SED 스트림 에디터

Ken Pizzini

도운이

최영덕

ydwolf@orgio.net

이 문서는 유닉스 표준 유틸리티인 SED의 GNU 확장판 메뉴얼을 번역한 것입니다.

고친 과정
고침 번역 2001.03.06고친이 ydwolf
한국어 번역판
고침 v3.0.21998.06.28고친이 KP
영문 버전

1. 차례

- 서론
- 시작
- SED 프로그램
  - SED에서의 행 선택 (행 지정)
  - 정규표현식 문법 개요
  - SED가 데이타를 버퍼링시키는 곳
  - 자주 쓰이는 명령어
  - 덜 자주 쓰이는 명령어들
  - 끈질긴(die-hard) SED프로그래머들을 위한 명령어들
- 몇가지 예제 스크립트들
- 행길이 (무)제한에 관해
- SED에 관한 다른 학습자료들

- 버그 리포팅

2. 서론

SED는 스트림 에디터입니다. 스트림 에디터는 입력 스트림(파일이나 파이프를 통한 입력)에 대해서 단순한 텍스트 교체가 필요할 때 쓰입니다. SED는 입력물을 딱 한 번에 더 효과적이고 능률적으로 처리해주는 다른 스크립트 편집기(ED와 같은)와 비슷합니다. 하지만 다른 에디터와는 차원이 다른 파이프를 통한 텍스트 필터링이 SED만의 탁월함입니다.

3. 시작

SED는 다음과 같은 명령행 옵션을 갖습니다.

-v
--version

S
ED의 버젼과 저작권 정보를 보여줍니다.

-h
--help

벼묻뭏뮐뭍묍뭅뮴疵É행 옵션들에 대한 간략한 정보와 버그리포팅 주소를 보여줍니다.

-n
--quiet
--silent

쩨좇쾌죙좡졌좔죵結É션을 켜두면 'p'명령이 명시될 때만 출력을 생성하고 이외의 경우에는 출력을 생성하지 않게 합니다. (디폴트로 SED는 이 옵션을 끄기때문에, 스크립트를 실행하면서 각 순환의 끝마다 패턴 공간을 출력합니다.)

-e "스크립트"
--expression="스크립트"
"
스크립트" 입력 데이타를 처리할 명령 집합에 "스크립트"를 추가합니다.

-f "스크립트파일"
--file="스크립트파일"

쩨좇쾌죙좡졌좔죵韜쨉IJB謙만┘ 낯®할 명령 집합에 "스크립트파일"에 포함된 명령어들을 추가합니다.

'-e', '-f', '--expression', '--file' 옵션이 주어지지 않을 경우에는 명령행 인자들의 옵션이 아닌 첫번째 인자가 처리 명령으로 받아들여집니다.

위의 옵션들이 처리되고 남은 명령행 인수들은 처리될 파일들의 이름으로 해석됩니다. 하이픈(-)은 표준입력스트림을 가리키며, 파일 이름이 주어지지 않았을 때에도 표준 입력 스트림으로 처리 데이타가 들어오는 것으로 간주됩니다.

4. SED 프로그램

SED프로그램은 '-e', '-f', '--expression', '--file' 옵션에 기술된 하나 이상의 명령이나 명령집합파일(또는 옵션 이외의 첫번째인자도 포함하여)에 포함된 명령어들의 집합입니다. 이 문서는 그 SED 스크립트(script : 명령이나 명령 집합 파일에 포함된 모든 것들의 순차적 연쇄적 기록)에 대해서 기술할 것입니다.

각각의 SED 스크립트는 주소와 명령어가 한쌍을 이루어 구성됩니다.(주소 혹은 주소공간들은 생략될 수 있으며, 명령어들은 관련된 코드들을 동반할 수 있습니다.)

4.1. SED에서의 행 선택 (행 지정)

'숫자'

현펩평폼폈펀펼퐈仙廢G毬じ└茱墟玖湧韜쩍뵈®림에서 그 행만 선택됩니다(SED는 연속된 입력파일들에 대해서 연속적으로 행수를 셉니다. 주의하십시요.)

'시작점~증가값'

쩨좆좨죙좡졌좔죵堅褐NU 확장은 '시작점'에서 시작하여 '증가값'만큼씩 더해진 행들을 선택합니다. 좀더 세밀하게는 "시작점+(N*증가값)=남은행수"를 만족하는 양수의 N이 존재할 때만 행의 선택이 이뤄집니다. 예를들어 "홀수"만 선택할 때는 '1~2', "두번째 행에서 시작하여 각 세번째 행"을 선택할 때는 '2~3'의 주소지정법을 사용할 수 있을 것이며, "열번째 라인부터 매 다섯번째 행"의 선택은 '10~5'입니다. 또한 '50~0'은 50번째 행을 가리키는 모호한(?) 방법입니다.

'$'

쩨좆좨죙좡졌좔죵韜쩽컥舅퓔뗍嗤렷敾뺐「®킵니다.

'/정규표현식/'

쩨좆좨죙좡졌좔죵隔痼봐ㅁ沌H治커「탠〉풔쩐錚꽁敾繭捉돔궈쳬蠻莩求墨麻 만약 정규표현식 내부에 슬래쉬(/)문자가 포함된다면 각 슬래쉬 문자는 백슬러쉬 문자(\)를 선행기술하여 이스케이프(의미 탈락)시켜야 합니다('\/'처럼.)

'\%정규표현식%'

(
'%'문자는 다른 문자 한 개로 바꾸어 쓸 수 있습니다.) 이 방식 역시 정규 표현식에 매치되는 어떤 행이라도 선택합니다만 정규 표현식 구분자로 슬래쉬(/) 대신에 다른 문자를 쓸 수 있게 허락합니다(현재 예제에서는 '%'문자.) 이 방식은 정규 표현식 내부에 슬래쉬

문자가 많이 포함되어 모든 슬래쉬 문자를 이스케이프 시키는 지루한 작업을 피하고 싶을 때 특별히 유용합니다. 만약 정규 표현식 내부에 구분자(현재 예제에서는 '%'문자)가 들어 갈 경우에는 역시 백슬러쉬(\)로 이스케이프 시켜줘야 합니다.

'/정규표현식/I'
'\%정규표현식%I'

닫녈녹놓노네녘놨鎰®자'I'의 GNU확장 수식어는 정규표현식을 case-insensitive하게(대소문자 구분없이) 적용시킵니다.

주소가 주어지지 않으면 행의 전부가 주소 공간으로 지정됩니다. 하나의 주소가 주어지면 주소와 대응되는 한 행만이 주소 공간으로 지정됩니다.

주소범위는 콤마(,)로 구분된 두 주소를 기술함으로써 지정됩니다. 주소범위는 첫번째 주소 지정부터 시작하여 두번째 주소 지정까지 계속 적용됩니다(전부 지정됨.) 만약 두번째 주소 지정이 정규표현식이라면 첫번째 주소가 적용된 행부터 시작하여 정규 표현식과 대응되는 첫 행까지가 주소공간으로 적용됩니다. 두번째 주소지정이 첫번째 주소지정보다 작거나 같은 크기의 숫자라면 단지 한 행만이 주소공간으로 적용됩니다.

느낌표(!)가 주소 기술의 마지막 문자로 오는 경우에는 적용방식이 반대로 작용합니다. 이말은 만약 느낌표(!) 문자가 주소 범위 기술 뒤에 붙으면 주소 범위와 대응되지 않는 행들만이 지정된다는 것입니다. 낱개의 주소들과 아마도 짖궂겠지만, 빈(Null) 주소에도 적용됩니다.

4.2. 정규표현식 문법 개요

'정규표현식'은 SED 뿐만 아니라 UNIX세계에서 가장 중요한 것 중 하나입니다. 하지만 어디까지를 '정규표현식'으로 볼 것인가, 혹은 '정규표준식'의 표준이라는 문제에 대해서는 의견이 분분합니다. '정규표현식'을 사용하는 프로그램마다 표현법이 조금씩 틀릴 뿐만 아니라, '확장 정규표현식'이라는 별종까지 여기저기 등장해 사뭇 혼란스럽습니다. 여기서는 SED에서 쓰이는 '정규표현식'만을 다루겠습니다. SED의 '정규표현식'을 익힌다면, 일단 어디서나 쓸 수있는 '정규표현식'의 mini-set을 익히는 것이 되겠습니다. 좀더 '정규표현식'의 세계를 느끼고 싶은 분들은 나중에 펄 문서등을 참조하시면 될 것입니다.

'정규표현식'을 간단히 말한다면, 텍스트 꾸러미에서 원하는 부분만 골라내고자 할 때 사용하는 일종의 집합(set)입니다. 몇개 되지 않으니 주욱 써보도록 하겠습니다.

'.'(점, dot)

쩽좇쵀죙좡졌좔죵ㅁ沌H治캐뻠恕【 "어떤 문자이던지 상관하지 않고 단지 한 글자"를 뜻합니다.

'*'(별, asterik)

벽묻뭏뮐뭍묍뭅뮴慕括缺鰥〕た都篇®자가 몇번이던지 연속될 때 "연속된 문자 모두"를 뜻합니다. 여기서 "몇번이던지"라는 문구에는 0번, 즉 한번도 연속되지 않은 상태도 포함된다는 것에 주의하십시요.

'^'(삿갓, carat)

현펩평폼폈펀펼퐈斂퓰쳄邦뻑戀爛求墨麻 행의 처음에 있는 문자가 아니라 행의 논리적 처음을 가리킵니다.

'$'(달러, dollar)

현펩평폼폈펀펼퐈斂풍÷뻑戀爛求墨麻 행의 마지막에 있는 문자가 아니라 행의 논리적 끝을 가리킵니다. SED는 입력 문자열에 대해서는 줄바꿈문자를 행의 끝으로 보지만, 패턴공간이나 홀드공간에 있는 데이타에 대해서는 '$'가 줄바꿈 문자가 아닌 데이타의 끝을 가리키는 것에 주의 하십시요.

'[문자들]'

"문자들"
에 포함된 "어떤 한 문자와 대응되는 한 문자"를 가리킵니다. '.'와는 달리 변역이 정해져 있는 한 문자라는 뜻입니다. 문자들이 연속된 ASCII 값을 갖는다면 '-' 문자를 써서 변역의 시작과 끝을 나타낼 수도 있습니다. 예를 들어 '[0-9]'라는 것은 0부터 시작하여 9사이에 있는 모든 문자, 즉 '어떤 숫자 하나'를 가리킵니다.

'[^문자들]'

숏상샜샹샛사샐섐胄ㅈ®자로 시작하는 변역 지정은 "문자들"에 포함되지 않은 문자 하나를 가리킵니다. 즉 전체 문자 집합에 대한 "문자들"집합의 차집합을 의미합니다.

윗목록에 포함되지 않은 텍스트들은 그냥 텍스트 자체로 해석됩니다.

정규표현식의 몇가지 예를 들어 보겠습니다.

```
wolf@zizik.net]$ nl/etc/hosts
 1        231.101.9.11    ulsan           울산  101
 2        231.102.9.11    kangbyun        강변  102
 3        231.104.9.11    world           월드  104
 4        231.107.9.11    seohyun         서현  107
 5        231.108.9.11    chungju         청주  108
 6        231.109.9.11    kuri            구리  109
 7        231.111.9.11    bupyung         부평  111
 8        231.112.9.11    juyeup          주엽  112
 9        231.113.9.11    daejun          대전  113
10        231.114.9.11    yeonsu          연수  114
11        231.115.9.11    chunan          천안  115
12        231.116.9.11    sangmu          상무  116
13        231.117.9.11    saha            사하  117
14        231.118.9.11    daegu           대구  118
15        3.7.23.107      kwanak          관악  119
16        3.7.23.232      pusan           부산  120
17        3.1.2.229       ilsan           일산  121
```

’/./’

"

어떤 문자라도 하나"라는 뜻이므로, hosts화일의 모든 텍스트와 대응됩니다.

’/.*/’

"

어떤 문자라도 0번이상 나타날 때"라는 뜻이므로, hosts화일의 모든 텍스트와 대응됩니다.

’/^2/’

"

행의 첫 문자로 온 2"를 뜻하므로 1행부터 14행의 처음 한문자와 대응됩니다.

’/2$/’

"

행의 마지막으로 온 2"를 뜻하므로 2행, 8행의 마지막 한문자와 대응됩니다.

’/[a-z]*/’

"

a부터 z까지의 문자 중에서 어떤 문자라도 0번이상 대응될 때"를 뜻하므로 각 행의 모든 영문자와 대응됩니다. (실제로는 그렇게 간단하게 대응되는 것이 아니라, "입력스트림의 a부터 z까지의 문자가 0번 출현하는 ’논리적’출현이 포함되어", 입력스트림의 모든 문자 사이의 논리적 틈새와, 모든 영어문자의 연속이 대응됩니다. ’*’의 사용에는 언제나 이런 난해함이 동반되지만, 잘 모르겠다는 분은, SED가 제법 똑똑하기 때문에 "모든 영문자"라고 생각하고 사용하시다가 나중에 혹시 문제가 생기면 다시 한번 생각해 보는 기회를 가지십시요. -_-;)

’/[^a-z]*/’

"

a부터 z까지의 문자가 아닌 것의 0번 이상 반복"이라는 뜻이므로, 각행의 ’숫자’, ’공백’, ’.’, ’한글’을 가리킵니다.

’/san/’

혀폡평폼폈펀펼퐈灣뵈® 자체인 "san"을 가리키므로 1행, 16행, 17행의 ’san’과 대응됩니다.

’/107[ ]*[^]* *[^]*/’

"

107공백0개이상공백이아닌문자0개이상공백0개이상공백이아닌문자0개이상"이라는 뜻입니다. 그러므로 4행의 "107.9.11 seohyun"과 "107" 그리고, 14행중에서 "107 kwanak 관악"이 대응됩니다. (이부분도 실제로는 굉장히 복잡하게 해석됩니다. 시간있는 분은 한번 생각해 보시길.^^)

예제에서 보았듯이 '*'문자는 대응 범위를 예측하기 어렵습니다. 정규표현식에서는 '*'문자를 최소화하고, 정확한 실제 문자를 많이 넣어주는 것이 중요합니다.

4.3. SED가 데이타를 버퍼링시키는 곳

SED는 활성된 패턴공간(pattern space)과 외부로 확장된 홀드공간(host space)이라는 두개의 데이타 버퍼를 관리합니다. 보통의 처리에서 SED는 입력 스트림으로부터 한 행을 읽어 패턴 공간에 위치시킵니다. 이 패턴 공간은 자료의 조작이 일어나는 공간입니다. 홀드 공간은 텅 빈 채로 초기화되지만 패턴 공간과 홀드 공간 양쪽으로 데이타를 이동시키는 명령어가 존재합니다.

4.4. 자주 쓰이는 명령어

SED를 쓰게 된다면 어떤 경우에도 다음 명령어들이 필요하게 될 것입니다.

'#'

(
주소 지정 없음) '#' 명령어는 주석을 시작합니다. 주석은 행의 끝까지 이어집니다. 이식성이 중요하다면, (POSIX.2 규정을 따르지않는) 몇몇 SED 구현에서는 #으로 시작되는 한 개의 한 줄짜리 문장만을 주석으로 지원한다는 것을 알아야합니다. 경고 : SED 스크립트의 시작에 나오는 문자 두 개가 만약 #n이라면 -n(자동 출력 없음)옵션이 강제 지정됩니다. 꼭 #n으로 시작되는 행이 SED 스크립트의 처음에 와야하기때문에 이 작동을 원하지 않는다면 대문자 'N'을 쓰거나 'n' 앞에 한칸 공백을 두십시요.

's/정규표현식/교체어/플래그'

(
'/' 문자는 어떤 s 명령에서도 다른 한 문자로 항상 교체할 수 있습니다.) '/' 문자(혹은 그 대신 편의에 따라 쓰인 문자)는 '정규표현식'과 '교체어' 안에 쓰일 때에는 '\'문자가 선행됐을 때에만 사용할 수 있습니다. 또 정규표현식 안에서 줄바꿈은 '\n'의 두 문자 집합으로 표현됩니다.

s 명령어는 주어진 정규표현식과 대응되는 패턴공간을 찾습니다. 대응이 성공하면 대응된 패턴 공간의 일부분은 '교체어'로 대체됩니다. '교체어'에는 '\N'(N은 1부터 9까지의 숫자)참조가 포함될 수 있습니다. '\N' 참조는 '정규표현식'의 N번째 '\('과 '\)'사이의 기술과 대응된 패턴공간의 내용을 가리킵니다. 또한 '교체어'는 '정규표현식'과 대응된 패턴공간 전체를 가리키는 '&' 문자를 포함할 수 있습니다. 그러나 '\', '&', 그리고 줄바꿈 등을 '교체어' 내부에 쓸때에는 반드시 '\'를 선행하여 '\\', '\&', '\n'처럼 기술하여야 합니다.

's' 명령어에는 다음과 같은 '플래그'들을 쓸 수 있습니다. (안 쓸 수도 있습니다.)

'g' : '정규표현식'과 대응되는 모든 패턴 공간을 교체합니다. ('처음 한 번만 교체'가 기본값입니다.)
'P' : 교체가 이뤄지면, 교체된 패턴 공간을 출력합니다.
'숫자 n' : n번째 대응되는 패턴 공간만을 교체합니다.

’w 파일’ : 교체가 이뤄지면, 교체된 패턴 공간을 파일로 출력합니다.
’I’ : (이것은 GNU 확장 기능입니다.) 정규표현식 대응을 대소문자 구분없이 작동시킵니다(case-insensi

’q’

(
한 개의 주소 지정만 쓸 수 있슴) SED가 더이상 처리없이 종료됩니다. 자동 출력 옵션을 꺼두지 않았다면 패턴 공간이 출력됨에 주의하십시요.

’d’

혁폡평폼폈펀펼퐈斤構彭@뻣窪 ㄱ構雌節척牡슥瑛謙IO뼝쳄徘爛求墨麻

’p’

혁폡평폼폈펀펼퐈斤構彭@뼙綏쪄爛求墨瑪표준출력으로.) 이 명령어는 보통, -n 명령줄 옵션이 주어졌을 때 사용됩니다. 자동 출력 옵션이 꺼지지 않은 상태에서 p 명령어가 사용되면 출력이 두번 이루어지는 몇몇 SED 구현에 주의 하십시요. 또 다른 SED 구현에서는 한 번만 출력되기도 합니다. 그러나 두 구현 모두 POSIX.2 표준에 부합되기 때문에 에러라고 생각할 수 있는 것은 아닙니다. 그러므로 이식성있는 SED 스크립트는 두 구현 모두 믿어서는 안됩니다. -n 옵션을 켜고 원할 때만 정확히 출력시키거나, p 명령을 쓰지 않아야 합니다. (이것은 s 명령의 p 플래그에도 적용됩니다.)

’n’

쩨좇쵀죙좡졌좔죵湄올綏쩔É션이 켜진 상태라면 패턴 공간을 출력하고, 패턴 공간을 무조건 다음 입력 행으로 대체합니다. 만약 다음 입력이 없다면 SED는 더이상 처리없이 종료됩니다.

’{명령어들}’
’
{’와 ’}’ 사이에 둘러싸인 명령어들을 그룹화합니다. (’}’는 주소 지정이 필요없는 명령어 문맥에 위치해야합니다.) 명령어 그룹은 하나의 주소(혹은 주소 공간) 대응이 여러개의 명령어들을 실행시키도록 할 때 특히 유용합니다(좀더 유식하게 말하면 트리거(Trigger)라고 합니다.)

4.5. 덜 자주 쓰이는 명령어들

물론 이전 장에서 설명한 명령어들보다는 덜 자주 쓰이지만, SED 스크립트에 포함되어 더욱 유용하게 쓰일 수 있는 다음과 같은 명령어들이 있습니다.

’y/대상문자목록/대치문자목록/’

(
모든 ’y’명령어의 사용에서 ’/’는 다른 하나의 문자로 대체될수 있습니다.) 대상문자목록에 들어있는 패턴공간의 모든 문자를 대상문자목록과 일대일로 위치대응되는 대치문자목록의 문자로 치환합니다. ’/’(혹은 대체된 다른 문자), ’\’, 줄넘김문자들은 ’\’로 이스케

이프 시킨 다음에 대상문자목록과 대치문자목록에 올수 있습니다. 대상문자목록과 대치문자목록은 서로 동일한 문자수를 유지해야 합니다(이스케이프용 문자'\'는 문자수에서 제외.)

'a\<줄바꿈>텍스트'

[
한 개의 주소만 지정가능] 'a\' 명령어는 다음줄에 나오는 텍스트를 큐(queue)에 저장했다가 현재 사이클이 종료되거나 다음 입력데이타가 읽힐 때 출력합니다. ('\'로 끝나는 줄은 다음 문장도 계속 출력 텍스트로 인식하며, 그 줄의 마지막에 있는 '\'는 출력되지 않습니다. [줄넘김문자 이스케이프라는 소리입니다])

'i\<줄바꿈>텍스트'

[
한 개의 주소만 지정가능] 'i\' 명령어는 다음 줄에 나오는 텍스트를 즉시 출력합니다. ('\'로 끝나는 줄은 다음 문장도 계속 출력 텍스트로 인식하며, 그 줄의 마지막에 있는 '\'는 출력되지 않습니다. [줄넘김문자 이스케이프])

'c\<줄바꿈>텍스트'

,
c\' 명령어는 대응되는 주소나 주소범위를 삭제하고, 주소의 마지막 줄 위치에(주소가 지정되지 않았다면 모든 줄의 위치에) 'c\' 명령어 다음 줄에 나오는 텍스트를 출력합니다. ('\'로 끝나는 줄은 다음 문장도 계속 출력 텍스트로 인식하며, 그 줄의 마지막에 있는 '\'는 출력되지 않습니다. [줄넘김문자 이스케이프]) 이 명령어가 실행되고 나면, 패턴공간이 삭제된 채로, 새로운 사이클이 시작됩니다.

'='

[
한 개의 주소만 지정가능] 현재 입력의 행번호를 출력합니다. (줄바꿈 문자를 단서로 합니다.)

'l'

(
영어 소문자 'l') 패턴 공간을 모호하지 않은(umambiguous) 형태로 출력합니다. 출력할수 없는 문자들은('\'문자를 포함하여) C언어스타일의 이스케이프된 형태로, 긴 문장들은 '\'문자로 구분하여 잘라서, 각 줄의 끝은 '$'문자로 출력됩니다.

'r 파일이름'

[
한 개의 주소만 지정가능] 파일의 내용을 읽어 큐(queue)에 저장했다가 사이클의 마지막에 혹은 다음 입력행이 읽혀질때, 출력 스트림에 집어 넣습니다. 만약 파일을 읽을 수 없다면 SED는 어떤 경고도 없이 단지 빈 파일이 주어진 것처럼 행동한다는 것에 주의하십시요.

'w 파일이름'

혁펩평폼폈펀펼퐈斤構彭@뼝컥臼±뭅求墨麻 파일은 첫번째 입력 행이 읽히기 전에 생성됩니다(혹은 내용이 지워집니다.) 같은 파일이름을 쓰는 모든 w 명령어들은(s명령어가 치환이 성공한 뒤 파일로 기록하게 하는 w플래그를 포함하여) 같은 파일 스트림을 통해 출력됩니다.

'D'

혁펩평폼폈펀펼퐈斤構彭@플뭐杯걍摹侮賓®자까지 지웁니다. 만약 패턴공간에 문자가 남는다면, (다음 행의 입력없이) 사이클을 다시 시작합니다. 남은 문자가 없을 시에는 정상적으로 새로운 사이클을 시작합니다.

'N'

쩽좇좨죙좡졌좔죵摹侮賓®자를 패턴 공간에 더하고 입력의 다음 줄을 읽어 패턴 스페이스에 덧붙입니다. 만약 더이상 입력이 없으면 SED 는 더이상 명령어들을 처리하지 않고 종료합니다.

'P'

혁펩평폼폈펀펼퐈斤構彭@플뭐杯걍摹侮賓®자까지만 출력합니다.

'h'

⺾弱彭@풍뼁育뼝斤構彭@풍뼁育막灌允션爛求墨麻

'H'

⺾弱彭@풍뼁肉≠摹侮賓®자를 더하고, 패턴공간의 내용을 홀드공간에 덧붙입니다.

'g'

혁펩평폼폈펀펼퐈斤構彭@풍뼁育뽁⺊弱彭@풍뼁育막灌允션爛求墨麻

'G'

혁펩평폼폈펀펼퐈斤構彭@풍뼁肉≠摹侮賓®자를 더하고, 홀드공간의 내용을 패턴공간에 덧붙입니다.

'x'

⺾弱彭0噓斤構彭@풍뼁育뻤로 교체합니다.

4.6. 끈질긴(die-hard) SED프로그래머들을 위한 명령어들

대부분의 경우, 아래의 명령어들을 쓰기보다는 펄(PERL)과 같은 다른 프로그래밍언어를 쓰는 편이 더 좋을 것입니다. 하지만 가끔 SED에 늘어붙어 꼼짝못하는 사람이나, 굉장히 복잡한 스크립트를 짜기를 좋아하는 사람도 있기 마련입니다.