són molt fàcils d'instal·lar amb el gestor propi de la distribució en la gran majoria de les distribucions Linux i Unix. Així, des de la mateixa línia de comandes, podem aproximar-nos a les dades molt ràpidament.

Les eines que hem vist, tant Grep com Sed, i també d'altres, fan un **ús eficient de la memòria**, sent adients per a tractar fitxers grans. Això és a causa que, com que en els anys 60 els ordinadors estaven molt limitats, estan dissenyades per a treballar amb aquesta limitació de recursos.

L'ús d'expressions regulars suposa un estalvi de feina i de manteniment. L'alternativa de fer un programa en el seu lloc no compleix aquests beneficis. Un programa d'aquestes característiques hauria d'anar mirant caràcter a caràcter i anar guardant l'estat (que és el que es fa amb les expressions regulars). Seria un programa amb molts casos, moltes regles, difícil de seguir i molt pro-

pens a contenir errors. Si calgués fer-hi un canvi, seria extremadament difícil de fer. No hi ha dubte que és molt millor emprar una o més expressions regulars, comentades i documentades.

Els RegEx són ideals per a tractar seqüències de caràcters que presenten un cert grau d'uniformitat. Però, depenent del tipus de document amb el qual haguem de treballar, les expressions regulars no són les més adequades, encara que poden ser útils per a tractar patrons simples. El que convé en aquests casos (json, xml, html, etc) és emprar un *parser* o una eina específica, com per exemple *xpath*, per a extreure informació d'xml.

Bibliografia

Bibliografia bàsica

Barnett, B. (2019). «The Grymoire's tutorial on SED». Disponible a: www.grymoire.com/Unix/Sed.html.

Dougherty, D.; Robbins, A. (1997). *Sed & Awk: UNIX Power Tools*. Massachusetts: O'Reilly Media.

Shotts, W. (2009). «20 – Regular Expressions». *The Linux Command Line*(1a. ed., pàg. 261-281). LinuxCommand.org.

Referències

Gnu.org (2018). «sed, a stream editor». Disponible a: www.gnu.org/software/sed/manual/html_node/index.html#SEC_Contents.

Fox, J. (2019). «Regex cookbook — Top 10 Most wanted regex». Medium. Disponible a: medium.com/factory-mind/regex-cookbook-most-wanted-regex-aa721558c3c1.

Frazier, M. (2008). «Bash Regular Expressions». Linuxjournal.com. Disponible a: www.linuxjournal.com/content/bash-regular-expressions.