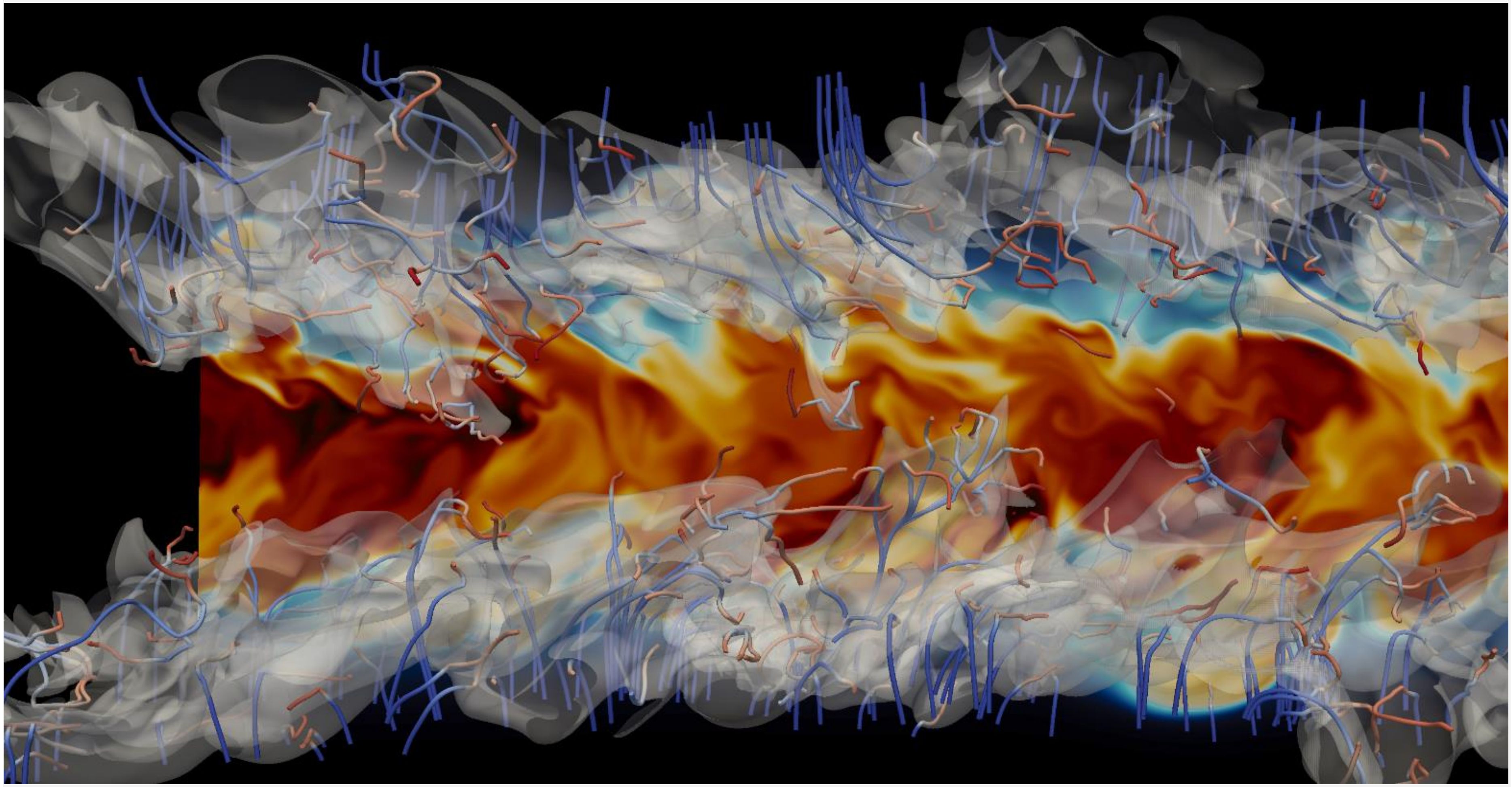
Software Tools for UNIX/Linux Systems

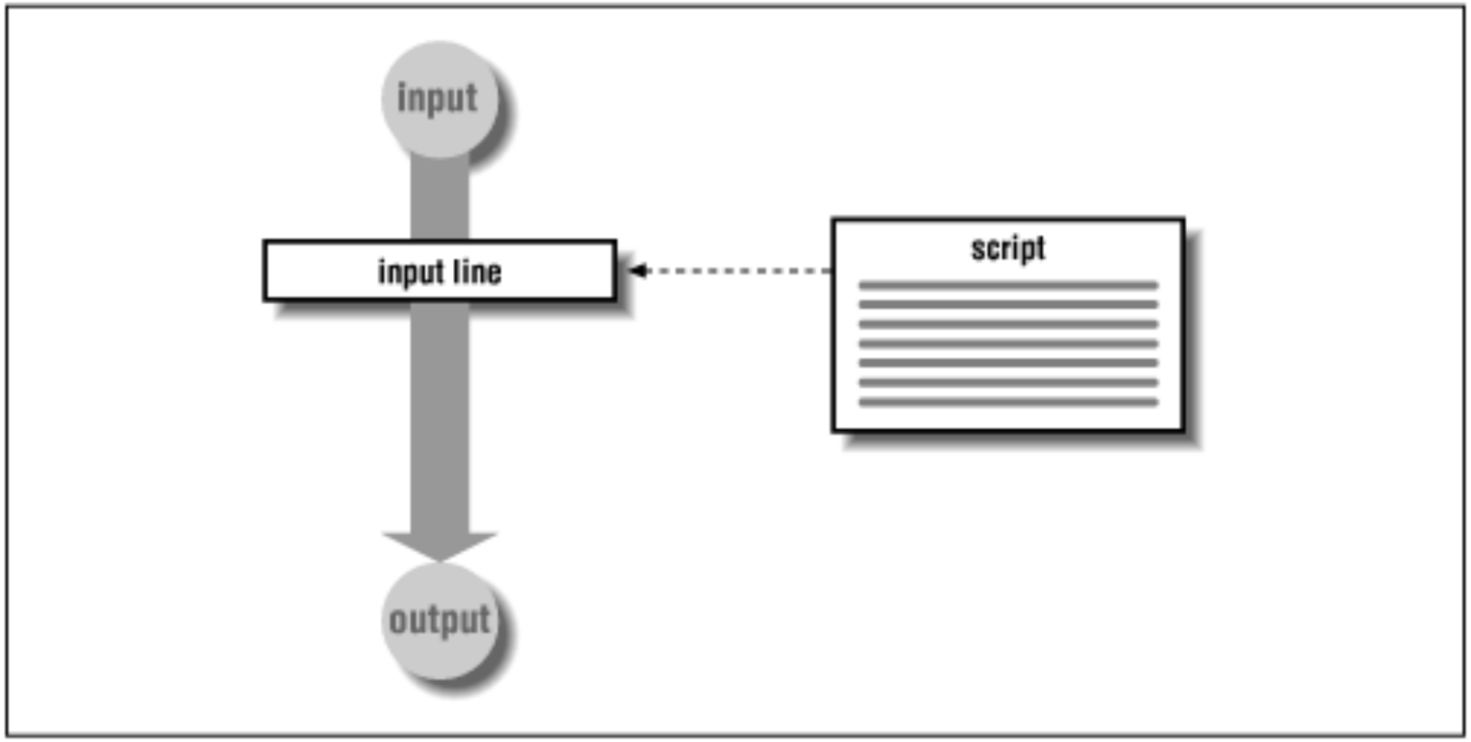
Part 6: sed & awk

C. Hasse



Introduction

How sed and awk works. Source: sed & awk, O‘Reilly



Overview

Overview

sed - basics

What ist sed ?

- **S**treamline **ed**itor
- Syntax:

  sed [options] *command* [*filenames*]
- important options:

Option	Description
-e	Editing instruction(s) follows
-f	Filename of script follows
-n	Suppress automatic output of input lines.
-i	Edit files in place

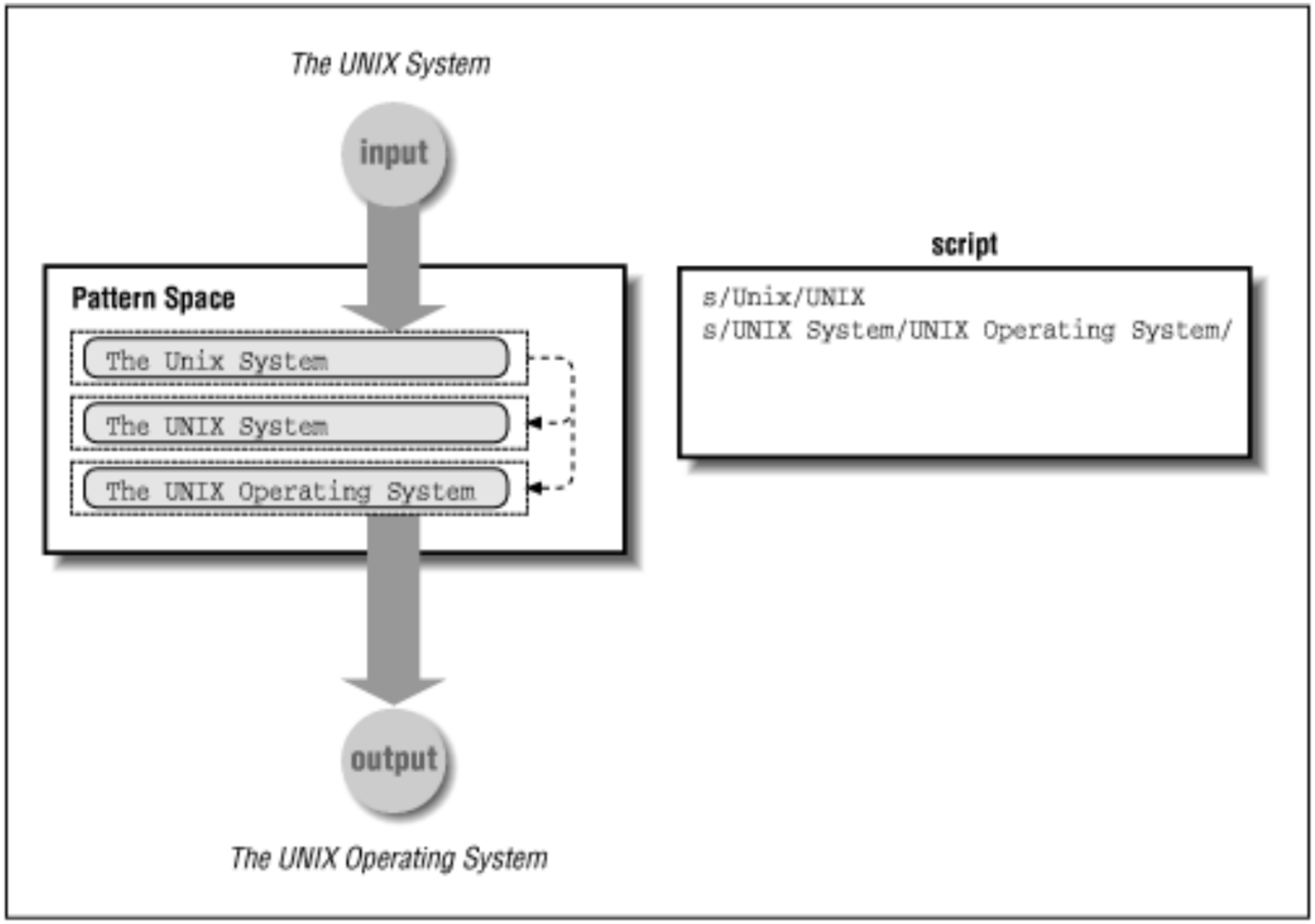
sed - basics

```
$> cat list
John Daggett, 341 King Road, Plymouth MA
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury MA
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston MA

$> sed 's/MA/Massachusetts/' list
John Daggett, 341 King Road, Plymouth Massachusetts
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury Massachusetts
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston Massachusetts
```

sed - basics

```
$> sed –n 's/MA/Massachusetts/' list

$> sed –n 's/MA/Massachusetts/p' list
John Daggett, 341 King Road, Plymouth Massachusetts
Eric Adams, 20 Post Road, Sudbury Massachusetts
Sal Carpenter, 73 6th Street, Boston Massachusetts

$> cat sedscr
s/MA/Massachusetts/p

$> sed –n –f sedscr list
John Daggett, 341 King Road, Plymouth Massachusetts
Eric Adams, 20 Post Road, Sudbury Massachusetts
Sal Carpenter, 73 6th Street, Boston Massachusetts

$> sed –n ’s!MA!Massachusetts!p' list
John Daggett, 341 King Road, Plymouth Massachusetts
Eric Adams, 20 Post Road, Sudbury Massachusetts
Sal Carpenter, 73 6th Street, Boston Massachusetts
```

sed - basics



```
s/Unix/UNIX
s/UNIX System/UNIX Operating System/
```

Changing the pattern space. Source: sed & awk, O‘Reilly

sed - basics

[address[,address]][!]command [arguments]

Address	**Description:** command applied to ...
number	… linenumber *number*
/pattern/	… lines matching *pattern*
add1,*add2*	… range of lines **between** *add1* and *add2* inclusively (not allowed for a, i, r, q and =)
address!	... all other lines **not** matching *address*

Command	**Description**
d	**Delete** pattern space
a\ *text*	Places *text* after current line (**append**)
i\ *text*	Places *text* before current line (**insert**)
c\ *text*	Replaces current line with *text* (**change**)
l	**List** pattern space, show non-printing characters as ASCII-Code
y/*abc*/*xyz*/	Transform each character by position (*a*➔*x*,*b*➔*y*,*c*➔*z*)
p	**Print** pattern space

sed - basics

Command	Description
=	Print line number
n	Print pattern space and read next line
r file	Read content of file in pattern space
w file	Write content of pattern space to file
q	Stop reading input (quit)

Substitution
- Syntax:

[address]s/*pattern*/*replacement*/*[flag]*

Flag	Description
n	Replace only at **n**-th occurrence of the *pattern* $n \in \{1, \ldots, 512\}$
g	Change **globally** on all occurrences in the pattern space
p	**Print** the contents of the pattern space
w *file*	**Write** content of pattern space to *file*

sed - basics

```
$> sed -n 's/ MA/, Massachusetts/p; s/ PA/, Pennsylvania/p' list
John Daggett, 341 King Road, Plymouth, Massachusetts
Terry Kalkas, 402 Lans Road, Beaver Falls, Pennsylvania
Eric Adams, 20 Post Road, Sudbury, Massachusetts
Sal Carpenter, 73 6th Street, Boston, Massachusetts

$> sed -n '/Plymouth/s/MA/Massachusetts/p' list
John Daggett, 341 King Road, Plymouth Massachusetts

$> sed '/MA/!d' list
John Daggett, 341 King Road, Plymouth MA
Eric Adams, 20 Post Road, Sudbury MA
Sal Carpenter, 73 6th Street, Boston MA

$> sed '2,/MA/d' list
John Daggett, 341 King Road, Plymouth MA
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston MA
```

sed - basics

```
$> sed 's!MA!Massachusetts (&)!' list
John Daggett, 341 King Road, Plymouth Massachusetts (MA)
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury Massachusetts (MA)
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston Massachusetts (MA)
$> sed -n '/MA/{
> =
> s/MA/Massachusetts/p
> }' list
1
John Daggett, 341 King Road, Plymouth Massachusetts
5
Eric Adams, 20 Post Road, Sudbury Massachusetts
8
Sal Carpenter, 73 6th Street, Boston Massachusetts
```

Overview

sed - advanced

- Three groupings:

1. Working with **multiline pattern space** (N,D,P)

2. Using the **hold space** to preserve the contents of the pattern space and make it available for subsequent commands (H,h,G,g,x)

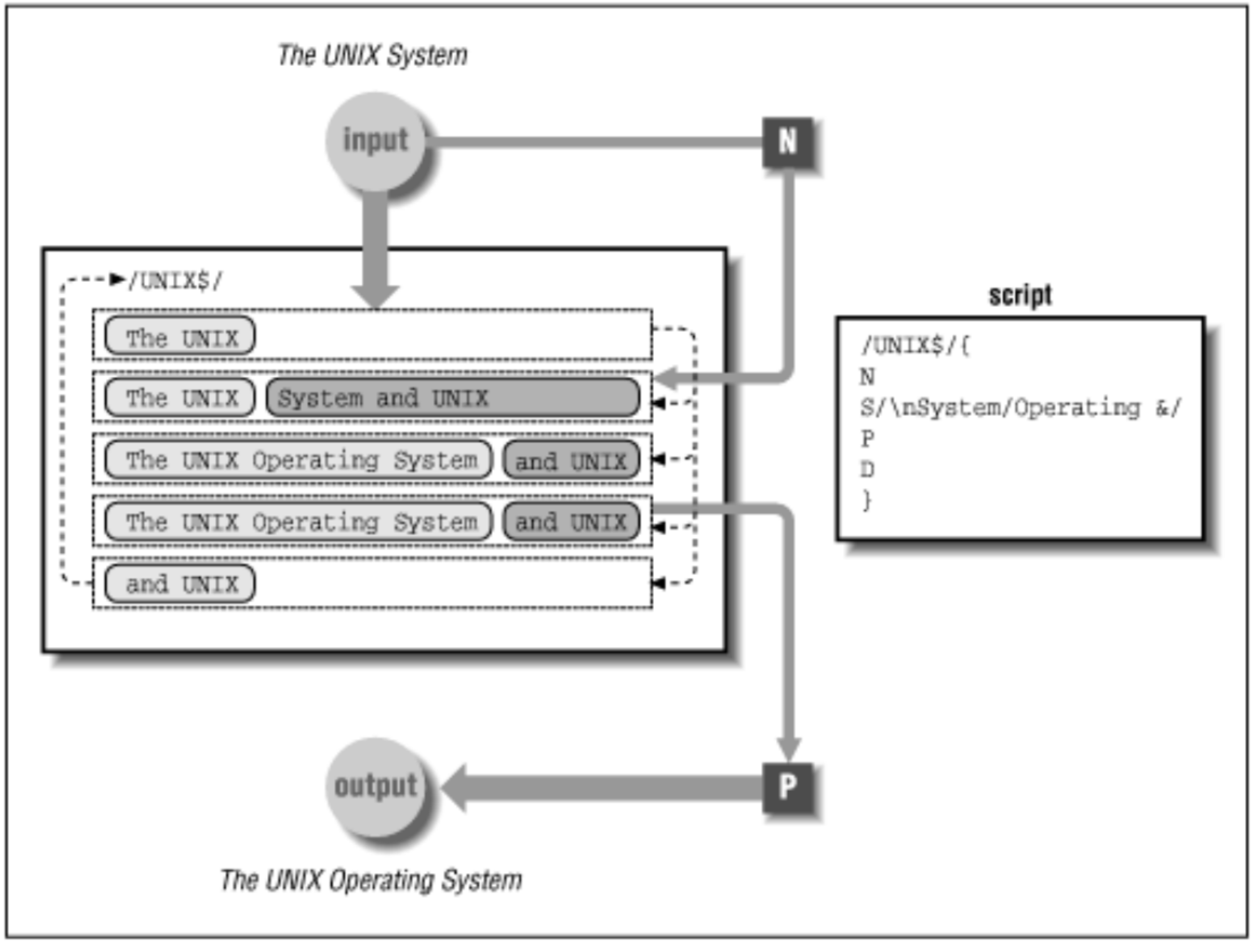
3. Writing scripts that use branching and conditional instructions to change the **flow of control** (:,b,t)

sed - advanced

Multiline pattern space

- sed has the ability to look at more than one line in the pattern space.
- The three multiline commands (N , D , P) correspond to lowercase commands (n , d , p)
- N read a new line of input and appending the pattern space
- D delete only the first line of a multiline pattern space
- P print the first line of a multiline pattern space

sed - advanced



```
/UNIX$/{
N
S/\nSystem/Operating &/
P
D
}
```

Set up an input/output loop. Source: sed & awk, O'Reilly

sed - advanced

```
$> cat multiline.txt
Consult Section 3.1 in the Owner and Operator
Guide for a description of the tape drives
available on your system.
Look in the Owner and Operator Guide shipped with your system.
Two manuals are provided including the Owner and
Operator Guide and the User Guide.
The Owner and Operator Guide is shipped with your system.
$> sed 's/Owner and Operator Guide/Installation Guide/
> /Owner/{
> N
> s/ *\n/ /
> s/Owner and Operator Guide */Installation Guide\
> /
> }' multiline.txt
Consult Section 3.1 in the Installation Guide
for a description of the tape drives
available on your system.
Look in the Installation Guide shipped with your system.
Two manuals are provided including the Installation Guide
and the User Guide.
The Installation Guide is shipped with your system.
```

sed - advanced

The Hold space

Command	Abbreviation	Function
Hold	H or h	Copy or append contents of pattern space to hold space.
Get	G or g	Copy or append contents of hold space to pattern space.
Exchange	x	Swap contents of hold space and pattern space.

Difference:

lowercase overwrites the contents of the target buffer
uppercase appends to the buffer's existing contents.

sed - advanced

```
$> cat numbers.txt
1
2
11
22

$> cat numbers.sh
#!/bin/bash
sed '/1/{h;n;};/2/G' numbers.txt

echo "####"

sed '/1/{h;d;};/2/g' numbers.txt

echo "####"

sed '/1/{h;d;};/2/G' numbers.txt
```

```
$> ./numbers.sh
1
2
1
11
22
11
####
1
11
####
2
1
22
11
```

sed - advanced

TECHNISCHE UNIVERSITÄT DARMSTADT

Advanced Flow Control Commands

- The branch (b) and test (t) commands transfer control in a script to a line containing a specified label. If no label is specified, control passes to the end of the script.
- Label
  - :mylabel
- Branching
  - [address]b[label]
- Test
  - [address]t[label]
    - It branches to a label, if a substitution has been made on the current line.

Overview

awk - basics

awk ?

- Pattern matching programming language
  - (developers: Aho, Weinberger and Kernighan)
- Syntax:
  - **awk** [options] *command* [*filenames*]
- important options:

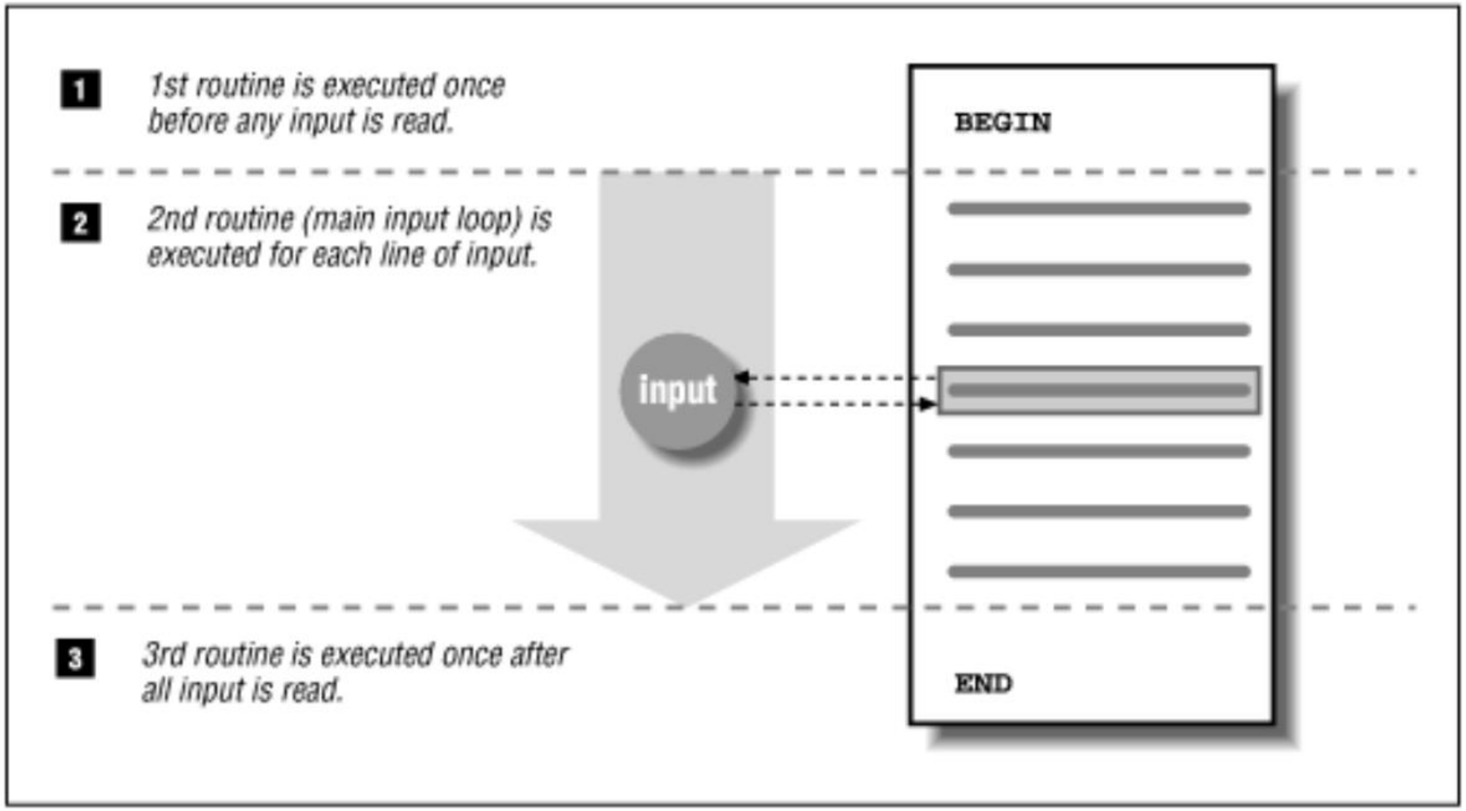
Option	Description
-f	Filename of script follows Editing instruction(s) follows
-F	Change field separator
-v	var=value follows.

awk - basics

```
$> awk '{ print $1 }' list
John
Alice
Orville
Terry
Eric
Hubert
Amy
Sal

$> awk '/MA/' list
John Daggett, 341 King Road, Plymouth MA
Eric Adams, 20 Post Road, Sudbury MA
Sal Carpenter, 73 6th Street, Boston MA

$> awk -F, '/MA/ { print $1 }' list
John Daggett
Eric Adams
Sal Carpenter

$> awk -v name=John '/MA/ {print name}' list
John
John
John
```

Overview

awk - programming and scripting


1
1st routine is executed once before any input is read.
BEGIN
2
2nd routine (main input loop) is executed for each line of input.
input
3
3rd routine is executed once after all input is read.
END

Flow and control in awk scripts. Source: sed & awk, O‘Reilly

awk - programming and scripting

Arithmetic and assignment operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo
^	Exponentiation

Operator	Description
++	Add 1 to variable
--	Subtract 1 from variable
+	Assign result of addition
-	Assign result of subtraction
*	Assign result of multiplication
/	Assign result of division
%	Assign result of modulo
^	Assign result of exponentiation

System Variables of awk:

Variable	Description
FILENAME	Current filename
FS	Field separator (default is a blank).
NF	Number of fields in current record.
NR	Number of the current record.
OFS	Output field separator (default is a blank).
ORS	Output record separator (default is a newline).
RS	Record separator (default is a newline).
$0	Entire input record.
$n	n th field in current record; Don’t confuse with shell variables
ARGCC	Number of arguments on command line.
ARGV	An array containing the command-line arguments.

awk - programming and scripting

System Variables of awk (ctd.):

Variable	Description
FNR	Like NR, but relative to the current file.
OFMT	Output format for numbers (default is %.6g).
RSTART	First position in the string matched by match function.
RLENGTH	Length of the string matched by match function.
SUBSEP	Separator character for array subscripts (default is \034).
ENVIRON	An associative array of environment variables.

awk - programming and scripting

```
$> echo a b c d | awk 'BEGIN { one = 1; two = 2 }
> { print $(one+two) } '
c

$> cat grades
john 85 92 78 94 88
andrea 89 90 75 90 86
jasper 84 88 80 92 84

$> cat grades.awk
# average five grades
{ total = $2 + $3 + $4 + $5 + $6
avg = total / 5
print $1, avg }

$> awk -f grades.awk grades
john 87.4
andrea 86
jasper 85.6
```

awk - programming and scripting

```
$> cat checkbook.test
1000
125  Market  -125.45
126  Hardware Store  -34.95

$> cat checkbook.awk
#checkbook.awk
BEGIN { FS = "  "; OFS = "\t" }
#1 Expect the first record to have the starting balance.
NR == 1 { print "Beginning Balance: \t" $1
        balance = $1
        next          # get next record and start over
}
#2 Apply to each check record, adding amount from balance.
{     print $1, $2, $3
      print balance += $3 #check have negative amounts
}

$> awk –f checkbook.awk checkbook.test
Beginning Balance:    1000
125   Market      -125.45
874.55
126   Hardware Store -34.95
839.6
```

awk - programming and scripting

Passing parameters into a script (two ways):

```
$> cat test
line one
line two

$> cat test2
first line

$> awk 'BEGIN {print n}
> {
> if (n==1) print "Read first file"
> if (n==2) print "Read second file"
> }' n=1 test n=2 test2

Read first file
Read first file
Read second file
```

```
$> awk -v n=1 'BEGIN {print
n}
> {
> if (n==1) print "Read first file"
> if (n==2) print "Read second
file"
> }' test n=2 test2
1
Read first file
Read first file
Read second file
```

awk - programming and scripting

Relational boolean operators

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to
~	Matches
!~	Does not match

Operator	Description
||	Logical OR
&&	Logical AND
!	Logical Not

Formated printing with **printf**, same usage as in C

awk - programming and scripting

Conditional Statements:

- **if** (expression) action1
  [**else** action2]
- **if** (expression) action1; [**else** action2]
- **if** (expression) {
    action1
    action2
  }

Conditional Operator:

- expr **?** action1 **:** action2

awk - programming and scripting

Loops:

- while loop:

  while (expression)

  action

- do while loop:

  do

  action

  while (expression)

- for loop:

  for (set_counter; test_counter; change counter) *action*

awk - programming and scripting

Change flow control of ...

- ... loops:
  - **break** no more iterations of the loop are performed
  - **continue** starts a new iteration at the top of the loop
- … main input loop:
  - **next** causes the next line of input to be read and go back to top of the script.
  - **exit** exits the main input loop and passes control to the END rule

awk - programming and scripting

```
$> cat factorial.sh
#!/bin/bash
awk ' BEGIN {
        printf("Enter number: ")
}
$1 ~ /[0-9]+/ {
        number = $1
        fact = number
        for (x = number - 1 ; x > 1; x--)
                fact *= x
        printf("The factorial of %d is %g\n", number, fact)
        exit
}
$1 !~ /[0-9]+/ { printf("\nInvalid entry. Enter a number: ") }' –

$> ./factorial.sh
Enter number: 4
The factorial of 4 is 24
```

awk - programming and scripting

Arrays:

- all arrays are *associative* arrays
- Syntax: array[subscript]=value
- Loops over arrays:
  - »**for** (var **in** array)
  - »   do something with array[var]
- Testing for membership:

  if (var **in** array) …
- Deleting Elements:

  delete array[subscript]
- “Multidimensional” arrays:

  array[i,j]=value   produces a 1D array with subscript “iSUBSEPj”

awk - programming and scripting

```
$> cat grades.awk
# grades.awk -- average student grades, determine
# letter grade as well as class averages.
# $1 = student name; $2 - $NF = test scores.
# set output field separator to tab.
BEGIN { OFS = "\t" }
{
  # add up grades
  total = 0
  for (i = 2; i <= NF; ++i)
        total += $i
  # calculate average
  avg = total / (NF - 1)
  # assign student's average to element of array
  class_avg[NR] = avg
  # determine letter grade
  if (avg >= 90)  grade="A"
  else if (avg >= 80) grade="B"
  else if (avg >= 70) grade="C"
  else if (avg >= 60) grade="D"
  else grade="F"
  # increment counter for letter grade array
  ++class_grade[grade]
  # print student name, average and letter grade
  print $1, avg, grade
…
```

```
…
}
END {
# calculate class average
for( x = 1; x <= NR; x++)
    class_avg_total += class_avg[x]
    class_average =  class_avg_total / NR
# determine how many above/below average
for( x = 1; x <= NR; x++)
    if (class_avg[x] >= class_average)
        ++above_average
    else
        ++below_average
# print results
print ""
print "Class Average: ", class_average
print "At or Above Average: ", above_average
print "Below Average: ", below_average
# print number of students per letter grade
for (item in class_grade)
    print item ":", class_grade[item]
}
```

awk - programming and scripting

```
$> cat grades.test
mona 70 77 85 83 70 89
john 85 92 78 94 88 91
andrea 89 90 85 94 90 95
jasper 84 88 80 92 84 82
dunce 64 80 60 60 61 62
ellis 90 98 89 96 96 92
```

```
$> awk -f grades.awk grades.test
mona	79	C
john	88	B
andrea	90,5	A
jasper	85	B
dunce	64,5	D
ellis	93,5	A

Class Average:	83,4167
At or Above Average:	4
Below Average:	2
A:	2
B:	2
C:	1
D:	1
```

awk - programming and scripting

Aritmetic functions

Function	Description
cos(x)	Returns cosine of x (x is in radians).
exp(x)	Returns e to the power x .
int(x)	Returns truncated value of x.
Log(x)	Returns natural logarithm (base- e) of x .
sin(x)	Returns sine of x (x is in radians).
sqrt(x)	Returns square root of x .
Atan2(y,x)	Returns arctangent of y / x in the range $-\pi$ to π.
rand()	Returns pseudo-random number r , where 0 <= r < 1.
srand(x)	Establishes new seed for rand() . If no seed is specified, uses time of day. Returns the old seed.

TECHNISCHE UNIVERSITÄT DARMSTADT

String functions

Function	Description
gsub (r , s , t)	Globally substitutes s for each match of the regular expression r in the string t . Returns the number of substitutions. If t is not supplied, defaults to $0 .
index (s , t)	Returns position of substring t in string s or zero if not present.
length (s)	Returns length of string s or length of $0 if no string is supplied.
match (s , r)	Returns either the position in s where the regular expression r begins, or 0 if no occurrences are found. Sets the values of RSTART and RLENGTH .
split (s , a , sep)	Parses string s into elements of array a using field separator sep ; returns number of elements. If sep is not supplied, FS is used. Array splitting works the same way as field splitting.
sprintf (" fmt ", expr)	Uses printf format specification for expr .
sub (r , s , t)	Substitutes s for first match of the regular expression r in the string t . Returns 1 if successful; 0 otherwise. If t is not supplied, defaults to $0 .
substr (s , p , n)	Returns substring of string s at beginning position p up to a maximum length of n . If n is not supplied, the rest of the string from p is used.
tolower (s)	Translates all uppercase characters in string s to lowercase and returns the new string.
toupper (s)	Translates all lowercase characters in string s to uppercase and returns the new string.

awk - programming and scripting

```
$> awk ‘BEGIN {
nextletter=sprintf("%c”,98)
> printf("ASCII of 98 is %s\n”,nextletter)}’
ASCII of 98 is b

$>  awk 'BEGIN { t="a,b,c,d"
> number=split(t,array,",")
> for (i=1;i<=number;++i)
> printf("array[%i]=%s\n",i,array[i])
> number=gsub(",",";",t)
> printf("new t is %s, #subs =
%i\n",t,number)
> number=sub(";",",",t)
> printf("new t is %s, #subs =
%i\n",t,number)
> pos=index(t,",")
> printf("pos of , in t is %i\n",pos)}'
array[1]=a
array[2]=b
array[3]=c
array[4]=d
new t is a;b;c;d, #subs = 3
new t is a,b;c;d, #subs = 1
pos of , in t is 2
```

```
$> awk 'BEGIN { t="a,b,c,d”
> num=length(t)
> printf("length of t is %i\n",num)
> num=match(t,",b")
> printf("pos of ,b in t is %i\n",num)
> printf("pos of ,b in t is %i and has a  length
of% i\n",RSTART,RLENGTH)
> substring=substr(t,2,4)
> printf("string from pos=2 and a length of 4
is      %s\n",substring)
> s=toupper(t);
> print s
> t=tolower(s);
> print t}'
length of t is 7
pos of ,b in t is 2
pos of ,b in t is 2 and has a length of 2
string from pos=2 and a length of 4 is ,b,c
A,B,C,D
a,b,c,d
```

awk - programming and scripting

- Writing your own functions
  - Syntax:
    - **function** name (parameter1,...) {
    - statements
    - [**return** expression]

- **getline** function:
  - similar to **next**: **getline** gets the next line without changing control in the script.
  - e.g.:

getline < "*data*" reading input from file *data*

getline < "**-**"reading input from stdin

"*cmd*" **| getline** reading input from a pipe

getline *var* assigning input to variable *var*

awk - programming and scripting

- **close()** function
  - allows to close open files and pipes
  - Syntax:
    - **close(“*cmd*“)**
- **system**() function
  - executes a supplied command
  - returns only exit status not the output
  - The script waits for the command to finish before continuing execution.
  - Syntax:
    - **system(“*cmd*”)**
- Directing output to files and pipe
  - e.g.:

print > “*data*“	write current record to file *data*
print >> “*data*“	append file *data* with current record
print | “*cmd*“	sends current record to a command

awk - programming and scripting

Exercise:

Let‘s program and play battleship :P