Unix/Linux Tutorial for Beginners
Session IV

Mihaela Martis

NBIS & Faculty of Medicine and Health Sciences
Division Cell Biology, IKE

NBIS NATIONAL BIOINFORMATICS INFRASTRUCTURE SWEDEN

elixir

LINKÖPING UNIVERSITY

dirname

- dirname – strips the filename itself, giving the 'directory' part of the pathname
- usage: dirname <name>

```
$ dirname /usr/bin/sort
/usr/bin

$ dirname mySong.mp3
.
```

basename

- basename – strips any 'path' name components from a filename
- usage: basename <filename> <suffix>

```
$ basename /home/duck/myProject/myScript.sh
myScript.sh
```

- can also strip a suffix from a filename, if it is identical with the end of the filename

```
$ basename /home/duck/myProject/myScript.sh .sh
myScript
```

sed – manipulate text in files

- sed – stream editor allows you to filter and transform text
- edits text line-by-line and displays output on the screen
- usage: sed <options> <'operation/regexp/replacement/flags'> <file>
- commands:
  - operation – specifies the action to be performed
  - regexp and replacement – specify search term and the substitution term
  - flags – are additional parameters that control the operation

sed – examples

- use sed without any commands → prints each line of the file

```
$ cat id_list.txt
ATMG00020
ATMG00030
ATMG00040
ATMG00050

$ sed '' id_list.txt
ATMG00020
ATMG00030
ATMG00040
ATMG00050
```

- use sed with commands, e.g. d

```
$ sed '2d' id_list.txt
ATMG00020
ATMG00040
ATMG00050
```

Search and replace with sed

- most used operation in sed – s/regexp/replacement/[flags]

```
$ cat my_frog.txt
Down by the river is a little frog, frog, frog.
```

- replace 'frog' with 'cat'

```
$ sed 's/frog/cat/' my_frog.txt
Down by the river is a little cat, frog, frog.
```

- replace multiple patterns → sed -e 'command1' -e 'command2' file

```
$ sed -e 's/frog/cat/' -e 's/Down/Up/' my_frog.txt
Up by the river is a little cat, frog, frog.
```

Other useful flags (I)

- g – replace all the instances of regexp with replacement (globally)

```
$ cat sed 's/frog/cat/g' my_frog.txt
Down by the river is a little cat, cat, cat.
```

- n (n=number) – replace n^{th} instance of the regexp with replacement

```
$ sed 's/frog/elephant/3' my_frog.txt
Down by the river is a little frog, frog, elephant.
```

- i – ignores case for matching regexp

```
$ sed 's/down/Up/i' my_frog.txt
Up by the river is a little frog, frog, elephant.
```

Other useful flags (II)

- w – write out the result to a file after substitution was made

```
$ sed 's/frog/elephant/wmyelephant.txt' my_frog.txt
Down by the river is a little elephant, frog, frog.
$ ls
myelephant.txt    my_frog.txt
```

- p – prints the line immediately if a substitution took place

```
$ sed 's/frog/elephant/p' my_frog.txt
Down by the river is a little elephant, frog, frog.
Down by the river is a little elephant, frog, frog.
```

- d – specified without replacement, deletes the found regexp

```
$ sed 'd' my_frog.txt
$
$ sed '1d' my_frog.txt
$
```

More sed examples (I)

- replace 'and' with 'or' except in the 2nd line

```
$ cat rime.txt
When witches go riding
and black cats are seen
the moon laughs and whispers
'tis near Halloween

$ sed '2!s/and/or/g' rime.txt
When witches go riding
and black cats are seen
the moon laughs or whispers
'tis near Halloween
```

More sed examples II

- show only the 3rd and 4th line

```
$ sed -n 3,4p rime.txt
the moon laughs and whispers
'tis near Halloween
```

- show all lines except for the lines from 2 to 3

```
$ sed 2,3d rime.txt
When witches go riding
'tis near Halloween
```

rev

- rev – reverse lines of a file or files

```
$ cat reverse_me.txt
MADAM I'M ADAM
HANNA
MAMA
$ rev reverse_me.txt
MADA M'I MADAM
ANNAH
AMAM
```

tr – translate

- usage: tr <OPTION> <SET1> <SET2>
- if both, SET1 and SET2, are specified without any options → tr replace each character in SET1 with each character in SET2

```
$ echo 'linux is not easy but a lot of fun' | tr "a-z" "A-Z"
LINUX IS NOT EASY BUT A LOT OF FUN
```

tr – translate

- usage: tr <OPTION> <SET1> <SET2>
- if both, SET1 and SET2, are specified without any options → tr replace each character in SET1 with each character in SET2

```
$ echo 'linux is not easy but a lot of fun' | tr "a-z" "A-Z"
LINUX IS NOT EASY BUT A LOT OF FUN
```

- translate all the white-space to tabs

```
$ echo 'For test purposes only' | tr [:space:] '\t'
For     test        purposes        only
```

Delete a specific pattern

- tr -d – delete specified characters

```
$ echo 'the geek stuff' | tr -d 'geek'
the stuff

$ echo 'my username is 4333434' | tr -d [:digit:]
my username is

$ echo 'We learn Linux today' | tr -d "[:space:]"
WelearnLinuxtoday
```

Sort the content of files

- sort – rearranges the lines in a text file so that they are sorted, numerically and alphabetically
- it **does** not change the file, because it sends the sorted result to the screen
- usage: sort <OPTION> <FILE>
- sorting rules: alphabetical order, number < lowercase < uppercase

Sort the content of files

- sort – rearranges the lines in a text file so that they are sorted, numerically and alphabetically
- it **does** not change the file, because it sends the sorted result to the screen
- usage: sort <OPTION> <FILE>
- sorting rules: alphabetical order, number < lowercase < uppercase

```
$ sort lengths.txt
  12 ethane.pdb
  15 propane.pdb
  20 cubane.pdb
  21 pentane.pdb
  30 octane.pdb
   9 methane.pdb
```

Numerical sorting

- -n – compare according to string numerical value

```
$ sort -n lengths.txt
  9 methane.pdb
 12 ethane.pdb
 15 propane.pdb
 20 cubane.pdb
 21 pentane.pdb
 30 octane.pdb
```

Column based sorting

- -k – sort a file on the basis of a column

```
$ sort -k 2 lengths.txt
  20 cubane.pdb
  12 ethane.pdb
   9 methane.pdb
  30 octane.pdb
  21 pentane.pdb
  15 propane.pdb
```

Reverse sorting and uniqueness

- -r – prints the sorted output in reverse order

```
$ sort -n -r lengths.txt
  30 octane.pdb
  21 pentane.pdb
  20 cubane.pdb
  15 propane.pdb
  15 ethane.pdb
   9 methane.pdb
```

- -u – sorts lines and removes duplicate lines from the sorted output

```
$ sort -n -u lengths.txt
  30 octane.pdb
  21 pentane.pdb
  20 cubane.pdb
  15 ethane.pdb
   9 methane.pdb
```

Removing repeated lines

- uniq – filters out adjacent, repeated lines in a file and writes the filtered data to an output file
- uniq <OPTION> <INPUT> <OUTPUT>

```
$ cat myfile.txt
This is a line.
This is a line.
This is a line.
This is also a line.
This is also a line.
This is also also a line.

$ uniq myfile.txt
This is a line.
This is also a line.
This is also also a line.
```

Options for *uniq*

- `-c` – counts how often a line has been seen

```
$ uniq -c myfile.txt
  3 This is a line.
  2 This is also a line.
  1 This is also also a line.
```

Options for *uniq*

- -c – counts how often a line has been seen

```
$ uniq -c myfile.txt
  3 This is a line.
  2 This is also a line.
  1 This is also also a line.
```

- -d – prints only duplicate lines

```
$ uniq -d myfile.txt
  This is a line.
  This is also a line.
```

Options for *uniq*

- -c – counts how often a line has been seen

```
$ uniq -c myfile.txt
  3 This is a line.
  2 This is also a line.
  1 This is also also a line.
```

- -d – prints only duplicate lines

```
$ uniq -d myfile.txt
  This is a line.
  This is also a line.
```

- -u – prints only uniq lines

```
$ uniq -u myfile.txt
  This is also a line.
```

Extract text from files

- cut – removes or 'cuts out' sections of each line of a file
- cut <OPTION> <FILE>

```
$ cat employees.txt
Simon Strange 62
Pete Brown 37
Mark Brown 46
$ cut -d ' ' -f 3 employees.txt
62
37
46
$cut -c 1 employees.txt
S
P
M
```

How to search for a pattern?

- grep – finds and prints lines in files that match a pattern
- is a contraction of '**g**lobal/**r**egular **e**xpression/**p**rint'
- usage: grep <OPTIONS> PATTERN <FILE>

How to search for a pattern?

- grep – finds and prints lines in files that match a pattern
- is a contraction of '**g**lobal/**r**egular **e**xpression/**p**rint'
- usage: grep <OPTIONS> PATTERN <FILE>
- options and files are optional

How to search for a pattern?

- grep – finds and prints lines in files that match a pattern
- is a contraction of '**g**lobal/**r**egular **e**xpression/**p**rint'
- usage: grep <OPTIONS> PATTERN <FILE>
- options and files are optional
- pattern - is mandatory
  - can be a certain word, or a regular expression
  - *regular expression* – a sequence of characters (letter, number, punctuation mark) that describes the searched pattern

A basic example

- find pattern in a file

```
$ cat input.txt
Welcome to Linux.
This is an introductory workshop and aimed for those
with little or no experience.

$ grep "Linux" input.txt
Welcome to Linux
```

- no input file → grep appears to 'hang' waiting for input

```
$ grep "food"
I bought some food.
I bought some food.
```

Ctrl d – 'end of file' signal, which tells grep that it has reached the end of the file and can exit.

Pattern matching is case sensitive

- -i – ignore case of letters in both the pattern and input files

```
$ grep "LInUx" input.txt

$ grep -i "LInUx" input.txt
Welcome to Linux.
```

Search in more that one file

- multiple input files → grep searches for the pattern or string in all files

```
$ grep "Linux" input.txt output.txt
input.txt: Welcome to Linux.
output.txt: I hope you enjoyed working on Linux.
```

- search in a complete directory using the '*' argument

```
$ grep "Linux" *
input.txt: Welcome to Linux.
output.txt: I hope you enjoyed working on Linux.
Binary file Linux_exercises.pdf matches
grep: new_dir: Is a directory
```

Search recursively

- -r – search for pattern recursively in the sub-directories

```
$ grep -r "Linux" *
input.txt: Welcome to Linux.
new_dir/new.txt: Linux vs Windows
output.txt: I hope you enjoyed working on Linux
Binary file Linux_exercises.pdf matches
```

Search words only

- -w – forces grep to select only those lines containing matches that form whole words

```
$ cat groceries.txt
milk      potato     lemon      buttermilk    chocolate
banana    milksoap   cucumber   milkshake     tomato
butter

$ grep "milk" groceries.txt
milk
buttermilk
milksoap
milkshake

$grep -w "milk" groceries.txt
milk
```

Count the matched lines

- -c – print only a count of matching lines for each input file

```
$ grep -c "milk" groceries.txt
4
```

- -n – numbers the lines that match

```
$ grep -n "milk" groceries.txt
1:milk
4:buttermilk
7:milksoap
9:milkshake
```

Grep invert match

- -v – return all the lines that don't match the search expression

```
$ grep -v "milk" groceries.txt
butter
lemon
chocolate
banana
cucumber
tomato
potato
```

Grep using regular expressions

- regular expressions are complex and powerful
- allows to do complex searches
- e.x. find all words that have an 'o' in the second position

```
$ grep "^.o" groceries.txt
tomato
potato
```

Regular expression operators

Operator	Effect
.	matches any single character
$	matches the end of a line
^	matches the begin of a line
?	matches one occurrence of the character preceding
*	the preceding item will be matched 0 or more times
+	the preceding item will be matched 1 or more times
[]	matches one or more characters between the brackets
{N}	the preceding item is matched exactly N times
|	matches 2 conditions together (this|that)

Examples related to nucleotid/protein sequences

Pattern	Match
^ATG	find a pattern starting with ATG
^A[T,G,C]G	find patterns matching either with ATG, AGG, or ACG
TAG$	find a pattern ending with TAG
TA[G,A]$	find patterns matching either TAG or TAA
^A[T,G,C]G*TGTGAACT*TA[G,A]$	find gene containing a specific motif

Practical bioinformatics examples

1. How many genes are in the file 'brachy_CDS.fa'?
2. Extract all identifier from a fasta file.
3. Linearize the sequences in the fasta file 'brachy_CDS.fa'.
4. Extract sequences from a fasta file for a given subset of gene identifiers.

How many genes are in the file 'brachy_CDS.fa'?

```
$ grep ">" brachy_CDS.fa | wc -l
31029

$ grep -c ">" brach_CDS.fa
31029
```

Extract all identifier from a fasta file.

```
$ grep -o -E ">\w+" brachy_CDS.fa
>Bradi0009s00230
>Bradi0009s00240
>Bradi0009s00250

$ grep -o -E ">\w+" brach_CDS.fa | tr -d '>'
Bradi5g27675
Bradi5g27680
Bradi5g27687

$ grep -o -E ">\w+" brach_CDS.fa | tr -d '>' > bd_Ids.txt
$ cat bd_Ids.txt
Bradi5g27700
Bradi5g27710
Bradi5g27720
```

Linearize the sequences in the fasta file 'brachy_CDS.fa'.

```
$ sed -e 's/\(^>.*$\)/#\1#/' brachy_CDS.fa|tr -d "\n"|tr "#"
    "\n"|sed -e '/^$/d'
```

Extract sequences from a fasta file for a given subset of gene identifiers.

```
$ head -20 bd_Ids.txt > mySubsetIDs.txt

$ sed -e 's/\(^>.*$\)/#\1#/' brachy_CDS.fa | tr -d "\n" | tr
    "#" "\n" | sed -e '/^$/d'|grep -A1 -f mySubsetIDs.txt |
    sed '/^--$/d' > subset.fa

$ grep -c ">" subset.fa
20
```

Summary

- text processing commands: sort, sed, tr, basename, dirname
- filtering file content: cut, grep, uniq