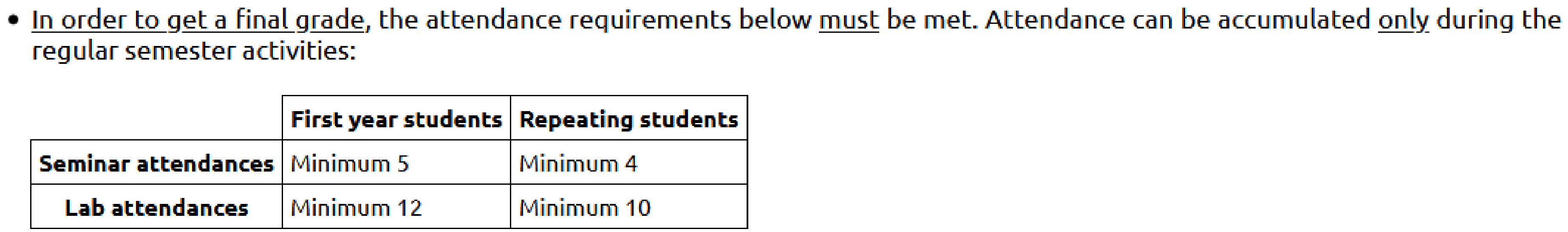
Cerinte prezenta



- In order to get a final grade, the attendance requirements below must be met. Attendance can be accumulated only during the regular semester activities:

	First year students	**Repeating students**
Seminar attendances	Minimum 5	Minimum 4
Lab attendances	Minimum 12	Minimum 10

Linia de comanda

The command structure

a. Space is separator

b. First word is the command

c. Next words are arguments

i. Values: ls /etc

ii. Options

1. Short form: ls -l

2. Long form: ls --all

3. Short form with value: cut –d : –f 1,2,3 /etc/passwd

4. Long form with value: cut –delimiter=: –fields=1,2,3 /etc/passwd

Expresii regulate

- Every character that appears in a regular expression can have two meanings, normal or special, depending on the escape character \ appearing in front of it.
- Depending on the program used to process the regular expressions, a characters special meaning is achieved with or without escaping it. Below are the meanings as required by the programs we will use for this class.

Expresii regulate

.	Matches any single character
\	Escape, changes the meaning of the character following it, between normal and special
[abc]	Matches any single character that appears in the list (this case: a or b or c)
[a-z]	Matches any single character that belongs to the range (in this case any lower-case letter)
[^0-9]	Matches any single character that does not belong to the range (in this case anything that is not a digit)
^	Beginning of line
$	End of line

Expresii regulate

\<	Beginning of word
\>	End of word
\(\)	Group several characters into an expression
*	Previous expression zero or more times
\+	Previous expression one or more times
\?	Previous expression zero or one times
\{m,n\}	Previous expression at least m and at most n times
\\|	Logical OR between parts of the regular expression

Expresii regulate

examples for the rules

a. `.*` – any sequence of characters
b. `[a-zA-Z02468]` – any letter, regardless of its case, and any even digit
c. `[ ]` – space
d. `^[^0-9]\+$` – non-empty lines containing any characters except digits
e. `\([Nn][Oo] \)\+` - any refusal, no matter how insistent (eg No no no no no)

Grep

1. Let’s search for things in file /etc/passwd

    a. Display all lines containing “dan”. The solution is below

    i. `grep “dan” /etc/passwd`

    b. Display the line of username “dan”. The username is the first field on the line, it is not empty, and it ends at the first :. We will rely on these aspects to insure that we only search the usernames, and not anything else.

    i. `grep –i “^dan:” /etc/passwd`

    c. Display the lines of all users who do not have digits in their username.

    i. `grep “^[^0-9]\+:” /etc/passwd`

Grep

d. Display the lines of all users who have at least two vowels in their username. This is a little tricky, because the vowels do not need to be consecutive, so we need to allow for any characters between the vowels (including none), but we cannot allow : to appear between vowels, or else we would be searching outside the username.

   i. `grep -i "^[^:]*[aeiou][^:]*[aeiou][^:]*:" /etc/passwd`

   ii. `grep -i "^[^:]*\([aeiou][^:]*\)\{2,\}:" /etc/passwd`

e. There will be lots of users displayed for the problem above, so let's search for usernames with at least 5 vowels in their username. The first solution above will be really long for this case, but the second will be very easy to adapt, by changing 2 into 5.

   i. `grep -i "^[^:]*\([aeiou][^:]*\)\{5,\}:" /etc/passwd`

Grep

f. Display the lines of all the users not having bash as their shell. The shell is the last value on the line, so we will use that when searching.

i. `grep -v "/bash$" /etc/passwd`

g. Display the lines of all users named Ion. We will have to search in the user-info field (the fifth field) of each line, ignore the upper/lower case of the letters, and insure that we do not display anybody containing the sequence “ion” in their names (eg Simion, Simionescu, or Ionescu).

i. `grep -i "^\([^:]*:\)\{4\}[^:]*\<ion\>" /etc/passwd`

Grep

2. Let’s consider a random text file a.txt, and search for things in it

    a. Display all the non-empty lines

    i. `grep “.” a.txt`

    b. Display all the empty lines

    i. `grep “^$” a.txt`

    c. Display all lines containing an odd number of characters

    i. `grep “^\(..\)*.$” a.txt`

    d. Display all lines containing an ocean name

    i. `grep –i`

```
“\<atlantic\>\|\<pacific\>\|\<indian\>\|\<arctic\>\|\<antarct
ic\>” a.txt
```

Grep

e. Display all lines containing an email address

i. What does an email address look like? It has the following structure.

1. username – let’s assume it can contain any character, except for @, *, !, and ?

2. @ - separator between the username and the hostname

3. hostname

a. Sequence of at least two elements separated by .

b. Let’s assume an element can contain any letter, digit, dash, or underscore

ii.
```
grep -i “\<[^@*\!?]\+@[a-z0-9_-]\+\(\.[a-z0-9_-]\+\)\+\>” a.txt
```

‘’ vs “”

The shell decides to do or not to do variable expansion before passing the arguments to grep based on quotes usage. Because the last step in shell processing of arguments is quote removal, grep never even sees the quotes. The shell also does command substitution and arithmetic expansion inside double quotes. For example:

```
$ echo "$(date) and 2+2=$((2+2))"
Tue Aug  5 18:52:39 PDT 2014 and 2+2=4
$ echo '$(date) and 2+2=$((2+2))'
$(date) and 2+2=$((2+2))
```

Sed

Sed stands for stream editor. It is mostly used as a special editor for modifying files automatically.
Sed has several commands, but most people only learn the substitute command: s. The substitute command changes all occurrences of the regular expression into a new value. A simple example is changing "day" in the "old" file to "night" in the "new" file:

```
sed s/day/night/ <old >new
```

The character after the s is the delimiter. It is conventionally a slash, because this is what ed, more, and vi use. It can be anything you want, however.

Sed

It is easier to read if you use an underline instead of a slash as a delimiter:

```
sed 's_/usr/local/bin_/common/bin_' <old >new
```

Some people use colons:

```
sed 's:/usr/local/bin:/common/bin:' <old >new
```

Others use the "|" character.

```
sed 's|/usr/local/bin|/common/bin|' <old >new
```

Sed

Sometimes you want to search for a pattern and add some characters, like parenthesis, around or near the pattern you found. It is easy to do this if you are looking for a particular string:

```
sed 's/abc/(abc)/' <old >new
```

This won't work if you don't know exactly what you will find. How can you put the string you found in the replacement string if you don't know what it is?

The solution requires the special character "&." It corresponds to the pattern found.

```
sed 's/[a-z]*/(&)/' <old >new
```

You can have any number of "&" in the replacement string. You could also double a pattern, e.g. the first number of a line:

```
% echo "123 abc" | sed 's/[0-9]*/& &/'
123 123 abc
```

Sed

The escaped parentheses (that is, parentheses with backslashes before them) remember a substring of the characters matched by the regular expression. You can use this to exclude part of the characters matched by the regular expression. The "\1" is the first remembered pattern, and the "\2" is the second remembered pattern. Sed has up to nine remembered patterns.
If you wanted to keep the first word of a line, and delete the rest of the line, mark the important part with the parenthesis:

```
sed 's/\([a-z]*\).*/\1/'
```

Sed

However, all it is doing is a grep and substitute. That is, the substitute command is treating each line by itself, without caring about nearby lines. What would be useful is the ability to restrict the operation to certain lines:

Specifying a line by its number.

Specifying a range of lines by number.

All lines containing a pattern.

All lines from the beginning of a file to a regular expression

All lines from a regular expression to the end of the file.

All lines between two regular expressions.

Sed can do all that and more. Every command in sed can be proceeded by an address, range or restriction like the above examples. The restriction or address immediately precedes the command: restriction command

Sed

The simplest restriction is a line number. If you wanted to delete the first number on line 3, just add a "3" before the command:

```
sed '3 s/[0-9][0-9]*//' <file >new
```

To search for a regular expression sed uses the same convention, provided you terminate the expression with a slash. To delete the first number on all lines that start with a "#," use:

```
sed '/^#/ s/[0-9][0-9]*//'
```

specify a range on line numbers by inserting a comma between the numbers. To restrict a substitution to the first 100 lines, you can use:

```
sed '1,100 s/A/a/'
```

Sed

Let’s manipulate the content of /etc/passwd

a. Display all lines, replacing all vowels with spaces

   i. `sed “s/[aeiou]/ /gi” /etc/passwd`

b. Display all lines, converting all vowels to upper case

   i. `sed “y/aeiou/AEIOU/” /etc/passwd`

c. Display all lines, deleting those containing numbers of five or more digits:

   i. `sed “/[0-9]\{5,\}/d” /etc/passwd`

d. Display all lines, swapping all pairs of letters

   i. `sed “s/\([a-z]\)\([a-z]\)/\2\1/gi” /etc/passwd`

e. Display all lines, duplicating all vowels

   i. `sed “s/\([aeiou]\)/\1\1/gi” /etc/passwd`

Sed

Delete with d

Using ranges can be confusing, so you should expect to do some experimentation when you are trying out a new script. A useful command deletes every line that matches the restriction: "d." If you want to look at the first 10 lines of a file, you can use:

```
sed '11,$ d' <file
```

which is similar in function to the head command. If you want to chop off the header of a mail message, which is everything up to the first blank line, use:

```
sed '1,/^$/ d' <file
```

Sed

Printing with p

The "p" command will duplicate the input. The command

```
sed 'p'
```

will duplicate every line. If you wanted to double every empty line, use:

```
sed '/^$/ p'
```

The q or quit command

There is one more simple command that can restrict the changes to a set of lines. It is the "q" command: quit. the third way to duplicate the head command is:

```
sed '11 q'
```

which quits when the eleventh line is reached. This command is most useful when you wish to abort the editing after some condition is reached.

Sed

Translation with y

```
y/string1/string2/
```

Replace all occurrences of characters in string1 with the corresponding characters in string2. If the number of characters in string1 and string2 are not equal, or if any of the characters in string1 appear more than once, the results are undefined. Any character other than <backslash> or <newline> can be used instead of <slash> to delimit the strings. If the delimiter is not 'n', within string1 and string2, the delimiter itself can be used as a literal character if it is preceded by a <backslash>. If a <backslash> character is immediately followed by a <backslash> character in string1 or string2, the two <backslash> characters shall be counted as a single literal <backslash> character. The meaning of a <backslash> followed by any character that is not 'n', a <backslash>, or the delimiter character is undefined.

Awk

The essential organization of an AWK program follows the form:

```
        pattern { action }
```

The pattern specifies when the action is performed. Like most UNIX utilities, AWK is line oriented. That is, the pattern specifies a test that is performed with each line read as input. If the condition is true, then the action is taken. The default pattern is something that matches every line. This is the blank or null pattern. Two other important patterns are specified by the keywords "BEGIN" and "END". As you might expect, these two words specify actions to be taken before any lines are read, and after the last line is read.

Awk

1. Manipulate the content of /etc/passwd, using AWK with the program provided on the command line

a. Display all the usernames, but only the usernames, and nothing else. We will use argument –F to tell AWK that the input file is separated by :, and then we will print the first field of each line, by not providing any selector for the block.

i. `awk –F: ‘{print $1}’ /etc/passwd`

b. Print the full name (the user info field) of the users on odd lines

i. `awk –F: ‘NR % 2 == 1 {print $5}’ /etc/passwd`

c. Print the home directory of users having their usernames start with a vowel

i. `awk -F: '/^[aeiouAEIOU]/ {print $6}' /etc/passwd`

d. Print the full name of users having even user ids

i. `awk -F: '$3 % 2 == 0 {print $5}' /etc/passwd`

Awk

e. Display the username of all users having their last field end with “nologin”

i. `awk -F: '$NF ~ /nologin$/ {print $1}' /etc/passwd`

f. Display the full names of all users having their username longer than 10 characters

i. `awk -F: 'length($1) > 10 {print $5}' /etc/passwd`

Awk

2. Keep using /etc/passwd as input file, but provide AWK programs in a file. The command will look like

   a. `awk -F: -f prog.awk /etc/passwd`

   b. Provide the content of file prog.awk so that the command above will print all user on even line having a group id less than 20

```
NR % 2 == 0 && $4 < 20 {print $5}
```

   c. Display the sum of all user ids

```
BEGIN {  sum=0}
{  sum += $3}
END {  print sum}
```

Awk

d. Display the product of the differences between the user id and the group id

```
    BEGIN {  prod=1}
    {  prod *= $3-$4}
    END {  print prod}
```