5. Потоци, пренасочвания и обработка на текст  
• Файлово и входно-изходно пренасочване;  
• Пренасочване между програми (piping);  
• File globbing (wildcard модели);  
• Файлови статистики (wc);  
• Текстова обработка със sed и awk;  
• Подмяна на символи (tr);  
• Сортиране (sort);  
• Откриване на дублирания (uniq);  
• Извличане на колони от текст (cut);  
• Комбиниране на файлове (cat, paste);  
• Търсене във файлове чрез grep;  
• Регулярни изрази.

Част 5

**Потоци, пренасочвания и обработка на текст**  
• Файлово и входно-изходно пренасочване;  
• Пренасочване между програми (piping);  
• File globbing (wildcard модели);  
• Файлови статистики (wc);  
• Текстова обработка със sed и awk;  
• Подмяна на символи (tr);  
• Сортиране (sort);  
• Откриване на дублирания (uniq);  
• Извличане на колони от текст (cut);  
• Комбиниране на файлове (cat, paste);  
• Търсене във файлове чрез grep;  
• Регулярни изрази.

Много често в ежедневните задачи на един линукс системен администратор му се налага да обработва големи потоци от данни ( както споменах по рано огромни лог файлове безкрайно дълги комфигурационни файлове и .т.н.)

За по лесна работа програмистите са създали набор от инструменти който позволява по лесна обработка на масиви от информация .

В тази секция ще се запознаем как да пренасочваме входът от една команда да бъде изход във файл или вход за друга програма . Как да заместваме изрази и фрази във файлове или най образно казано как да извличаме максимална информация от даден файл .

Ще започнем с пренасочването или piping да кажем че искаме да си направиме копие на всички потребители в нашата операционна система но нямаме права за това ( може и да имаме не е задължително) но да кажеме че искаме да копираме списъка с потребителите в линукс в нашата домашна папка за да може да го позлваме за друго и към него да добавим и списъка с паролите

Файлът ще ни се казва users

За целта ще използваме командата cat и ще пренасочиме изходът и във въпросният файл

Ето как ще изглежда командата в нашият случай

Cat /etc/passwd > /home/user/users



Обаче освен списъка с потребителите ни трябва във същият файл и списък с паролите

Ако изпълниме командата с единичен знак > ще презапише върху съществуващите записи новата информация за това ще използваме >>

Командата за извличане на пароли е

Cat /etc/shadow >> /home/user/users

По този начин извкичаме само съдържанието на файлът без да променяме правата му както ще ви покажа по късн окато стигнеме до текстови редактори .

До тук се запознахме с командата cat и как да пренасочваме изходът и във файл който в последствие може да редактираме .

Това е една от най често използваните команди в линукс. Командата има за цел да извлича съдържанието от файл и да го показва на екрана или да го пренасочи в друг файл както видяхме по нагоре .

Командата **cat** има следния синтаксис: cat файл1 файл2 файлN.  
Тя се използва най-вече с текстови файлове. Извежда на стандартния изход съдържанието на файл1, файл2, , файлN последователно.

Ако се прнасочи към файл може да копита съдържанието на множество файлове в 1

Сега ако катнем нашият файл който създадохме ще видим че съдържаниеот на двата файла е обединено

До тук се зяапознахме и със основното пренасочване между програми но ще го обогатиме в следващите редове

Запознахме се с част от операторите за пренасочване като | > и >> но те могат да работят и в обратна посока тоест да са вход за някъква команда и съответно да са изход в нов текстов файл

Да кажем че искаме в нашият лог файл да търсиме само съобщения които съдържат определена фраза например кернел

Обаче не ни се чете в целият лог файл

За целта ще си направим cat /var/log/messages и ще добавиме към изхода на командата да ни търси за думичката кернел .

Командата ни вече ще изглежда така **cat /var/log/messages | grep “kernel”**

Както се вижда от горният пример може да търсиме за определена дума в определен файл с командата grep

Командата grep представлява търсачка ред по ред като търси за определена дума или фраза във посочен файл или списък със файлове .

Що се отнася до разликите между Linux дистрибуциите, в този случай те не играят никаква роля, тъй като командата grep, която ви интересува, е достъпна по подразбиране в повечето сборки и се използва по същия начин. Днес бихме искали да обсъдим не само ефекта на grep , но и основните аргументи, които могат значително да опростят процедурата за търсене.

### **Търсене със заснемане на низ**

Понякога потребителите трябва да намерят не само точно съвпадение в редовете, но и да открият информацията, която идва след тях, например при докладване на конкретна грешка. Тогава правилното решение е да приложите атрибутите. Въведете grep -A3 "word" testfile в конзолата, за да включите следните три реда в резултата след мача. Можете да напишете -A4 , след което ще бъдат уловени четири реда, няма ограничения.

Ако вместо -A използвате аргумента -B + количество низ , резултатът е данните, които са до точката на въвеждане.

Аргументът -С , от своя страна, улавя линии около ключова дума.

По-долу можете да видите примери за задаване на посочените аргументи. Моля, обърнете внимание, че трябва да вземете предвид случая и да поставите двойни кавички.

grep -B3 "word" testfile   
grep -C3 "word" testfile

### **Търсете ключови думи в началото и в края на редовете**

Необходимостта да се дефинира ключова дума, която е в началото или в края на ред, най-често се среща по време на работа с конфигурационни файлове, където всеки ред е отговорен за един параметър. За да видите точния запис в началото, трябва да регистрирате grep "^word" testfile . Знакът ^ отговаря само за използването на тази опция

Търсенето на съдържание в края на реда става приблизително по същия начин, само в кавички трябва да добавите знака $ и командата ще изглежда така: grep "word$" testfile .

### **Търсете номера**

При търсене на желаните стойности, потребителят не винаги има информация относно точната дума, присъстваща в низ. След това процедурата за търсене може да се извърши чрез номера, което понякога значително опростява задачата. Необходимо е само да използвате командата под формата на grep "[0-7]" testfile , където "[0-7]" е диапазонът от стойности, а testfile е името на файла, който ще бъде сканиран.

### **Анализ на всички директорийни файлове**

Сканирането на всички обекти в същата папка се нарича рекурсивно. Потребителят трябва да приложи само един аргумент, който анализира всички файлове в папката и показва съответните линии и тяхното местоположение. Ще трябва да въведете grep -r "word" /home/user/folder , където / home / user / folder е пътят към директорията за сканиране.

### **Търсете низове без определена дума.**

Обмислената програма може не само да намира думи във файлове, но и да показва линии, в които няма зададена от потребителя стойност. След това, преди да въведете стойността на ключа и файла, -v добавя -v . Благодарение на нея, когато активирате командата, ще видите само съответните данни.

Синтаксисът на *grep* събра още няколко аргумента, които могат да бъдат обяснени накратко:

* -I - показва само имената на файловете, които отговарят на критериите за търсене;
* -s - деактивиране на известията за открити грешки;
* -n - показва номера на реда във файла;
* -b - показва номера на блока преди реда.

Нищо не ви пречи да използвате множество аргументи за едно откритие, просто ги въведете разделени с интервал, без да забравяте да вземете предвид случая.

Както забелязахме grep командата е широко използвана и ще ни се налага често да я ползваме .

Да си представиме че имаме директория с милиони файлове от различни типове и желаеме да ги сортираме по папки как да стане лесно това . Вече знам ка кда ползваме команди като mv и cp но ако желаем да преместим хиляди файлове от даден тип в определена папка наистина ли трябва да си играем файл по файл ?

Отогворът в линукс е Не за целта има така наречените File globbing

Това е променлива която замества чат от името на файл или цялото име на файла до определена буква или символ .

Както споменах по горе да кажеме че имаме директория със смесено съдържание текстови файлове снимки музика ..... и искаме да си ги организираме по папки

Стандартният начин е със cp име\_на\_файл1.разширение име\_на\_файл2.разширение /път/до/новата/папка

И така до безкрайност ами ако използваме глобална променлива каквато е \*

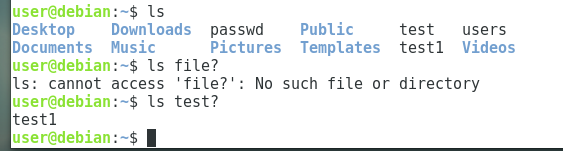
Нашата команда ще изглежда по следният начин cp \*.разширение /път/до/новата/папка

Както се вижда от следният пример копирахме всички файлове с .разширение в новата папка

По същият начин може и да ги преместим с командата mv

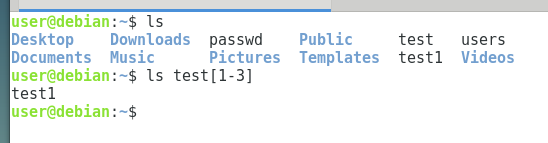
Друг пример за глобална променлива е ? тя може да се ползва за съвпадение на до 1 буква

Например искаме да видиме всички файлове които започват с името test и съдържат още един символ



Може също и да се посочи интервал на търсенето например със [ ]

Да кажем че искаме да намериме всички файлове които в името си имат числа от 1 до 3



Както се вижда от примерът виждаме само файловете тест които съдържат число накрая

Друг пример е изключение например в горният пример искаме да видиме всички файлове които започват с тест но не са с числото !3 в името .

В случаят нямаме такива

Синтаксисът е почти същият като командата изглежда по следният начин ls test[!3]

Много внимавайте как ползвате \* че може по погрешка да изтриете файлове които не е трябвало

Никога не използвайте изтрий файлове и директории в руут \*

Или **rm –rf /\*** това ще изтрие всички файлове и папки в операционната система

Реално \* означава всички освен ако не е посочено от кой тип независимо дали става дума за файлове или папки

Grep е малка Unix програма за намиране на съвпадащи модели. Започна като програма Unix, може да се намери на Linux, Mac и BSD. Тя може да чете почти всеки текст, което означава, че може да чете вход от други команди, или може да отвори и разглежда директно файлове. Grep е невероятно полезно, особено за преглеждане на директории от командния ред.

С това, защо не опитате една от най-често използваните за Grep, намиране на файл в директория. Разгледайте папката "~ / Downloads" за всички изображения "jpg".

Grep ще ги изброи и ще подчертае частта ".jpg", тъй като това е, което търсите.

Извеждате изхода (с командата "|") на почти всяка команда в grep . Ако имате обикновен текстов файл, който се намира наоколо, извадете го и преместете резултата в Grep, за да намерите конкретна дума.

Grep ще отпечата всички линии във файла, съдържащи думата, която сте му казали да търси.

**Системите, подобни на Unix, са чувствителни към буквите, което означава, че "Ubuntu" е съвсем различно от "ubuntu". Това може да е болка при търсене на нещо. Греп има знаме, за да премахне проблема. Добавете флага -i към търсенето си и Grep ще пренебрегне случая.**

## Търсене в файл

## Не би било толкова удобно да се нуждаете от cat всеки път, когато искате да търсите файл. Всъщност не. Grep може да търси чрез самите файлове. Предайте пътя до файла и Греп ще направи останалото.

grep -i word /path/to/file.txt

## Рекурсивни търсения

## Grep може да търси едновременно повече от един файл или директория. Бъдете предупредени, ако търсите имена на файлове, Grep ще търси и файлове по подразбиране. Знакът -r казва на Греп да търси рекурсивно.

ls ~ / Downloads grep -r .deb

Можете да го сдвоите и с други знамена. Може да искате да включите флага " -I когато правите рекурсивни търсения, за да запазите Grep да преглежда през двоични файлове.

ls ~ / Downloads grep -iIr .deb

## Намерете противоположното

Можете също така да кажете на Греп да търси всичко, което *не* съдържа посочения шаблон. Това би било добре в случаите, когато трябва да намерите грешка или имате директория с много файлове от един или два вида.

grep -rv '192.168.1.110' / etc / nginx

## Думи и линии

Също така може да е полезно да кажете на Греп да търси пълни думи или линии, вместо нещо, съдържащо определен модел. Представете си, че търсите поредица от символи, които са често срещани като част от една дума, но трябва да са отделни. Например думата "то". Очевидно нямаше да има много пъти, *всъщност* бихте искали да търсите модел като "то", но това е доказателство.

котка textfile.txt grep -w it

Вместо да отпечатва всяка дума, която съдържа модела, Grep само ще отпечата думата само по себе си. Той прави същото за всички редове с флага -x, така че ако търсите фраза или единичен ред в конфигурационния файл, това наистина може да помогне.

Ако се интересувате от разследването на *всичко, което* Греп може да направи, стартирайте man grep в терминал за цялата документация. Тази статия обхваща основите и всичко, което обикновено се нуждаете от този мощен инструмент.

Следващите две команди които ще разгледаме са **cat** и **paste**

CAT e една от най често използваните команди в линукс заедно със cd ls и grep .

Най- често се използва ако искаме да видим съдържанието на файл без да го отваряме за редакция като може да изкарва съдържанието на различни файлове . Синтаксисът е като на повечето линукс команци

Cat [Options ] [file1] [file2][file N]

Най често се комбинира с други команди като grep , more , less

Друга много полезна команда е paste тя се използва за да обедини файлове рколона по колона

Синтаксисът е същият

paste [OPTION]... [FILE]...

За по нагледно ще дадеме следният пример

Имаме 3 файла които съдържат съответно

Файл 1 номерация от 1 до 5

Файл 2 имена

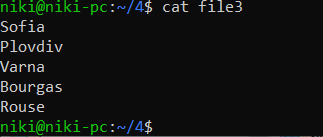
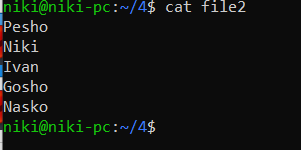
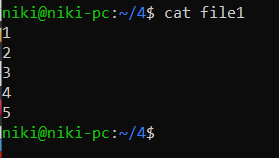
Файл 3 градове

Да кажеме че искаме да получиме нов файл в който се съдържат трите горни файла и да са ред по ред

Командата ни ще изглежда по следният начин

Paste file1 file2 file2 > file4

Съдържанието е както следва

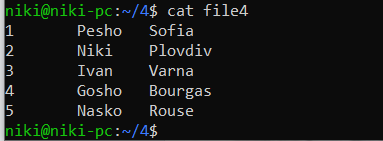


Но да кажем че трябва да ги обединиме тези файлове във 1 общ файл ще го наречеме file4

Да кажем че искаме да ги сортираме в същият ред тоест номер име град

Командата ще ни изглежда по следният начин 

И съответно да видим резултата



Да обаче да кажеме че не ни харесва форматирането на текста а искаме например да са разделени със запетайка тоест ще добавиме и параметър към командата

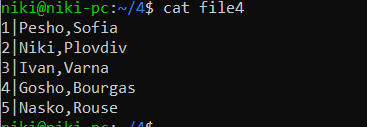
И тя ще изглежда по следният начин



Като в случаят –d е за разделител а в кавичките е какъв да е той като в случаят между числата и името ще използваме черта а между името и града запетайка а може да бъде и всеку друг символ като две точки точка и запетайка и т.н

Тоест това което направихме е след първият файл ( с номерата ) добавяме | съединяваме със вторият файл (имената) и слагаме запетайка , и добавяме третият файл

Изходът ни изглежда по следният начин

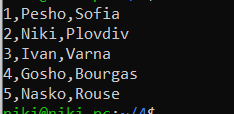


Като правилото е разделителят/разделителите да са оградени в кавички

Ако искаме всичко да е само със запетайки просто посочваме че желаем само един разделител запетайка



И съответно файловете са ни разделени както следва



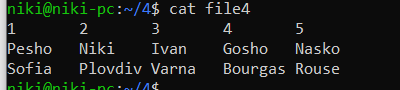
Може също така да ги сортираме файловете и не само вертикално но и хоризонтално

За селта се ползва опцията –s

Командата ще изглежда по следният начин

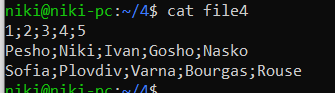


и резултатът



Съответно и тук може да ползваме разделител като например точка и запетайка



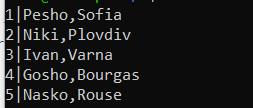


Както се вижда файловете са вертикално подредени разделени от точка и запетайка .

Друг начин да извличаме колони от текст е с командата **cut**

**Тя най често се използва за да се премахне секция или дума от всеки ред във файл**

**За целта ще ползваме горният пример като ще върнеме съдържанието на файл4 в колона (вижте командата paste )**



Да кажем че от въпросният файл трябва да махнем първата колона ( тази с цифрите )

Начините са 2 като за единят хипотетично ще кажем че сме изтрили файловете първоизточници (file1 file2 file3 ) тоест можеше да ползваме командата paste и да обединим файл 2 и файл 3 обаче както казахме сме ги изтрили .

Тук на помощ ни идва командата **cut**

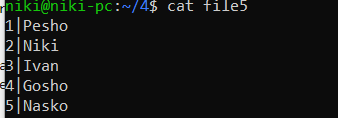
**Да кажем че искаме да премахнем третата колона тоест града**

**командата ще ни изглежда по следният начин**

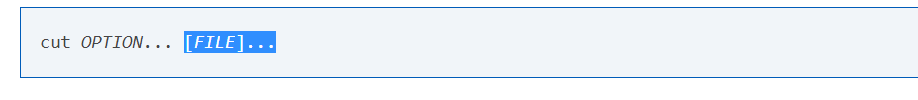


**Където –d е разделиятелят след който ще премахваме –f 1 полетата които ще премахнем**

**Файл 4 ни е входният файл а файл 5 изходният**



**Синтаксисът е както следва**



**Ето и опциите които се ползват към командата**

|  |  |
| --- | --- |
| **-b**, **--bytes=***LIST* | **Изберете само байтовете от всеки ред, както е посочено в LIST. LIST определя байт, набор от байтове или диапазон от байтове; вж. по-долу Задаване на СПИСЪК.** |
| **-c**, **--characters=***LIST* | **изберете само тези знаци** |
| **-d**, **--delimiter=***DELIM* | **използвайте DELIM вместо TAB за разделяне на полета** |
| **-f**, **--fields=***LIST* | **изберете само тези полета; също така отпечатва всеки ред, който не съдържа разделител, освен ако не е зададена опцията -s** |
| **-n** | **Тази опция се игнорира, но е включена поради причини за съвместимост.** |
| **--complement** | **допълват набора от избрани байтове, знаци или полета.** |
| **-s**, **--only-delimited** | **не отпечатва линии, които не съдържат разделители.** |
| **--output-delimiter=***STRING* | **използвайте STRING като низ за разделяне на изхода. По подразбиране се използва входният ограничител.** |
| **--help** | **Покажете помощно съобщение и излезте.** |
| **--version** | **изходна информация за версията и изход.** |

Следващите команди които ще ртагледаме са sort и uniq

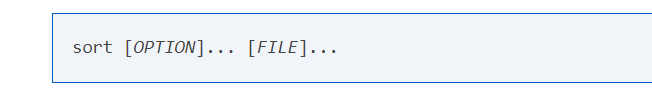
sort е проста и много полезна команда, която ще пренареди линиите в текстов файл, така че да бъдат подредени, числено и по азбучен ред. По подразбиране правилата за сортиране са:

Линии, започващи с номер, ще се появят преди редове, започващи с буква.

Линии, започващи с буква, която се появява по-рано в азбуката, ще се появи преди редове, започващи с буква, която се появява по-късно в азбуката.

Линии, започващи с малка буква, ще се появяват преди редове, започващи с една и съща буква в главни букви.

Правилата за сортиране могат да бъдат променяни според опциите, които предоставяте на командата за сортиране; те са изброени по-долу.

Синтаксисът е както следва 

Съответно в зависимост от опциите които му се зададат може да сортира по различен начин

Списъкът със опциите е както следва

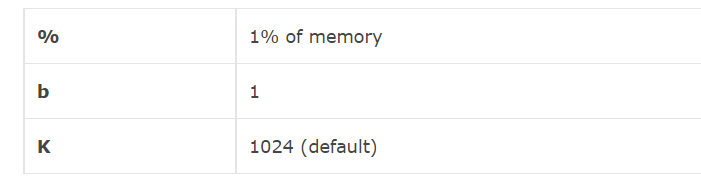
|  |  |
| --- | --- |
| **-b**, **--ignore-leading-blanks** | Игнорирайте водещите пропуски. |
| **-d**, **--dictionary-order** | Помислете само за бланки и букви и цифри. |
| **-f**, **--ignore-case** | Сгънете малки букви до главни букви. |
| **-g**, **--general-numeric-sort** | Сравнете според общата цифрова стойност. |
| **-i**, **--ignore-nonprinting** | Помислете само за печатните знаци. |
| **-M**, **--month-sort** | Сравнете (неизвестно) <`JAN '<... <` DEC'. |
| **-h**, **--human-numeric-sort** | Сравнете четимите от човека числа (напр. "2K", "1G"). |
| **-n**, **--numeric-sort** | Сравнете според числовата стойност на низ. |
| **-R**, **--random-sort** | Сортиране по случаен хеш на ключовете. |
| **--random-source=***FILE* | Получавайте случайни байтове от FILE. |
| **-r**, **--reverse** | Резултатът в обратен ред |
| **--sort=***WORD* | сортиране според WORD: общ-число -g, човеко-числово -h, месец -M, числово -n, случайно -R, версия -V. |
| **-V**, **--version-sort** | Естествени номера (версии) в текста. |

Други опции

|  |  |
| --- | --- |
| **--batch-size=***NMERGE* | Обединяване на най-много NMERGE входове едновременно; за повече използване на временни файлове. |
| **-c**, **--check**, **--check=***diagnose-first* | Проверка за сортиран вход; не сортирайте. |
| **-C**, **--check=***quiet*, **--check=***silent* | Подобно на -c, но не съобщава първата лоша линия. |
| **--compress-program=***PROG* | Компресиране на времеви с PROG; декомпресирайте с PROG -d. |
| **--debug** | Анотирайте частта от линията, използвана за сортиране, и предупреждавайте за съмнителна употреба на stderr. |
| **--files0-from=***F* | Четене на данни от файловете, посочени от NUL-терминирани имена във файл F; Ако F е '-', тогава прочетете имена от стандартен вход. |
| **-k**, **--key=***POS1*[,*POS2*] | Започнете клавиш на POS1 (начало 1), завършете го на POS2 (по подразбиране в края на линията). Вижте синтаксиса на ПОС по-долу. |
| **-m**, **--merge** | Обединяване на вече сортирани файлове; не сортирайте. |
| **-o**, **--output=***FILE* | Запишете резултат в FILE вместо стандартния изход. |
| **-s**, **--stable** | Стабилизиране на сортирането чрез деактивиране на последното сравнение. |
| **-t**, **--field-separator=***SEP* | Използвайте SEP вместо празна за празен преход. |
| **-T**, **--temporary-directory=***DIR* | Използвайте DIR за временни, а не $ TMPDIR или / tmp; множество опции определят множество директории. |
| **--parallel=***N* | Променете броя на сортовете, изпълнявани едновременно с N. |
| **-u**, **--unique** | С -c, проверете за строга поръчка; без -c, извежда само първия от равния пробег. |
| **-z**, **--zero-terminated** | Край на редове с 0 байт, а не с нов ред. |
| **--help** | Покажете помощно съобщение и излезте. |
| **--version** | Показване на информацията за версията и излизане. |
|  |  |

POS приема формата F [.C] [OPTS], където F е номерът на полето и C позицията на символа в полето; и двата са произход 1. Ако нито -t, нито -b е в сила, символите в полето се броят от началото на предходното празно пространство. OPTS е една или повече опции за подреждане с една буква, които отменят глобалните опции за поръчка за този ключ. Ако не е даден ключ, използвайте целия ред като ключ.

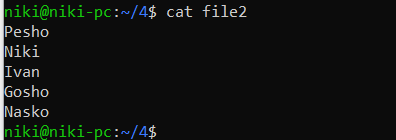
SIZE може да бъде последвано от следните мултипликационни суфикси:



И още **M**, **G**, **T**, **P**, **E**, **Z**, **Y**.

За да го разберем ще ползваме нашият пример с имената от по нагоре

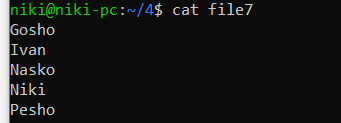
Да кажеме че искаме да ги сортираме по азбучен ред



Командата ще ни изглежда така :



И съответно изходът



Командата е доста полезна когато се налага да се сортират файлове

Друга много полезна команда е **uniq**

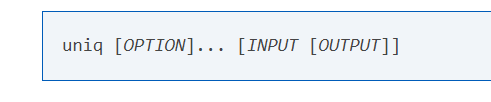
uniq филтрира съседни, съответстващи линии от входния файл INPUT, записвайки филтрираните данни към изходния файл OUTPUT.

Ако INPUT не е зададен, uniq се чете от стандартния вход.

Ако OUTPUT не е зададен, uniq записва към стандартния изход.

Ако не са зададени опции, съвпадащите линии се обединяват с първото събитие.

Синтаксисът

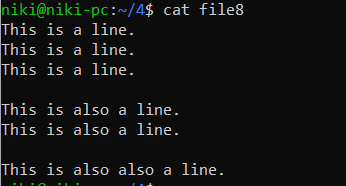


И съответно опциите

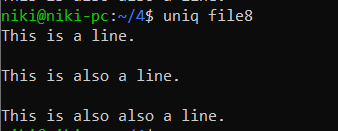
|  |  |
| --- | --- |
| **-c**, **--count** | Prefix lines with a number representing how many times they occurred. |
| **-d**, **--repeated** | Only print duplicated lines. |
| **-D**, **--all-repeated**[**=***delimit-method*] | Print all duplicate lines. *delimit-method* may be one of the following:   |  |  | | --- | --- | | **none** | Do not delimit duplicate lines at all. This is the default. | | **prepend** | Insert a blank line before each set of duplicated lines. | | **separate** | Insert a blank line between each set of duplicated lines. |   The **-D** option is the same as specifying **--all-repeated=none**. |
| **-f** *N*, **--skip-fields=***N* | Avoid comparing the first *N* fields of a line before determining uniqueness. A field is a group of [characters](https://www.computerhope.com/jargon/c/charact.htm), [delimited](https://www.computerhope.com/jargon/d/delimite.htm) by [whitespace](https://www.computerhope.com/jargon/w/whitspac.htm).  This option is useful, for instance, if your document's lines are numbered, and you want to compare everything in the line except the line number. If the option **-f 1** were specified, the adjacent lines  1 This is a line.  2 This is a line.  would be considered identical. If no **-f** option were specified, they would be considered unique. |
| **-i**, **--ignore-case** | Normally, comparisons are [case-sensitive](https://www.computerhope.com/jargon/c/casesens.htm). This option performs case-insensitive comparisons instead. |
| **-s** *N*, **--skip-chars=***N* | Avoid comparing the first *N* characters of each line when determining uniqueness. This is like the **-f** option, but it skips individual characters rather than fields. |
| **-u**, **--unique** | Only print unique lines. |
| **-z**, **--zero-terminated** | End lines with 0 byte ([NULL](https://www.computerhope.com/jargon/n/null.htm)), instead of a [newline](https://www.computerhope.com/jargon/n/newline.htm). |
| **-w**, **--check-chars=***N* | Compare no more than *N* characters in lines. |
| **--help** | Display a help message and exit. |
| **--version** | Output version information and exit. |

Да кажем че имаме файл с N на брой повратящи се редове

Нещо от типът



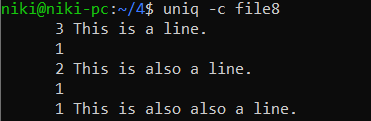
Обаче ни трябват само уникалните редове тоест без повтарящите се



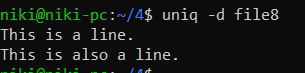
Както се вижда имаме 3 уникални реда

Съответно апо подадеме различна опция може да получиме различен резултат

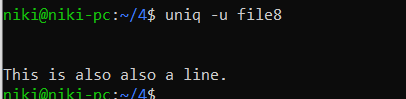
Да кажем –с тоест да ни преброи редовете



Или -d да ни покаже само дублираните редове

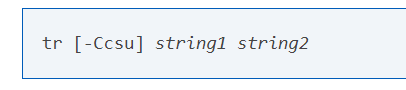


Или с –u да ни покаже само редовете които не са дублирани

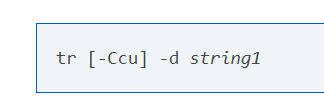


Следващата команда с която ще се запознаем е **tr**

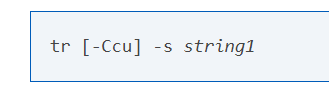
Помощната програма tr копира стандартния вход към стандартния изход със замяна или изтриване на избрани символи.



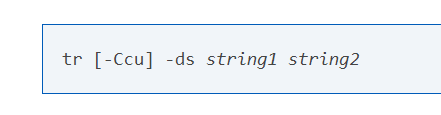
В тази форма символите в низ string1 се превеждат в символите в string2, където първият знак в string1 се превежда в първия символ в string2 и така нататък. Ако string1 е по-дълъг от string2, последният намерен в string2 дублира докато string1 не бъде изчерпан.



В тази форма символите в string1 се изтриват от входа.



В тази форма символите в string1 се компресират, както е описано за опцията -s (виж по-долу).



В четвъртата форма символите в string1 се изтриват от входа и символите в string2 се компресират, както е описано за опцията -s.

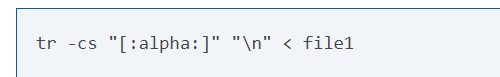
Съответно опциите са

|  |  |
| --- | --- |
| -С | Допълнете набора от символи в string1, т.е. "-C ab" включва всеки символ, с изключение на "a" и "b". |
| -с | Същото като -C, но допълва набора от стойности в string1. |
| -d | Изтриване на символи в string1 от входа. |
| -s | Стиснете многобройни появявания на символите, изброени в последния операнд (или string1 или string2) във входа на един единствен екземпляр на символа. Това се случва след като всички заличавания и преводи са завършени. |
| -u | Гарантирайте, че всеки изход е без буфер. |

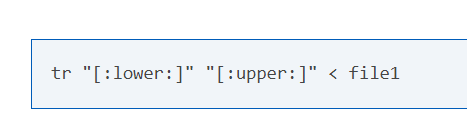
Изход

tr връща статус на изход 0, ако работи успешно, и стойност, по-голяма от нула, ако възникне грешка.

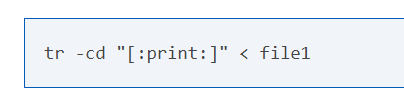
Примери



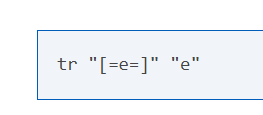
Създайте списък на думите във файл1, по един на ред, където думата се приема за максимален низ от букви.



Превод на съдържанието на файла1 в главни букви.



Премахване на всички символи, които не могат да бъдат отпечатани от file1.



Премахнете всички "диакритични" знаци от акцентираните версии на буквата e.

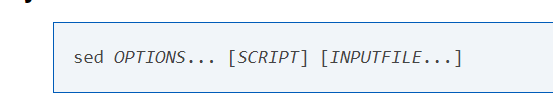
Следващият набор от команди се ползва доста по често от tr а именно **sed и awk;**

**Ще започнем с командата SED**

Oписание

sed е редактор на поток. Редактор на поток се използва за извършване на основни текстови трансформации на входен поток (файл или вход от тръбопровод). Докато в някои отношения е подобен на редактор, който позволява скриптови редакции (като ed), sed работи, като прави само едно преминаване над входа (ите) и следователно е по-ефективно. Но способността на sed е да филтрира текст в тръбопровод, което го отличава особено от други типове редактори.

Синтаксисът е :



Ако не укажете INPUTFILE или ако INPUTFILE е "-", sed филтрира съдържанието на стандартния вход. Скриптът всъщност е първият параметър без опция, който sed специално разглежда сценарий, а не входен файл, ако и само ако нито една от другите опции не указва скрипт, който да бъде изпълнен (т.е., ако нито една от -e и -f опции).

Съответно опциите са

|  |  |
| --- | --- |
| **-n**, **--quiet**, **--silent** | Потискайте автоматичния печат на пространството на шаблона. |
| **-e** *script*, **--expression=***script* | Добавете скрипт скрипта към командите, които ще се изпълняват. |
| **-f** *script-file*, **--file=***script-file* | Добавете съдържанието на скриптовия файл към командите за изпълнение. |
| **--follow-symlinks** | Следвайте символиките при обработката на място. |
| **-i**[*SUFFIX*], **--in-place**[**=***SUFFIX*] | Редактирайте файловете на място (това прави архивиране с файлово разширение SUFFIX, ако е доставено SUFFIX). |
| **-l** *N*, **--line-length=***N* | Задайте желаната дължина на линията, N, за командата "l". |
| **--POSIX** | Деактивирайте всички GNU разширения. |
| **-r**, **--regexp-extended** | Използвайте разширените регулярни изрази в скрипта. |
| **-s**, **--separate** | Разгледайте файловете като отделни, а не като един непрекъснат дълъг поток. |
| **-u**, **--unbuffered** | Зареждайте минимални количества данни от входните файлове и изпускайте по-често изходните буфери. |
| **--help** | Покажете помощно съобщение и излезте. |
| **--version** | Информация за версията на изхода и изход. |

Sed програми

Сед програмата се състои от една или повече sed команди, подадени от една или повече опции -e, -f, --expression и --file, или първия аргумент без опция, ако нито една от тези опции не се използва. Тази документация често се отнася за "скрипта sed"; това трябва да се разбира като подреждане на всички подредени скриптове и скриптове.

Командите в скрипт или скрипт-файл могат да бъдат разделени с точка и запетая (";") или с нови редове (ASCII код 10). Някои команди, поради техния синтаксис, не могат да бъдат последвани от точка и запетая, работещи като разделители на команди и следователно трябва да бъдат прекратени с нови редове или да бъдат поставени в края на скрипт или скриптов файл. Командите могат да бъдат предшествани и с незадължителни променливи знаци.

Всяка команда sed се състои от незадължителен адрес или диапазон от адреси (например номера на редове, в които се посочва коя част от файла трябва да работи; вижте Избор на линии за подробности), последвани от едносимволно име на команда и всеки допълнителен код, специфичен за командата.

Как се работи

sed поддържа два буфера за данни: активното пространство на шаблона и спомагателното пространство за задържане. И двете са първоначално празни.

sed действа чрез изпълнение на следния цикъл на всеки ред от входа: първо, sed чете един ред от входния поток, премахва всякакъв нов ред и го поставя в пространството на шаблона. След това се изпълняват команди; всяка команда може да има адрес, свързан с него: адресите са вид код на условие, а команда се изпълнява само ако условието е проверено преди командата да бъде изпълнена.

Когато се стигне до края на скрипта, освен ако не се използва опцията -n, съдържанието на пространството на шаблона се отпечатва в изходния поток, като се добавя последващата нова линия, ако е била премахната. След това започва следващият цикъл за следващия входен ред.

Освен ако не се използват специални команди (като „D“), пространството на шаблона се изтрива между два цикъла. Пространството за задържане, от друга страна, запазва данните си между циклите (вижте командите „h“, „H“, „x“, „g“, „g“, за да преместите данни между двата буфера).

Избиране на линии с sed

Адресите в sed скрипт могат да бъдат във всяка от следните форми:

|  |  |
| --- | --- |
| *number* | Задаването на номер на линия ще съответства само на този ред във входа. (Забележете, че sed преброява непрекъснато линии във всички входни файлове, освен ако не са зададени опции -i или -s.) |
| *first***~***step* | Това разширение на GNU на sed съвпада с всяка стъпка, започваща с първа линия. По-специално, линиите ще бъдат избрани, когато съществува не-отрицателно n, така че текущият номер на линията е равен на първото + (n \* стъпка). По този начин, за да се изберат нечетно номерираните линии, човек би използвал 1 ~ 2; за да избере всяка трета линия, започваща с втората, ще се използва "2 ~ 3"; за да изберете всяка пета линия, започваща с десетата, използвайте "10 ~ 5"; и "50 ~ 0" е просто друг начин да се каже 50. |
| **$** | Този адрес съвпада с последния ред от последния файл на входа или последния ред на всеки файл, когато са зададени опциите -i или -s. |
| **/***regexp***/** | Това ще избере всеки ред, който съответства на регулярния израз regexp. Ако самият regexp включва някакви символи "/", всеки трябва да бъде изваден от обратна наклонена черта ("\ t  Празният регулярен израз ‘// the повтаря последния съвпадение на регулярния израз (същото важи и ако празната регулярна формулировка се предава на командата s). Имайте предвид, че модификаторите на регулярните изрази се оценяват, когато се компилира регулярния израз, поради което е невалидно да се определят заедно с празния регулярен израз. |
| **\***%regexp%* | (% Може да бъде заменен с всеки друг единичен символ.)  Това също така съответства на регулярния израз regexp, но позволява да се използва различен разделител от "/". Тази опция е особено полезна, ако самото regexp съдържа много наклонени черти, тъй като избягва досадното избягване на всеки "/". Ако самият regexp включва никакви символи за разделяне, всяка от тях трябва да бъде изведена от обратната наклонена черта ("\ ) |
| **/***regexp***/I**  **\***%regexp%***I** | Модификаторът I за съвпадение на редовни изрази е разширение GNU, което причинява съответствие на регулярните изрази в нечувствителен към регистъра (за разлика от главни и малки букви) начини. |
| **/***regexp***/M**  **\***%regexp%***M** | Модификаторът M за съвпадение на регулярни изрази е разширение GNU sed, което кара ^ и $ да съвпадат съответно (в допълнение към нормалното поведение) празния низ след нов ред и празния низ преди нов ред. Има специални поредици от символи ("" и ""), които винаги съвпадат с началото или края на буфера. M означава многоредов. |

Ако не са дадени адреси, тогава всички линии са съпоставени; ако е даден един адрес, тогава съвпадат само линии, съответстващи на този адрес.

Обхватът на адресите може да бъде зададен чрез задаване на два адреса, разделени със запетая (","). Обхватът на адресите съвпада с линиите, започващи от мястото, където съвпада първият адрес, и продължава до втория адрес (включително).

Ако вторият адрес е regexp, тогава проверката за крайното съвпадение ще започне с реда, следващ реда, който съответства на първия адрес: обхватът винаги ще обхваща най-малко две линии (с изключение, разбира се, ако входният поток завършва).

Ако вторият адрес е число, по-малко от (или равно на) линията, съответстваща на първия адрес, тогава съвпада само един ред.

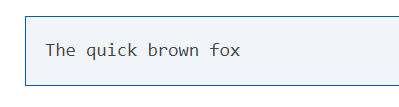
GNU sed също така поддържа някои специални форми с два адреса; всички те са разширения на GNU:

|  |  |
| --- | --- |
| **0,/***regexp***/** | Номер на линия 0 може да се използва в адресната спецификация като 0, / regexp /, така че sed да се опита да съпостави regexp в първия входен ред. С други думи, 0, / regexp / е подобно на 1, / regexp /, с изключение на това, че ако addr2 съвпада с първия ред на въвеждане, 0, / regexp / form ще го прецени като край на диапазона, докато 1, / regexp / форма ще съвпада с началото на неговия обхват и следователно ще направи диапазона до втората поява на регулярния израз.  Имайте предвид, че това е единственото място, където адресът 0 има смисъл; няма "0-та" линия, а команди, на които е даден адрес 0 по друг начин, ще дадат грешка. |
| *addr1***,+***N* | Съответства на addr1 и N реда след addr1. |
| *addr1***,~***N* | Съвпада с addr1 и редовете, следващи addr1, до следващия ред, чийто входен номер е кратен на N. |

Добавяне на! символа в края на спецификацията на адреса отрича смисъла на мача. Това е, ако! символът следва диапазон от адреси, след което ще бъдат избрани само линии, които не съответстват на обхвата на адреса. Това също работи и за единични адреси, а може би и за извратено, за нулевия адрес.

Преглед на синтаксиса на регулярното изразяване

За да знаете как да използвате sed, трябва да разбирате регулярните изрази ("regexp" за кратко). Редовният израз е шаблон, който се съпоставя с предметния низ отляво надясно. Повечето герои са обикновени: те се представят в модел и съответстват на съответните символи в темата. Като прост пример, моделът



... съответства на част от темата, която е идентична с нея. Силата на регулярните изрази идва от способността да се включат алтернативи и повторения в модела. Те са кодирани в модела чрез използването на специални символи, които не стоят сами за себе си, а се тълкуват по някакъв специален начин. Ето кратко описание на синтаксиса на регулярния израз, който се използва в sed:

|  |  |
| --- | --- |
| *char* | Един символ съвпада със себе си. |
| **\*** | Съответства на поредица от нула или повече съвпадения за предходния регулярен израз, който трябва да бъде обикновен символ, специален символ, предхождан от ",", "," групиран regexp (виж по-долу), или израз в скоба. Като GNU разширение, постфиксният регулярен израз може да бъде последван и от "\*"; например, a \*\* е еквивалентно на \*. POSIX 1003.1-2001 казва, че \* се обозначава, когато се появява в началото на редовен израз или подекспресия, но много реализации от nonGNU не поддържат това и преносимите скриптове трябва вместо това да използват "\*" в тези контексти. |
| **\+** | Подобно на \*, но съответства на едно или повече. Това е разширение GNU. |
| **\?** | Подобно на \*, но съвпада само с нула или с едно. Това е разширение GNU. |
| **\{***i***\}** | Подобно на \*, но съвпада точно с i последователностите (i е десетично цяло число; за съвместимост трябва да го държите между 0 и 255 включително). |
| **\{***i***,***j***\}** | Съвпадения между i и j, включително, последователностите. |
| **\{***i***,\}** | Съответства на повече или равно на i последователности. |
| **\(***regexp***\)** | Групира вътрешното regexp като цяло; това се използва за:  Приложете постфикс оператори, като: (abcd \*): това ще търси нула или повече цели последователности от 'abcd', докато abcd \* ще търси 'abc', последван от нула или повече от 'd'. Обърнете внимание, че поддръжката за (abcd) \* се изисква от POSIX 1003.1-2001, но много не-GNU изпълнения не я поддържат и следователно не е универсално преносима.  Използвайте обратно препратки (виж по-долу). |
| **.** | Съответства на всеки символ, включително нов ред. |
| **^** | Съответства на нулевия низ в началото на пространството на шаблона, т.е. това, което се появява след като ^ трябва да се появи в началото на пространството на шаблона.  В повечето скриптове пространството на шаблона се инициализира в съдържанието на всеки ред. Така че е полезно опростяването да се мисли за ^ # да включва като съвпадение само линии, където "#include" е първото нещо, което е на линия - ако има интервали преди, например, мачът не успее. Това опростяване е валидно, докато оригиналното съдържание на пространството на шаблона не се променя, например с команда s.  ^ действа като специален символ само в началото на регулярния израз или подекспресията (т.е. след (или).) Преносимите скриптове трябва да избягват ^ в началото на подекспресията, въпреки че POSIX позволява реализации, които третират ^ обикновен характер в този контекст. |
| **$** | То е същото като ^, но се отнася до края на пространството на шаблона. $ също действа като специален символ само в края на регулярния израз или подекспресията (т.е. преди или \ \_) и неговата употреба в края на подекспресията не е преносима. |
| **[***list***]**  **[^***list***]** | Съответства на всеки един символ от списъка: например, [aeiou] съответства на всички гласни. Списъкът може да включва последователности като char1-char2, който съответства на всеки знак между char1 и char2. Например, [b-e] съответства на някой от символите b, c, d или e.  Водещ ^ обръща смисъла на списъка, така че той съответства на всеки един символ, който не е в списъка. За да включите] в списъка, го направете първия знак (след ^ ако е необходимо); да включите - в списъка, направете го първо или последно; да включите ^ да го поставите след първия знак.  Символите $, \*,., [, И обикновено не са специални в списъка. Например, [\*] съответства на ‘или or \*, защото тук не е специално. Въпреки това, низове като [.ch.], [= A =] и [: space:] са специални в списъка и представляват съответно символи за сортиране, класове еквивалентност и класове знаци, и [следователно е специален в списъка, когато е последвано от., =, или:. Също така, когато не са в режим POSIXLY\_CORRECT, в списъка се разпознават специални прозорци като n и \ t За повече информация вж. |
| *regexp1***\|***regexp2* | Съответства на regexp1 или regexp2. Използвайте скоби, за да използвате сложни алтернативни регулярни изрази. Процесът на съвпадение се опитва на свой ред всяка алтернатива, от ляво на дясно, и се използва първият, който успява. Тази опция е разширение GNU. |
| *regexp1regexp2* | Съответства на конкатенацията на regexp1 и regexp2. Конкатенацията свързва по-плътно от \ \_, ^ и $, но по-малко плътно от другите оператори на регулярни изрази. |
| **\***digit* | Съответства на цифровата (...) издънка в скоби в регулярния израз. Тази опция се нарича обратна справка. Подекспресиите са имплицитно номерирани чрез преброяване на появата на (от ляво на дясно). |
| **\n** | Съответства на новия ред. |
| **\***char* | Съответства на char, където char е едно от $, \*,., [, Или ^. Забележете, че единствените C-подобни обратни наклонени черти, които можете да приемете за интерпретиране, са n и \ t по-специално не е преносим и съвпада с "t" при повечето реализации на sed, а не в символа за табулация. |

Обърнете внимание, че съвпадението на регулярния израз е алчен, т.е. мачовете се опитват отляво надясно и ако са възможни две или повече съвпадения, започвайки от един и същ знак, той избира най-дългия.

Например:

|  |  |
| --- | --- |
| **abcdef** | Съвпада с "abcdef". |
| **a\*b** | Съответства на нула или повече символи "a", последвани от единичен "b". Например, "b" или "aaaaaaab". |
| **a\?b** | Съвпада с "b" или "ab". |
| **a\+b\+** | Съответства на един или повече символа "a", последвани от един или повече "b" s. "ab" е най-краткото възможно съвпадение, но други примери са "aaaaab", "abbbbbb" или "aaaaaabbbbbbb". |
| **.\***or **.\+** | Всеки един от тези изрази ще съвпадне с всички символи в не празен низ, но само. \* Ще съвпадне с празния низ. |
| **^main.\*(.\*)** | Това съвпада с низ, започващ с "main", последван от отваряща и затваряща скоба. "N", "(" и ")" не е необходимо да бъдат съседни. |
| **^#** | Това съвпада с низ, започващ с "#". |
| **\\$** | Това съвпада със низ, завършващ с една обратна наклонена черта. Regexp съдържа две обратни наклонени черти за излизане. |
| **\$** | Това съвпада със низ, състоящ се от един знак за долар. |
| **[a-zA-Z0-9]** | В C локал това съвпада с всички ASCII букви или цифри. |
| **[^** *tab***]\+** | (Тук табулацията означава един символ на табулатор.) Това съвпада с низ от един или повече символи, който не съдържа интервал или раздел. Обикновено това означава една дума. |
| **^\(.\*\)\n\1$** | Това съвпада с низ, състоящ се от два еднакви подноса, разделени от нов ред. |
| **.\{9\}A$** | Това съответства на девет символа, последвани от „A“. |
| **^.\{15\}A** | Това съвпада с началото на низ, който съдържа 16 знака с последния знак „A“. |

Често използвани команди

Ако изобщо използвате sed, вероятно ще искате да знаете тези команди.

|  |  |
| --- | --- |
| **#** | (Няма разрешени адреси с тази команда.) Символът # започва коментар; коментарът продължава до следващия нов ред.  Ако сте загрижени за преносимостта, трябва да сте наясно, че някои реализации на sed (които не са POSIX съвместими) могат да поддържат само един коментар от един ред и то само когато първият символ на скрипта е #.  Предупреждение: ако първите два символа на sed скрипта са #n, то опцията -n (no-autoprint) е принудена. Ако искате да поставите коментар в първия ред на вашия скрипт и този коментар започва с буквата „n“ и не искате това поведение, тогава не забравяйте да използвате главно „N“ или да поставите поне едно място преди „n“. |
| **q** [*exit-code*] | Тази команда приема само един адрес.  Излезте от sed без обработка на повече команди или въвеждане. Обърнете внимание, че текущото пространство на шаблона се отпечатва, ако автоматичното отпечатване не е забранено с опциите -n. Възможността да се върне изходен код от sed скрипта е разширение GNU sed. |
| **d** | Изтриване на пространството на шаблона; незабавно започнете следващия цикъл. |
| **p** | Разпечатайте пространството на шаблона (към стандартния изход). Тази команда обикновено се използва само във връзка с опцията -n от командния ред. |
| **n** | Ако автоматичното отпечатване не е деактивирано, отпечатайте пространството на шаблона, след което, независимо от това, заменете пространството на шаблона със следващия ред на въвеждане. Ако няма повече входен сигнал, тогава sed излиза без обработка на повече команди. |
| **{** *commands* **}** | Група от команди може да бъде затворена между знаците {и}. Тази опция е особено полезна, когато искате група команди да се задействат от един адрес (или адресен диапазон). |

Командата

Синтаксисът на командата s (който означава "заместител") е: s / regexp / replacement / flags '. Символите / могат да бъдат равномерно заменени от всеки друг един символ в рамките на дадена команда s. Символът / или какъвто и да е друг символ, който се използва вместо него, може да се появи в regexp или replacement, само ако е предшестван от символ.

Командата s вероятно е най-важната в sed и има много различни опции. Неговата основна концепция е проста: командата s се опитва да съответства на пространството на шаблона спрямо доставения regexp; ако съвпадението е успешно, тогава тази част от пространството на шаблона, която е съчетана, се заменя с подмяна.

Заместването може да съдържа n (n е число от 1 до 9, включително) препратки, които се отнасят до частта от съвпадението, която се съдържа между n-тата (и нейното съвпадение). Също така, заместването може да съдържа неизползвани символи, които се отнасят до цялата съвпадаща част от пространството на шаблона. Накрая, като разширение GNU sed, можете да включите специална последователност, направена от обратна наклонена черта и една от буквите L, l, U, u или E. Значението е както следва:

|  |  |
| --- | --- |
| **\L** | Завъртете подмяната на малки букви, докато се намери U или E |
| **\l** | Завъртете следващия знак в малки букви |
| **\U** | Завъртете заместителя на главни, докато се намери L или E |
| **\u** | Превърнете следващия знак в главни букви |
| **\E** | Преобразуване на случай на спиране, стартирано от L или U |

За да включите литерал &, или нов ред при последната подмяна, не забравяйте да предшествате желаното &, или нов ред при замяна с \ t

Командата s може да бъде последвана от нула или повече от следните флагове:

|  |  |
| --- | --- |
| **g** | Приложете заместването на всички съвпадения на регулярното изречение, а не само на първото. |
| *number* | Заменете само числото на редовното число.  Забележка: Стандартът POSIX не уточнява какво трябва да се случи, когато смесвате модификаторите g и number, и понастоящем няма широко съгласувани значения в sed реализациите. За GNU sed взаимодействието се дефинира като: игнорира мачове преди числото, след което съвпада и заменя всички съвпадения от числото на. |
| **p** | Ако заместването е направено, след това отпечатайте новото пространство на шаблона.  Забележка: когато и двата p и e опции са посочени, относителната подредба на двете дава много различни резултати. Като цяло, ep (оценява след това печат) е това, което искате, но опериране по друг начин може да бъде полезно за отстраняване на грешки. Поради тази причина сегашната версия на GNU sed интерпретира специално наличието на опции p преди и след e, отпечатване на пространството на шаблона преди и след оценката, докато по принцип флаговете за командата s показват своя ефект само веднъж. Това поведение, макар и документирано, може да се промени в бъдещите версии. |
| **w** *file* | Ако подмяната е направена, запишете резултата в указания файл. Като разширение GNU sed се поддържат две специални стойности на файла: / dev / stderr, който записва резултата в стандартната грешка, и / dev / stdout, който записва към стандартния изход. |
| **e** | Тази команда позволява да се въвеждат данни от командната команда в пространството на шаблона. Ако е извършена подмяна, командата, която се намира в пространството на шаблона, се изпълнява и пространството на шаблона се заменя с неговия изход. Приключва се нова линия; резултатите са неопределени, ако командата за изпълнение съдържа нулев символ. Тази опция е разширение GNU sed. |
| **I**, **i** | Модификаторът I за съвпадение с регулярни изрази е разширение GNU, което прави сед съвпадение regexp по начин, нечувствителен към регистър. |
| **M**, **m** | Модификаторът M за съвпадение на регулярни изрази е разширение GNU sed, което кара ^ и $ да съвпадат съответно (в допълнение към нормалното поведение) празния низ след нов ред и празния низ преди нов ред. Има специални поредици от символи ("и"), които винаги съвпадат с началото или края на буфера. M означава многоредов. |

По-малко често използвани команди

Макар че може би по-рядко се използват от тези в предишния раздел, някои много малки, но полезни sed скриптове могат да бъдат изградени с тези команди.

|  |  |
| --- | --- |
| **y/***source-chars***/***dest-chars***/** | (Символите / могат да бъдат равномерно заменени от всеки друг единичен знак в рамките на дадена команда y.)  Транслитерирайте всички символи в пространството на шаблона, които съответстват на всеки от символите-източници със съответния знак в dest-символите.  Случаи на / (или какъвто и да е друг символ се използва вместо това), или нови редове могат да се появяват в изходните символи или списъци с дестинации, осигурявайки, че всеки екземпляр се избягва от един Изходните символи и списъците с дестинации трябва да съдържат еднакъв брой знаци (след деактивиране). |
| **a\** *text* | Като разширение GNU, тази команда приема два адреса.  Поставете ред в текста, който следва тази команда (всеки, но последният завършва с един, който се премахва от изхода), за да бъде изведен в края на текущия цикъл, или когато се чете следващия ред.  Ескейп последователностите в текста се обработват, така че трябва да използвате текстовете, за да отпечатате една обратна наклонена черта.  Като разширение GNU, ако между a и newline има различна последователност, тогава текстът на този ред, започващ от първия не-интервал след символа a, се приема като първия ред от текста. блок. (Това дава възможност за опростяване на скриптовете с едноредова добавка.) Това разширение работи и с командите i и c. |
| **i\** *text* | Като разширение GNU, тази команда приема два адреса.  Незабавно изведете редовете на текста, които следват тази команда (всяка с изключение на последния, завършващ с a, които се отстраняват от изхода). |
| **c\** *text* | Изтрийте редовете, съответстващи на адреса или адресния диапазон, и изведете редовете от текста, които следват тази команда (с изключение на последния край, който се премахва от изхода) вместо последния ред (или на мястото на линия, ако не са зададени адреси). След тази команда се стартира нов цикъл, тъй като пространството на шаблона ще бъде изтрито. |
| **=** | Като разширение GNU, тази команда приема два адреса.  Отпечатайте текущия номер на входната линия (с нов ред). |
| **l** *n* | Отпечатайте пространството на шаблона в недвусмислена форма: непечатаните символи (и символът) се отпечатват в изпъкнала форма в С-стил; дългите линии се разделят, като следният знак показва разделянето; Краят на всеки ред е маркиран с $.  n указва желаната дължина на линията-обвивка; дължина от 0 (нула) означава никога да не се увиват дългите линии. Ако е пропуснат, се използва стойността по подразбиране, посочена в командния ред. Параметърът n е разширение GNU sed. |
| **r** *file name* | Като разширение GNU, тази команда приема два адреса.  Опашете съдържанието на името на файла, което ще бъде прочетено и вмъкнато в изходния поток в края на текущия цикъл, или когато бъде прочетен следващият входен ред. Обърнете внимание, че ако името на файла не може да бъде прочетено, то се третира като празен файл, без никаква индикация за грешка.  Като разширение GNU sed, специалната стойност / dev / stdin се поддържа за името на файла, който чете съдържанието на стандартния вход. |
| **w** *file name* | Напишете пространството на шаблона в името на файла. Като разширение GNU sed се поддържат две специални стойности на името на файла: / dev / stderr, който записва резултата в стандартната грешка и / dev / stdout, който записва към стандартния изход.  Файлът ще бъде създаден (или съкратен) преди прочитането на първия ред за въвеждане; всички w команди (включително екземпляри на w флаг при успешни s команди), които се отнасят до едно и също име на файл, се извеждат без да се затваря и отваря отново файлът. |
| **D** | Ако пространството на шаблона не съдържа нов ред, започнете нормален нов цикъл, както ако командата d е издадена. В противен случай изтрийте текста в пространството на шаблона до първия нов ред и рестартирайте цикъла с полученото пространство на шаблона, без да четете нов ред на въвеждане. |
| **N** | Добавете нов ред в пространството на шаблона, след което добавете следващия ред на входа към пространството на шаблона. Ако няма повече входен сигнал, тогава sed излиза без обработка на повече команди. |
| **P** | Разпечатайте частта от пространството на шаблона до първия нов ред. |
| **h** | Заменете съдържанието на пространството за задържане със съдържанието на пространството на шаблона. |
| **H** | Добавете нов ред към съдържанието на пространството за задържане и след това добавете съдържанието на пространството на шаблона към това на пространството за задържане. |
| **g** | Заменете съдържанието на пространството на шаблона със съдържанието на пространството за задържане. |
| **G** | Добавете нов ред към съдържанието на пространството на шаблона и след това добавете съдържанието на пространството за задържане към това на пространството на шаблона. |
| **x** | Обменете съдържанието на пространствата за задържане и модела. |

Команди за sed gurus

В повечето случаи използването на тези команди показва, че вероятно е по-добре да програмирате в нещо като awk или Perl. Но понякога се ангажираме да се придържаме към sed, и тези команди могат да позволят да се пишат доста сложни скриптове.

|  |  |
| --- | --- |
| **:** *label* | [Няма разрешени адреси с тази команда.] Посочете местоположението на етикета за командите на клон. Във всички останали аспекти, no-op (без операция). |
| **b** *label* | Безусловно се разклонява към етикета. Етикетът може да бъде пропуснат, в който случай започва следващият цикъл. |
| **t** *label* | Разделяне на етикет само ако е имало успешно заместване, откакто е прочетен последният ред за въвеждане или е взето условно клонче. Етикетът може да бъде пропуснат, в който случай започва следващият цикъл. |

Команди, специфични за GNU sed

Тези команди са специфични за GNU sed, така че трябва да ги използвате внимателно и само когато сте сигурни, че скриптът няма да трябва да бъде пренасян. Те ви позволяват да проверите за GNU sed разширения или да изпълнявате задачи, които се изискват доста често, но не се поддържат от стандартните seds.

|  |  |
| --- | --- |
| **e** [*command*] | Тази команда позволява да се въвеждат данни от командната команда в пространството на шаблона. Без параметри, командата e изпълнява командата, която се намира в пространството на шаблона и замества пространството на шаблона с изхода; последващата нова линия е потисната.  Ако е зададен параметър, вместо това командата e го интерпретира като команда и изпраща своя изход към изходния поток (като r прави). Командата може да се изпълнява в няколко реда, с изключение на последния, завършващ с обратна наклонена черта.  И в двата случая резултатите са неопределени, ако командата, която трябва да се изпълни, съдържа нулев символ. |
| **F** | Разпечатайте името на файла на текущия входен файл (с нов ред). |
| **L** *n* | Това разширение GNU sed запълва и се присъединява към линии в пространството на шаблони, за да произвежда изходни линии на (най-много) n знака, подобно на fmt; ако n е пропуснато, се използва стойността по подразбиране, посочена в командния ред. Тази команда се счита за неуспешен експеримент и освен ако няма достатъчно искане (което изглежда малко вероятно) ще бъде премахнато в бъдещи версии. |
| **Q** [*exit-code*] | Тази команда приема само един адрес.  Тази команда е същата като q, но няма да отпечата съдържанието на пространството на шаблона. Подобно на q, той дава възможност да се върне изходящ код на повикващия.  Тази команда може да бъде полезна, защото единствените алтернативни начини за постигане на тази очевидно тривиална функция са използването на опцията -n (която може ненужно да усложни скрипта ви) или прибягване до следния фрагмент, който губи време, като чете целия файл без видим ефект. :  : ям #Quit тихо на последния ред: $ d # Прочетете друга линия, тихо: N # Преобразуване на пространството на шаблона всеки път, за да спестите памет: g b ям. |
| **R** *file name* | Опашка на ред от име на файл, който да бъде прочетен и вмъкнат в изходния поток в края на текущия цикъл, или когато се чете следващия ред за въвеждане. Обърнете внимание, че ако името на файла не може да бъде прочетено, или ако неговият край е достигнат, няма добавена линия, без никаква индикация за грешка.  Както и при командата r, специалната стойност / dev / stdin се поддържа за името на файла, който чете линия от стандартния вход. |
| **T** *label* | Разделяне на етикет, само ако не е имало успешни замествания, откакто е прочетен последният ред за въвеждане или е взето условно клонче. Етикетът може да бъде пропуснат, в който случай започва следващият цикъл. |
| **v** *version* | Тази команда не прави нищо, но прави сед неуспешен, ако GNU sed разширения не се поддържат, защото други версии на sed не го изпълняват. Също така можете да укажете версията на sed, която вашият скрипт изисква, като например 4.0.5. По подразбиране е 4.0, защото това е първата версия, която е изпълнила тази команда.  Тази команда позволява всички GNU разширения, дори ако POSIXLY\_CORRECT е настроен в средата. |
| **W** *file name* | Напишете на дадено име на файла частта от пространството на шаблона до първия нов ред. Всичко казано под командата w за обработването на файлове също се съдържа тук. |
| **z** | Тази команда изпразва съдържанието на пространството на шаблона. Обикновено е същата като ‘s /.\*//’, но е по-ефективна и работи в присъствието на невалидни многобайтови последователности във входящия поток. POSIX изисква тези последователности да не се съвпадат с „.“, Така че няма преносим начин за изчистване на буферите на sed в средата на скрипта в повечето многобайтови локации (включително локали UTF-8). |

GNU разширения за отпечатъци в нормални изрази

Досега (така или иначе) в този документ сме срещали само бягства от формата, например, които казват на sed да не тълкува circumflex (каретката) като специален характер, а по-скоро да го приема буквално. За друг пример ‘\* съответства на една звездичка, а не на нула или повече обратни наклонени черти.

Този раздел въвежда друг вид бягство - т.е. бягства, които се прилагат към характер или поредица от символи, които обикновено се вземат буквално, и че sed заменя със специален символ. Това осигурява начин за кодиране на непечатани символи в модели по видим начин. Няма ограничение за появата на непечатани символи в sed скрипта, но когато се подготвя скрипт в черупката или чрез редактиране на текст, обикновено е по-лесно да се използва един от следните изходни последователности от двоичния знак, който представлява:

|  |  |
| --- | --- |
| **\a** | Създава или съвпада с характер на Бел, което е "сигнал" (ASCII 7). |
| **\f** | Създава или съвпада с подаване на формуляр (ASCII 12). |
| **\n** | Създава или съвпада с нов ред (ASCII 10). |
| **\r** | Създава или съвпада с връщане на каретката (ASCII 13). |
| **\t** | Създава или съвпада с хоризонтален раздел (ASCII 9). |
| **\v** | Създава или съвпада с така наречения "вертикален раздел" (ASCII 11). |
| **\c***x* | Създава или съвпада с Control-x, където x е всеки символ. Точният ефект на "cx" е следният: ако x е малка буква, тя се преобразува в горна буква. Тогава бит 6 на символа (hex 40) се инвертира. Така cz става Hex 1A, но {c {’става Hex 3B, докато; c;’ става Hex 7B. |
| **\d***xxx* | Създава или съответства на знак, чиято десетична ASCII стойност е xxx. |
| **\o***xxx* | Създава или съответства на знак, чиято осмична ASCII стойност е xxx. |
| **\x***xx* | Създава или съответства на знак, чиято шестнадесетична ASCII стойност е xx. |

‘**\b**’ (backspace) е пропуснато поради конфликта със значението на съществуващата "дума граница".

Другите излизания съвпадат с определен клас знаци и са валидни само в регулярни изрази:

|  |  |
| --- | --- |
| **\w** | Съответства на всеки символ "дума". Символ "дума" е всяка буква или цифра или знак за подчертаване. |
| **\W** | Съответства на всеки "не-дума" характер. |
| **\b** | Съответства на граница на дума; това означава, че съответства, ако знакът отляво е символ на думата и знакът отдясно е "не-дума" характер, или обратното. |
| **\B** | Съвпада навсякъде, но на граница на думи; то е съвпадащо, ако знакът отляво и знакът отдясно са или "думата", или "не-думата" символи. |
| **\`** | Съответства само в началото на пространството на шаблона. Тази опция е различна от ^ в многоредов режим. |
| **\'** | Съответства само на края на пространството на шаблона. Тази опция е различна от $ в многоредов режим. |

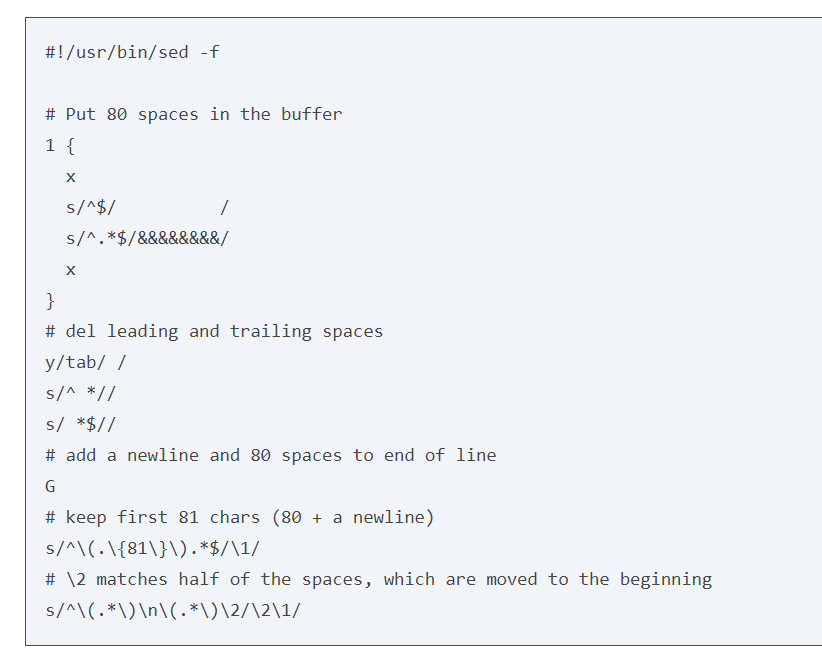
Някои примерни скриптове

Ето някои сценарии на sed, които да ви водят в изкуството на овладяването на sed.

Примерен сценарий: Центриране на линии

Този скрипт центрира всички редове на файл с ширина 80 колони. За да промените тази ширина, номерът в {... ...} трябва да бъде заменен и броят на добавените интервали също трябва да бъде променен.

Забележете как буферните команди се използват за разделяне на частите в регулярните изрази, които трябва да бъдат съчетани - това е обща техника.

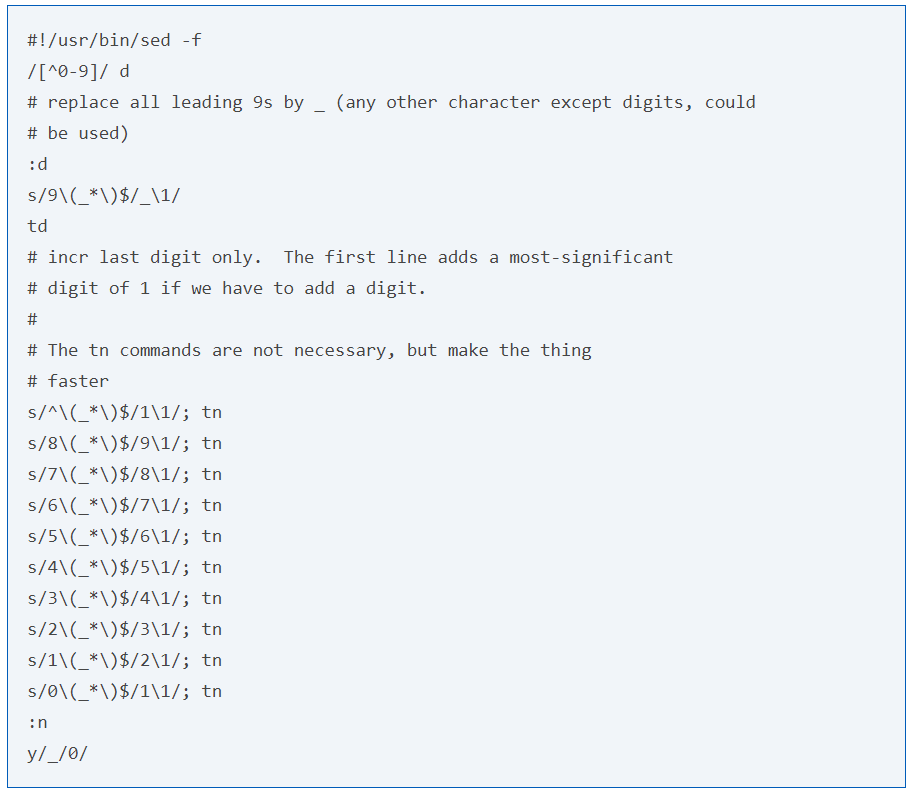


Примерен скрипт: Увеличаване на брой

Този скрипт е един от малкото, които показват как се прави аритметика в sed. Този скрипт е наистина възможен, но трябва да се извърши ръчно.

За да увеличите едно число, просто добавете 1 към последната цифра, като я замените със следната цифра. Има едно изключение: когато цифрата е девет, предишните цифри също трябва да се увеличават, докато нямате девет.

Това решение е много умно и умно, защото използва един буфер; ако нямате това ограничение, алгоритъмът, използван в номерационните линии, е по-бърз. Той работи чрез замяна на последващите деветки с долна черта, след това с помощта на няколко s команди, за да увеличите последната цифра, и след това отново да заместите долните черти с нули.



Примерен скрипт: Преименувайте файловете в долния случай

Този скрипт е доста странна употреба на sed. Преобразуваме текста и го преобразуваме в команди на черупката, след което просто ги въвеждаме в shell. Не се притеснявайте, още по-лоши хакове се правят, когато използвате sed. Скриптове дори са били написани, конвертиране на изхода на датата в bc програма ... Така че, странни неща са се случили.

Основната част от това е скриптът sed, който преначертава името от долната към горната (или обратното) и дори проверява дали пренаписаното име е същото като оригиналното име. Забележете как скриптът се параметризира с помощта на обвивните променливи и правилното цитиране.

#! /bin/sh

# rename files to lower/upper case...

#

# usage:

# move-to-lower \*

# move-to-upper \*

# or

# move-to-lower -R .

# move-to-upper -R .

#

help()

{

cat << eof

Usage: $0 [-n] [-r] [-h] files...

-n do nothing, only see what would be done

-R recursive (use find)

-h this message

files files to remap to lower case

Examples:

$0 -n \* (see if everything is ok, then...)

$0 \*

$0 -R .

eof

}

apply\_cmd='sh'

finder='echo "$@" | tr " " "\n"'

files\_only=

while :

do

case "$1" in

-n) apply\_cmd='cat' ;;

-R) finder='find "$@" -type f';;

-h) help ; exit 1 ;;

\*) break ;;

esac

shift

done

if [ -z "$1" ]; then

echo Usage: $0 [-h] [-n] [-r] files...

exit 1

fi

LOWER='abcdefghijklmnopqrstuvwxyz'

UPPER='ABCDEFGHIJKLMNOPQRSTUVWXYZ'

case `basename $0` in

\*upper\*) TO=$UPPER; FROM=$LOWER ;;

\*) FROM=$UPPER; TO=$LOWER ;;

esac

eval $finder | sed -n '

# remove all trailing slashes

s/\/\*$//

# add ./ if there is no path, only a file name

/\//! s/^/.\//

# save path+file name

h

# remove path

s/.\*\///

# do conversion only on file name

y/'$FROM'/'$TO'/

# now line contains original path+file, while

# hold space contains the new file name

x

# add converted file name to line, which now contains

# path/file-name\nconverted-file-name

G

# check if converted file name is equal to original file name,

# if it is, do not print nothing

/^.\*\/\(.\*\)\n\1/b

# now, transform path/fromfile\n, into

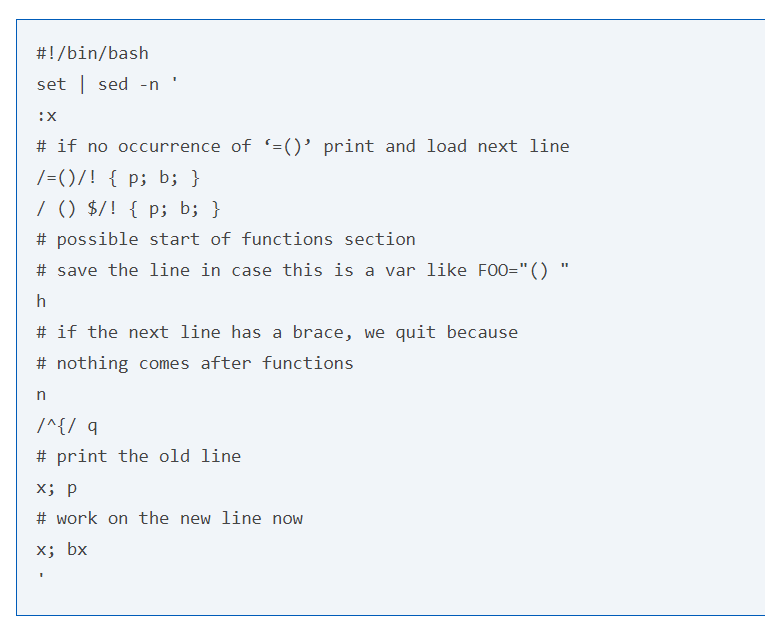
# mv path/fromfile path/tofile and print it

s/^\(.\*\/\)\(.\*\)\n\(.\*\)$/mv "\1\2" "\1\3"/p

' | $apply\_cmd

Примерен скрипт: Печат bash Околна среда

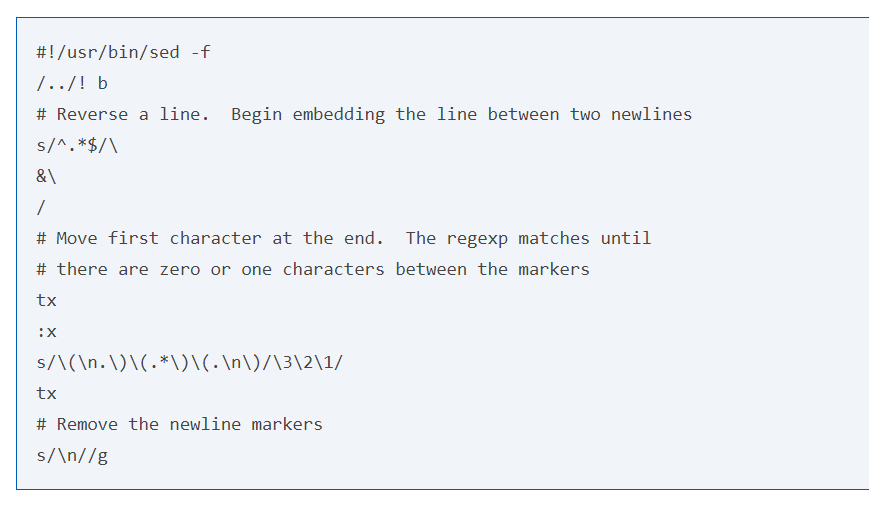
Този скрипт премахва дефиницията на черупките от изхода на командата set в Bourne-Again shell (bash).



Примерен сценарий: Обратни символи на линии

Този скрипт може да се използва за обръщане на позицията на знаците в редове. Техниката движи два символа едновременно, следователно е по-бърза от по-интуитивните реализации.

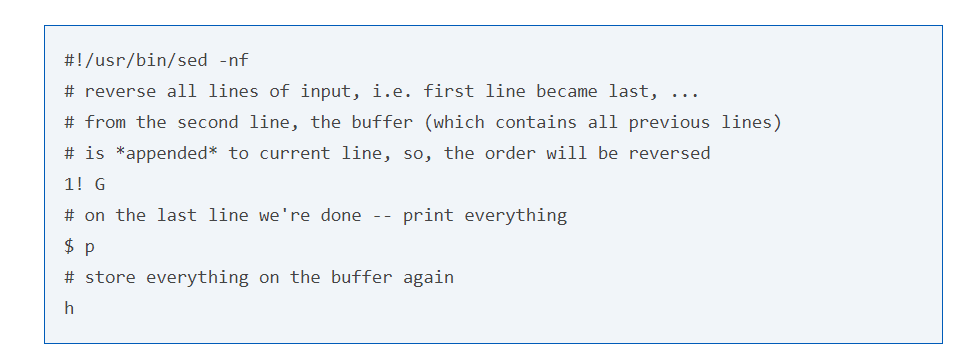
Отбележете командата tx преди дефинирането на етикета. Тази команда често е необходима за нулиране на флага, който се тества от командата t.



Примерен скрипт: Обратните линии на файлове

Този започва серия от напълно безполезни (но интересни) скриптове, които емулират различни Unix команди. Това, по-специално, е така наречената работа.

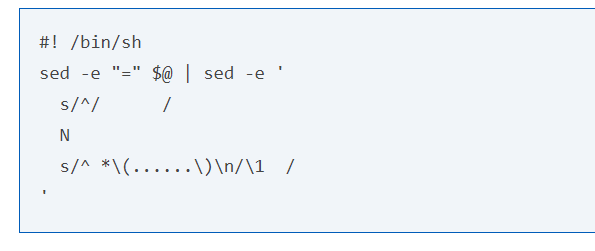
Имайте предвид, че при реализации, различни от GNU sed, този скрипт може лесно да препълва вътрешните буфери.



Примерен сценарий: Номериращи линии

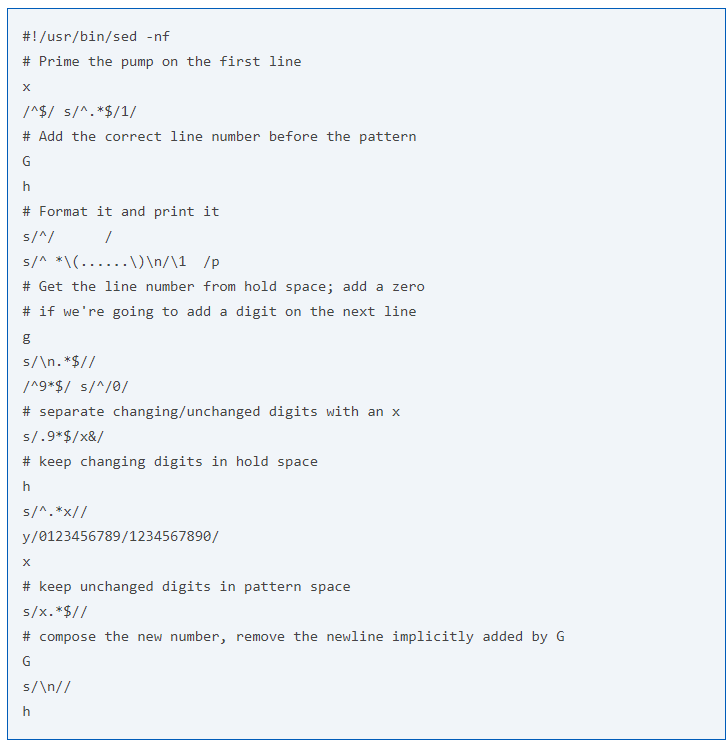
Този скрипт заменя "cat -n"; всъщност той форматира продукцията си точно както GNU cat.

Разбира се, това е напълно безполезно по две причини: първо, защото някой друг го е направил в C (командата cat), и второ, защото следният скрипт на Bourne-shell може да се използва за същата цел и ще бъде много по-бърз:



Той използва sed за отпечатване на номера на линията, след това групира по две по две с помощта на N. Разбира се, този скрипт не преподава толкова, колкото този, представен по-долу.

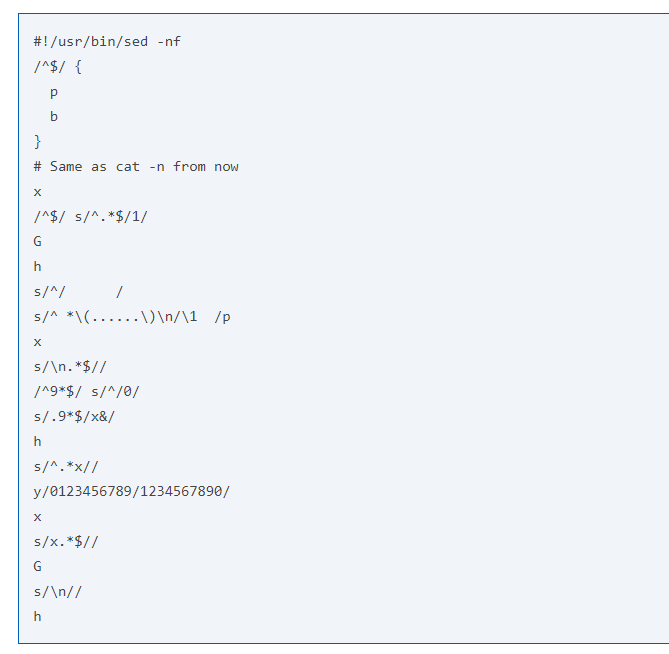
Алгоритъмът, използван за увеличаване, използва и двата буфера, така че линията се отпечатва възможно най-скоро и след това се отхвърля. Числото се разделя, така че променящите се цифри да отиват в буфер, а непроменените - в другия; променените цифри се променят в една стъпка (използвайки команда y). След това номерът на реда за следващия ред се съставя и съхранява в пространството за задържане, за да се използва в следващата итерация.



Примерен скрипт: Номериране на не-празни линии

Емулирането на „cat -b“ е почти същото като „cat -n“: трябва само да изберем кои редове да бъдат номерирани и кои не.

Частта, която е обща за този скрипт и предишната, не е коментирана, за да покаже колко е важно да се коментират правилно скриптовете sed ...



Примерен скрипт: Преброяване на знаци

Този скрипт показва друг начин да се направи аритметика със sed. В този случай трябва да добавим евентуално големи числа, така че реализирането му чрез последователни увеличения не би било осъществимо (и вероятно дори по-сложно да се измисли, отколкото този сценарий).

Подходът е да се съпоставят числата с буквите, видът на счетен принос, реализиран със sed. 'S a's са единици,'s b са десетки и така нататък: добавяме броя на знаците в текущия ред като единици, а след това разпространяваме носенето до десетки, стотици и т.н.

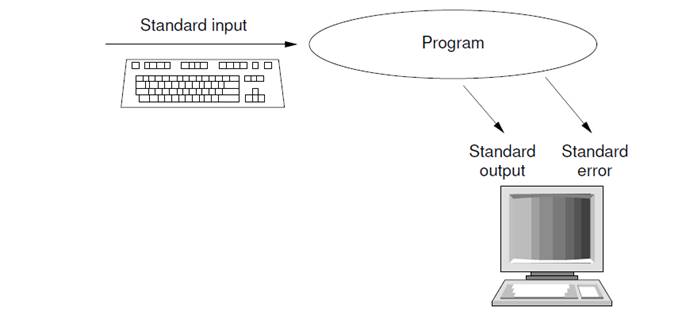
Както обикновено, текущите суми се държат в пространството за задържане.

В последния ред преобразуваме обратно формата за обратно към десетична. В името на разнообразието, това се прави с един цикъл, а не с някои 80-те команди: първо преобразуваме единици, премахвайки a's от числото; след това завъртаме буквите така, че десетки да станат, а, и така нататък, докато не останат повече букви.

**Управление на входно-изходни операции в операционна система Linux**

# **1.        Стандартни файлове**

В момента на стартирането си процесите (програмите) в Unix/Linux са свързани към три предварително отворени комуникационни канали (файлове) : стандартен входящ поток (Standard input) съкращаван като  ***stdin***, стандартен изходящ поток (Standard output) съкращаван като ***stdout*** и стандартен поток за грешки (Standard error) съкращаван като ***stderr***.



## Стандартен вход

**Стандартният вход** ***stdin*** е поток от данни (често текст), влизащи в дадена програма. По подразбиране той е свързан с клавиатурата но може да бъде свързан и със стандартния изход на друга програма. Програмата ползва тези данни чрез операция за четене. Символите въвеждани в интерактивни програми ( например текстов редактор) влизат в ***stdin.***

Не всички програми изискват използване  на ***stdin***. Например,  ***ls*** и ***echo***  могат да вземат  аргументи от командния ред, но изпълняват действието си без никакъв вход от стандартния входящ поток.

Командата ***cat*** чете информация от ***stdin***  (до срещане на символ край на потока ***Ctrl+D***) и я извежда на стандартния изход.

## Стандартен изход

***Стандартният изход*** ***stdout*** е поток, където програмата пише нейните изходни данни чрез операция за запис. Не всички програми генерират изходни данни. Например команда за преименуване на файл ***mv***  не извежда никаква информация при успех.

По подразбиране, стандартният изход е насочен към текстов терминал, който инициира програмата, но може да бъде свързан към стандартния вход на друга програма.

## Стандартен поток за грешки

Стандартният поток за грешки ***stderr*** е друг изходящ поток от данни, използван обикновено от програмите да извежда съобщение за грешки или симптоми (предупреждения) за такива. Това е поток, който е отделно от стандартния поток от данни и може да бъде пренасочван отделно. Този подход позволява разграничаване на изходящите данни и грешките. Обичайната дестинация е текстовия терминал, който стартира програмата за да се осигури най-добрата възможност да бъде видян, дори и, ако стандартния поток от данни е пренасочен.

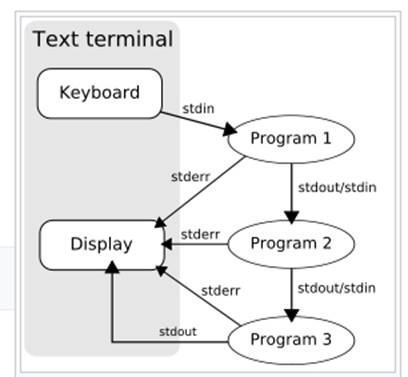
По подразбиране, стандартният изход е насочен към текстов терминал, който инициира програмата. Например- изходящият поток от данни от една програма се пренасочва във входящия на следващата програма, но грешките от всяка от двете програми отиват директно в текстовия терминал.

.Съобщенията се появяват в същия ред, в който програмата ги записва, освен ако има процес на буфериране. Например, често срещана ситуация е, когато стандартният поток от грешки не е буфериран, но стандартният изходящ поток се буферира линейно. В този случай текст, въведен по-късно към потока от грешки, може да се появи по-рано в терминала. Това се случва, ако буферирането на стандартния изходящ поток от данни все още не е завършило.

# **2.        Конвейери (Pipelines)**

Конвейерът представлява последователност от процеси които са свързани чрез техните потоци, така че изходът от единия процес ***stdout*** зарежда директно входът ***stdin*** на следващия процес.  Стандартният синтаксис за конвейер е списък от команди разделени с вертикална черта | ( наричани на кратко pipes ).

***$Program 1 | Program 2 | Program 3***



От програмите не се изисква нищо за да участват в конвейери. Те трябва просто да четат от ***stdin*** и да пишат в ***stdout.***

Например за да се изведат файловете в текущата директория (***ls***), като в списъка се включат само файловете в чиито имена има низ "key" (***grep***), и да се изведе резултата в скролираща страница  (less), трябва да бъде подадена следния pipe на терминала:

***$ls -l | grep key | less***

Програмите в конвейера работят на конкурентен принцип. Pipes са еднопосочни като посоката на обмена на данни е отляво на дясно.

Например следващият конвейр включва две команди

***$cat | grep pijo***

Стандартният входен поток на командата cat е клавиатурата, въведените от клавиатурата текстове се подават на командата  grep, която подава към стандартния си изход (екрана) сaмо текстовете съдържащи низа pijo.

# **3.        Свързване на програмите с файлове**

Стандартните потоци на програмите могат да бъдат пренасочвани към файлове.

**Стандартният входен поток *stdin*се пренасочва с „<“ към файл** от където програмата трябва да чете.

***$cat <test.txt***

Командата ползва файла ***test.txt*** като ***stdin*** , а като ***stdout*** – екрана на текстовия терминал. Командата извежда на екрана на текстовия терминал съдържанието на файла ***test.txt.***

**Стандартният входен поток *stdin*се пренасочва с „>“ към файл**. Ако файлът съществува, се презаписва.

***$cat >test1.txt***

създава файла ***test1.txt*** . Като входен поток се използва клавиатурата на текстовия терминал (завършва с ***CTRL+D)*** .

Двете пренасочвания могат да се използват едновременно.

***$cat <test.txt >test1.txt***

копира файла ***test.txt***  като ***test1.txt***.

Пренасочването на ***stdout*** с „***>>“*** добавя информацията към края на съществуващ файл. Ако файлът не съществува, той се създава.

# **4.        Пренасочване с файлови дескриптори**

Отворените файлове притежават номера наричани **файлови дескриптори**, които могат да бъдат използвани за пренасочване.

Трите стандартни файла имат следните дескриптори

***файл                                   дескриптор***

***Standard input                   0***

***Standard output                 1***

***Standard error                    2***

Дескрипторите 3-9 могат да се свържат с други файлове и се ползват обикновено в скриптове.

Пренасочването обикновено се извършва с ***stdin*** и ***stdout*** . За да бъде пренасочен друг файл файловият дескриптор трябва да се постави преди символа за пренасочване:

***$ mv tst test1 2>err***

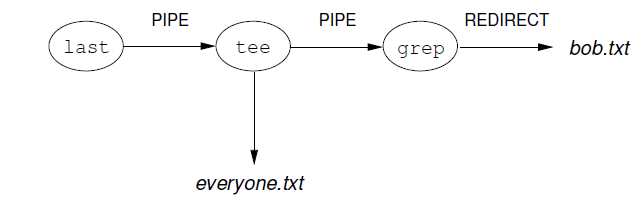
***$ cat err***

***mv: cannot stat 'tst': No such file or directory***

# **5.        Команда tee**

Командата tee прави Т-връзка в конвейер от команди. Тя копира ***stdin***си към ***stdout*** както и към файл. Така например за запазване детайлите на всички последни logins в системата (команда ***last***) в един файл, а от изхода на командата да бъдат отделени детайлите за тези на потребителя **bob** (команда ***grep***) – в друг:

***$ last | tee everyone.txt | grep bob > bob.txt***



Файлови статистики (wc);

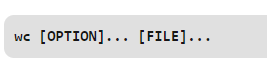
wc означава броя на думите. Както подсказва името, той се използва главно за целите на броенето.

Използва се, за да се намери броят на редовете, броят на думите, битовете и броя на символите във файловете, зададени в аргументите на файла.

По подразбиране той показва четири-колонен изход.

Първата колона показва броя на редовете във файла, втората колона показва броя на думите, които присъстват във файла, а третата - броя на символите във файла, а четвъртата колона е името на файла, дадено като аргумент. ,

Синтаксиса е :



Програмата може също да използва параметри за анализ на отделни стойности. Най-използваните са показани в таблицата по-долу:

|  |  |  |
| --- | --- | --- |
| Parameter | Long option | Value |
| -c | --bytes | Display object size in bytes. |
| -m | --count | Show the number of characters in the object |
| -l | --lines | Print the number of lines in the object. |
| -w | --words | Display the number of words in the object |

Следващата команда е AWC по подобие на SED и тя има изключително много параметри и опции с които може да се ползва.

Awk е скриптов език, използван за манипулиране на данни и генериране на отчети. Езикът за програмиране на командите awk не изисква компилиране и позволява на потребителя да използва променливи, цифрови функции, низови функции и логически оператори.

Awk е програма, която позволява на програмиста да пише малки, но ефективни програми под формата на изявления, които дефинират текстови шаблони, които трябва да се търсят във всеки ред на документ, и действието, което трябва да се предприеме, когато се намери съвпадение в рамките на линия. Awk се използва предимно за сканиране и обработка на шаблони. Той търси един или повече файлове, за да види дали те съдържат линии, които съвпадат с посочените модели и след това изпълнява свързаните с тях действия.

Awk е съкратено от имената на разработчиците - Aho, Weinberger и Kernighan.

КАКВО МОЖЕМ ДА НАПРАВИМ С AWK?

1. Операции на AWK:

(a) Сканира файл по ред

(б) Разделя всеки входен ред на полета

(c) Сравнява входните линии / полета с модела

г) Извършва действия на съответстващи линии

2. Полезно за:

а) Преобразуване на файлове с данни

б) да изготвят форматирани доклади

3. Конструктивни програми:

а) Форматиране на изходните линии

б) операции с аритметични и струнни операции

в) Условия и контури

Синтаксиса е :



Опциите са :

-f program-file: Чете източника на програмата AWK от файла

                   програма-файл, вместо от

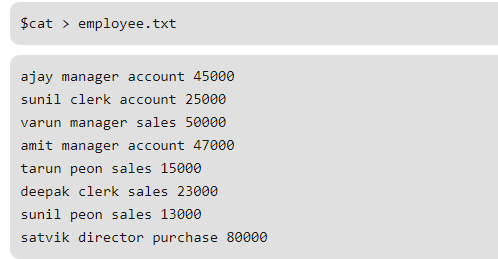
                   първия аргумент от командния ред.

-F fs: Използвайте fs за разделяне на полето за въвеждане

Примерни команди

Пример:

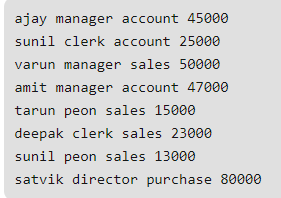
Помислете за следния текстов файл като входен файл за всички случаи по-долу.



По подразбиране поведение на Awk: По подразбиране Awk отпечатва всеки ред данни от посочения файл.



изход:

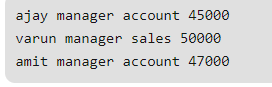


В горния пример не е даден модел. Така че действията са приложими за всички линии. Отпечатването на действие без аргумент отпечатва цялата линия по подразбиране, така че да отпечатва всички редове на файла без грешка.

2. Отпечатайте линиите, които съвпадат с дадения модел.



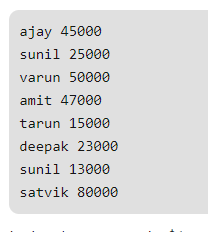
И съответно изхода



В горния пример командата awk отпечатва цялата линия, която съвпада с "мениджъра".

3. Разделяне на линия в полета: За всеки запис, т.е. линия, командата awk разделя записа, разделен с символ от празно пространство по подразбиране и го съхранява в $ n променливите. Ако редът има 4 думи, той ще се съхранява съответно в $ 1, $ 2, $ 3 и $ 4. Също така, $ 0 представлява цялата линия.





В горния пример $ 1 и $ 4 представляват съответно полета Name и Salary.

Вградени променливи в Awk

Вградените променливи на Awk включват полевите променливи - $ 1, $ 2, $ 3 и така нататък ($ 0 е цялата линия) - които разделят ред на текст на отделни думи или парчета, наречени полета.

NR: NR командата запазва текущия брой на входящите записи. Не забравяйте, че записите обикновено са линии. Командата Awk изпълнява командите шаблон / действие веднъж за всеки запис във файл.

Командата NF: NF запазва броя на полетата в текущия входен запис.

Командата FS: FS съдържа символа за разделяне на полетата, който се използва за разделяне на полета на входния ред. По подразбиране е „бяло пространство“, което означава символи за интервал и табулация. FS може да бъде пренасочена към друг символ (обикновено в BEGIN), за да се промени разделителя на полето.

Командата RS: RS съхранява символа за разделяне на текущия запис. Тъй като по подразбиране входният ред е входният запис, разделителният символ по подразбиране е нов ред.

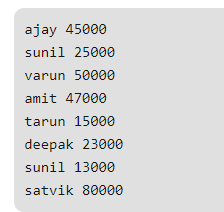
Командата OFS: OFS съхранява разделителя на изходното поле, който разделя полетата, когато Awk ги отпечатва. По подразбиране е празно пространство. Когато печат има няколко параметъра, разделени със запетаи, той ще отпечата стойността на OFS между всеки параметър.

ORS: ORS командата съхранява разделителя на изходните записи, който разделя изходните линии, когато Awk ги отпечатва. По подразбиране е символ за нов ред. печат автоматично извежда съдържанието на ORS в края на каквото и да е за печат.

Примери:

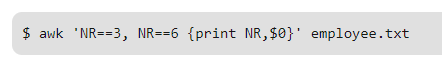
Използване на NR вградени променливи (Display Line Number)

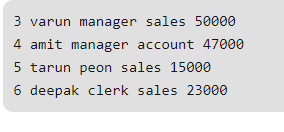




В горния пример $ 1 представлява Name, а $ NF представлява Salary. Можем да получим заплатата, като използваме $ NF, където $ NF представлява последното поле.

Друго използване на вградени променливи NR (ред от 3 до 6)



и още хиляди опции и функции които няма да разгледаме но ако желаете ги разгледайте в гугъл като напишете awc linux command with examples