Intermediate Linux Course

Release 1.1

Holger Dinkel & Frank Thommen

May 28, 2013

CONTENTS

Part I

More Commandline Tools

CHAPTER 1

COMMAND-LINE TOOLS

1.1 GZIP

gzip is a compression/decompression tool. When used on a file (without any parameters) it will compress it and replace the file by a compressed version with the extension '.gz' attached:

```
ls textfile*
 textfile
gzip textfile
ls textfile*
 textfile.gz
```

To revert this / to uncompress, use the parameter `-d`:

```
ls textfile*
 textfile.gz
gzip -d textfile
ls textfile*
 textfile
```

Note: As a convenience, on most Linux systems, a shellscript named `gunzip` exists which simply calls `gzip -d`

1.2 TAR

`tar` (tape archive) is a tool to handle archives. Initially it was created to combine multiple files/directories to be written onto tape, it is now the standard tool to collect files for distribution or archiving.

`tar` stores the permissions of the files within an archive and also copies special files (such as symlinks etc.), which makes it an ideal tool for archiving... Usually tar is used in conjunction with a compression tool such as gzip to create a compressed archive:

The most common command-line switches are:

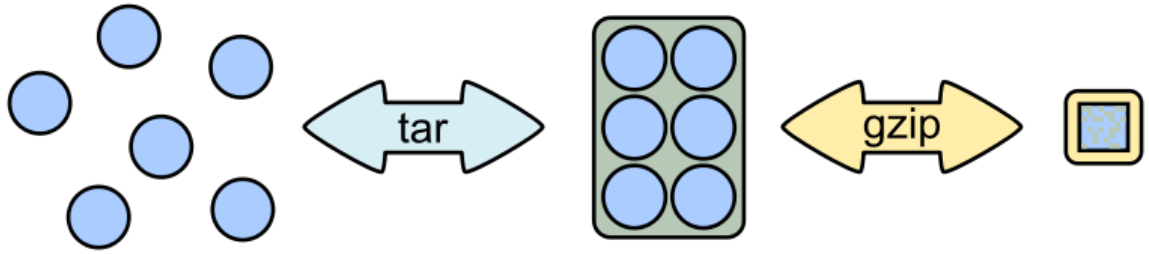

Figure 1.1: source: Th0msn80 (Wikipedia)

Option:	Effect:
-c	create an archive
-t	test an archive
-x	extract an archive
-z	use gzip compression
-f	filename filename of the archive

Note: Don't forget to specify the target filename. It needs to follow the `-f` parameter. Although you can combine options like such: `tar -czf archive.tar` the order matters, so `tar -cfz archive.tar` will *not* do what you want...

Creating an archive containing two files:

```
tar -cf archive.tar textfile1 textfile2
```

Listing the contents of an archive:

```
tar -tf archive.tar
 textfile1
 textfile2
```

Extracting an archive:

```
tar -xf archive.tar
```

Creating and extracting a compressed archive containing two files:

```
tar -czf archive.tar.gz textfile1 textfile2
tar -xzf archive.tar.gz
```

GREP Find lines matching a pattern in textfiles Usage: grep [options] pattern file(s)

```
grep -i ensembl P04637.txt

 DR Ensembl; ENST00000269305; ENSP00000269305; ENSG00000141510.
 DR Ensembl; ENST00000359597; ENSP00000352610; ENSG00000141510.
 DR Ensembl; ENST00000419024; ENSP00000402130; ENSG00000141510.
 DR Ensembl; ENST00000420246; ENSP00000391127; ENSG00000141510.
 DR Ensembl; ENST00000445888; ENSP00000391478; ENSG00000141510.
```

```
DR Ensembl; ENST00000455263; ENSP00000398846; ENSG00000141510.
```

Useful options:

Option:	Effect:
-v	Print lines that do not match
-i	Search case-insensitive
-l	List files with matching lines, not the lines itself
-L	List files without matches
-c	Print count of matching lines for each file

Count the number of fasta sequences (they start with a ">") in a file:

```
grep -c '>' twofiles.fasta
grep -c > twofiles.fasta
grep -c \> twofiles.fasta
grep -c \\> twofiles.fasta
 2
```

List all files containing the term "Ensembl":

```
grep -l Ensembl *.txt
 P04062.txt
 P12931.txt
```

1.3 SED

`sed` is a Stream EDitor, it modifies text (text can be a file or a pipe) on the fly.

Usage: '`sed command file`',

The most common usecases are:

Usecase	Command:
Substitute TEXT by REPLACEMENT:	's/TEXT/REPLACEMENT/'
Transliterate the characters x a, and y b:	'y/xy/ab/'
Print lines containing PATTERN:	'/PATTERN/p'
Delete lines containing PATTERN:	'/PATTERN/d'

```
echo "This is text." | sed 's/text/replaced stuff/'

 This is replaced stuff.
```

By default, text substitution are performed only once per line. You need to add a trailing 'g' option, to make the substitution 'global' ('s/TEXT/REPLACEMENT/g'), meaning all occurrences in a line are substituted (not just the first in each line). Note the difference:

```
echo "ACCAAGCATTGGAGGAATATCGTAGGTAAA" | sed 's/A/_/'

 _CCAAGCATTGGAGGAATATCGTAGGTAAA
```

```
echo "ACCAAGCATTGGAGGAATATCGTAGGTAAA" | sed 's/A/_/g'
 _CC__GC_TTGG_GG__T_TCGT_GGT___
```

When used on a file, sed prints the file to standard output, replacing text as it goes along:

```
echo "This is text" > textfile
echo "This is even more text" >> textfile
sed 's/text/stuff/' textfile
 This is stuff
 This is even more stuff
```

sed can also be used to print certain lines (not replacing text) that match a pattern. For this you leave out the leading 's' and just provide a pattern: '/PATTERN/p'. The trailing letter determines, what sed should do with the text that matches the pattern ('p': print, 'd': delete)

```
sed '/more/p' textfile

 This is text
 This is even more text
 This is even more text
```

As sed by default prints each line, you see the line that matched the pattern, printed twice. Use option '-n' to suppress default printing of lines.

```
sed -n '/more/p' textfile

 This is even more text
```

Delete lines matching the pattern:

```
sed '/more/d' textfile

 This is text
```

Multiple sed statements can be applied to the same input stream by prepending each by option '-e' (edit):

```
sed -e 's/text/good stuff/' -e 's/This/That/' textfile

 That is good stuff
 That is even more good stuff
```

Normally, sed prints the text from a file to standard output. But you can also edit files in place. Be careful - this will change the file! The '-i' (in-place editing) won't print the output. As a safety measure, this option will ask for an extension that will be used to rename the original file to. For instance, the following option '-i.bak' will edit the file and rename the original file to textfile.bak:

```
sed -i.bak 's/text/stuff/' textfile
cat textfile
 This is stuff
 This is even more stuff
cat textfile.bak
```

```
 This is text
 This is even more text
```

1.4 AWK

awk is more than just a command, it is a complete text processing language (the name is an abbreviation of the author's names). Each line of the input (file or pipe) is treated as a record and is broken into fields. Generally, awk commands are of the form: 'condition { action }', where:

condition is typically an expression # action is a series of commands

If no condition is given, the action is applied to each line, otherwise just to the lines that match the condition.

```
awk '{print}' textfile

 This is text
 This is even more text

awk '/more/ {print}' textfile
```

This is even more text

awk reads each line of input and automatically splits the line into columns. These columns can be addressed via $1, $2 and so on ($0 represents the whole line). So an easy way to print or rearrange columns of text is:

```
echo "Bob likes Sue" | awk '{print $3, $2, $1}'

 Sue likes Bob

echo "Master Obi-Wan has lost a planet" | awk '{print $4,$5,$6,$1,$2,$3}'
 lost a planet Master Obi-Wan has
```

awk splits text by default on whitespace (spaces or tabs), which might not be ideal in all situations. To change the field separator (FS), use option '-F' (remember to quote the field separator):

```
echo "field1,field2,field2" | awk -F',' '{print $2, $1}'
 field2 field1
```

Note two things here: First, the field separator is not printed, and second, if you want to have space between the output fields, you actually need to separate them by a comma or they will be catenated together...

```
echo "field1,field2,field2" | awk -F',' '{print $1 $2 $3}'
 field1field2field3
```

You can also combine the pattern matching and the column selection techniques:

```
awk '/more/ {print $3}' textfile

 even
```

awk really is powerful in filtering out columns, you can for instance print only certain columns of certain lines. Here we print the third column of those lines where the fourth column is 'more':

```
awk '$4=="more" {print $3}' textfile

 even
```

Note the double equal signs "==" to check for equality and note the quotes around "more". If you want to match a field, but not exactly, you can use '~' instead of '==':

```
awk '$4~"ore" {print $3}' textfile

 even
```

CHAPTER 2

HINTS

2.1 Quoting

In Programming it is often necessary to "glue together" certain words. Usually, a program or the shell splits sentences by whitespace (space or tabulators) and treats each word individually. In order to tell the computer that certain words belong together, you need to "quote" them, using either single (') or double (") quotes. The difference between these two is generally that within double quotes, variables will be expanded, while everything within single quotes is treated as string literal. When setting a variable, it doesn't matter which quotes you use:

```
MYVAR=This is set
 -bash: is: command not found

MYVAR='This is set'
echo $MYVAR
 This is set
MYVAR="This is set"
echo $MYVAR
 This is set
```

However, it does matter, when using (expanding) the variable: Double quotes:

```
export MYVAR=123

echo "the variable is $MYVAR"
 the variable is 123
echo "the variable is set" | sed "s/set/$MYVAR/"
 the variable is 123
```

Single quotes:

```
export MYVAR=123
echo 'the variable is $MYVAR'
 the variable is $MYVAR
echo "the variable is set" | sed 's/set/$MYVAR/'
 the variable is $MYVAR
```

Weird things can happen when parsing data/text that contains quote characters:

```
MYVAR='Don't worry'; echo $MYVAR
 >
```

```
# you need to press Ctrl-C to abort
MYVAR="Don't worry"; echo $MYVAR
 Don't worry
```

2.2 Expanding and Escaping

You already learned how to expand a variable such that its value is used instead of its name:

```
export MYVAR=123

echo "the variable is $MYVAR"
 the variable is 123
```

"Escaping" a variable is the opposite, ensuring that the literal variable name is used instead of its value:

```
export MYVAR=123

echo "the \$MYVAR variable is $MYVAR"
 the $MYVAR variable is 123
```

Note: The "escape character" is usually the backslash "\".

Part II

Solutions to the Exercises

CHAPTER 3

COMMANDLINE TOOLS

3.1 TAR & GZIP

1. Use gzip to compress the file P12931.txt

```
gzip P12931.txt
```

2. Decompress the resulting file P12931.txt.gz (revert previous command)

```
gunzip P12931.txt.gz
```

or

```
gzip -d P12931.txt.gz
```

3. Use tar to create an archive containing all fasta files in the current directory into an archive called "fastafiles.tar"

```
tar -c -f fastafiles.tar *.fasta
```

4. Use gzip to compress the archive "fastafiles.tar".

```
gzip fastafiles.tar
```

5. How can you achieve the two previous steps "using tar to create archive" and "gzip the archive" in one command?

```
tar -c -z -f fastafiles.tar.gz *.fasta
```

Note: Note the `-z`

---

6. Test (list the contents of) the compressed archive "fastafiles.tar.gz"

```
tar -tf fastafiles.tar.gz
```

7. Download the compressed PDB file for entry 1Y57 from rcsb.org (eg. "wget http://www.rcsb.org/pdb/files/1Y57.pdb.gz") and decompress it.

```
wget "http://www.rcsb.org/pdb/files/1Y57.pdb.gz"
gunzip 1Y57.pdb.gz
```

3.2 GREP

1. Which of the DNA files ENST* contains 'TATATCTAA' as part of the sequence?

```
grep TATATCTAA ENST0*
ENST00000380152.fasta:ACGGAAGAATGTGAGAAAAATAAGCAGGACACAATTACAACTAAAAAATATATCTAA
ENST00000544455.fasta:ACGGAAGAATGTGAGAAAAATAAGCAGGACACAATTACAACTAAAAAATATATCTAA
```

2. List only the names of the DNA files ENST* that contain 'CAACAAA' as part of the sequence.

```
grep -l CAACAAA ENST0*
ENST00000380152.fasta
ENST00000544455.fasta
```

3. Considering the previous example, would you consider grep a suitable tool to perform motif searches? Why not? Try to find the pattern 'CAACAAA' by manual inspection of the first two lines of each sequence.
4. Count the number of ATOMs (lines starting with 'ATOM') in the file 1Y57.pdb.
5. Does this number agree with the annotated number of atoms (Search the REMARKs for 'protein atoms')

```
grep -c "ATOM" 1Y57.pdb
3632
grep -i "protein atoms" 1Y57.pdb
REMARK   3   PROTEIN ATOMS            : 3600
```

This means there are 3600 atoms annotated in this PDB file, however we counted 3632. This is because grep also counted any occurrence of "ATOM" within REMARKS. We can avoid this by either filtering out the remarks:

```
grep -v "REMARK" 1Y57.pdb | grep -c "ATOM"
3600
```

...or by telling grep to only count those lines that start with "ATOM":

```
grep -c "^ATOM" 1Y57.pdb
3600
```

3.3 SED

1. Use sed to print only those lines that contain "version" in the files P05480.txt and P04062.txt

```
sed -n '/version/p' P05480.txt P04062.txt
```

2. Use sed to change the text 'sequence version 3' to 'sequence version 4' in the files P05480.txt and P04062.txt (without actually changing the files, just printing)

```
sed 's/sequence version 3/sequence version 4/' P05480.txt P04062.txt
```

3. Use sed to update the text 'sequence version 3' to 'sequence version 4' in the files P05480.txt and P04062.txt (this time, make the changes directly in the files)

```
sed -i.bak 's/sequence version 3/sequence version 4/' P05480.txt P04062.txt
```

4. Replace (transliterate) all occurrences of 'r' by 'l' and 'l' by 'r' (at the same time) in the file PROTEINS.txt (so that 'structural' becomes 'stluctular')

```
sed 'y/rRlL/lLrR/' PROTEINS.txt
```

3.4 AWK

1. Use awk to print only those lines that contain "version" in the files P12931.txt and P05480.txt and think about how this procedure is different to sed.

```
awk '/version/ {print}' P12931.txt P05480.txt
```

This is very similar to sed, you also have to use the slashes '/' to define the search pattern. However the sed notation is a little more concise...

2. For all FASTA files that begin with "P" ("P*.fasta") print only the second item of the header (split on "|") eg. for ">sp|P12931|SRC_HUMAN Proto-oncogene", print only "P12931"

```
awk -F"|" '/>/ {print $2}' P*.fasta
```

3. The file 'P12931.csv contains phosphorylation sites in the protein P12931. (If the file 'P12931.csv' does not exist, use 'wget' to download it from "http://phospho.elm.eu.org/byAccession/P12931.csv").

    1. Column three of this file lists the amino acid position of the phosphorylation site. You are only interested in position 17 of the protein. Try to use 'grep' to filter out all these lines containing '17'.

    ```
    grep 17 P12931.csv
    ```

    2. Now use awk to show all lines containing '17'.

    ```
    awk '/17/ {print}' P12931.csv
    ```

    3. Next try show only those lines where column three equals 17 (Hint: The file is semicolon-separated...).

```
awk -F";" '$3==17 {print}' P12931.csv
```

4. Finally print the PMIDs (column 6) of all lines that contain '17' in column 3.

```
awk -F";" '$3==17 {print $6}' P12931.csv
```

3.5 Quoting and Escaping

Familiarize yourself with quoting and escaping.

1. Run the following commands to see the difference between single and double quotes when expanding variables:

```
echo "$HOSTNAME"
...
echo '$HOSTNAME'
```

2. Next, use ssh to login to a different machine to run the same command there, again using both quoting methods:

```
ssh pc-atcteach01 'echo $HOSTNAME'
...
ssh pc-atcteach01 "echo $HOSTNAME"
```

Closely inspect the results; is that what you were expecting? Discuss this with your neighbour.

Part III

Comments

useful Links:

- ReST Cheat Sheet
- Writing Technical Documentation with Sphinx, Paver, and Cog
- using the sffms latex class in sphinx

Part IV

Indices and tables

- *genindex*