Linux II and III

Douglas Scofield
Evolutionary Biology Centre, Uppsala University
douglas.scofield@ebc.uu.se

Creating directories and files

- `mkdir`
Creating directories and files

- **touch**
  - if file does not exist, creates it with 0 size
  - if file exists, update its modification time

```
$ touch a
$ touch b c d
```

```
 milou-b: ~/course $ touch a
 milou-b: ~/course $ ls -l
 total 0
-rw-rw-r-- 1 douglas douglas 0 Aug 24 11:00 a
 milou-b: ~/course $ date
 Sun Aug 24 11:01:01 CEST 2014
 milou-b: ~/course $ touch a
 milou-b: ~/course $ ls -l
 total 0
-rw-rw-r-- 1 douglas_douglas 0 Aug 24 11:01 a
```

```
 milou-b: ~/course $ touch b c d
 milou-b: ~/course $ ls -l
 total 0
-rw-rw-r-- 1 douglas_douglas 0 Aug 24 11:01 a
-rw-rw-r-- 1 douglas_douglas 0 Aug 24 11:03 b
-rw-rw-r-- 1 douglas_douglas 0 Aug 24 11:03 c
-rw-rw-r-- 1 douglas_douglas 0 Aug 24 11:03 d
```

Creating directories and files

- **cat**
  - `concatenate`: dumps the contents of a file
  - can also be used to quickly create a short file

```
$ cat > e
this is a short file
$ cat > f
this is a short file
```

```
 milou-b: ~/course $ cat > e
 this is a short file
 milou-b: ~/course $ cat e
 this is a short file
 milou-b: ~/course $ ls -l
 total 32
-rw-rw-r-- 1 douglas_douglas 0 Aug 24 11:01 a
-rw-rw-r-- 1 douglas_douglas 0 Aug 24 11:03 b
-rw-rw-r-- 1 douglas_douglas 0 Aug 24 11:03 c
-rw-rw-r-- 1 douglas_douglas 0 Aug 24 11:03 d
-rw-rw-r-- 1 douglas_douglas 21 Aug 24 13:14 e
```

Ctrl-D = EOF = “End Of File”

Type this, then Return, then
Redirecting input:  

- `command < file`
  - give `command` input from ‘file’
  - for `command`, input comes from ‘standard input’, ‘stdin’

```bash
milou-b: ~/course $ cat < e
this is a short file
milou-b: ~/course $ 
```

- for flexible commands, it is not quite the same as giving a file on the command line

```bash
milou-b: ~/course $ cat e
this is a short file
milou-b: ~/course $ cat e e
this is a short file
milou-b: ~/course $ cat e e e
this is a short file
milou-b: ~/course $ cat e e e e
this is a short file
milou-b: ~/course $ 
```

Redirecting input:  

- `command < file`
  - Files and stdin generally cannot be mixed using ‘<’

```bash
milou-b: ~/course $ cat e e < e
this is a short file
this is a short file
milou-b: ~/course $ cat > f
this file is a little longer
milou-b: ~/course $ cat f
this file is a little longer
milou-b: ~/course $ cat e f
this is a short file
this file is a little longer
```

- command line files can be more flexible

```bash
milou-b: ~/course $ cat f - < e
this file is a little longer
this is a short file
milou-b: ~/course $ 
```

- Some commands understand the filename ‘-’ to mean ‘read from stdin’

```bash
milou-b: ~/course $ cat f &< e
Here, cat first reads f, then reads stdin, which has e
```

Redirecting output:  > and  >>

- **command** > file
  - ‘standard output, ‘stdout’, for command goes to ‘file’

```
milou-b: ~/course $ cat e > ee
milou-b: ~/course $ cat ee
this is a short file
milou-b: ~/course $ ls -l e ee
-rw-rw-r-- 1 douglas douglas 21 Aug 24 13:14 e
-rw-rw-r-- 1 douglas douglas 21 Aug 24 17:08 ee
milou-b: ~/course $
```

- **command** >> file
  - **appends** stdout from **command** to ‘file’

```
milou-b: ~/course $ cat f >> ee
milou-b: ~/course $ cat ee
this is a short file
this file is a little longer
milou-b: ~/course $ ls -l e f ee
-rw-rw-r-- 1 douglas douglas 21 Aug 24 13:14 e
-rw-rw-r-- 1 douglas douglas 21 Aug 24 17:04 ee
-rw-rw-r-- 1 douglas douglas 29 Aug 24 13:47 f
milou-b: ~/course $
```

Connecting stdout to stdin with the pipe:  |  

- **command1**  |  **command2**
  - stdout of **command1** is connected to stdin of **command2**

```
milou-b: ~/course $ cat f ee | cat > ff
milou-b: ~/course $ cat ff
this is a short file
this is a little longer
milou-b: ~/course $ cat f | wc -l
1
milou-b: ~/course $ cat ee | wc -l
2
milou-b: ~/course $ cat ff | wc -l
3
```

- ‘wc -l’ counts the number of lines in stdin, or a file or set of files, and prints the result to stdout
- ‘wc’ counts the number of lines, words and characters

```
milou-b: ~/course $ cat ff | wc
  3  17    79
```
The “other” output: standard error, ‘stderr’

- Commands which use pipes or redirect stdout may still produce output to the terminal:

```
milou-b: ~/course $ bwa mem -a -E3 -t 8 ref.fa reads.fq | samtools view -Sb - > aln.bam
[M:m:main_mem] read 32 sequences (4916 bp)...
[M:mm:mem_process_segs] Processed 32 reads in 0.014 CPU sec, 0.005 real sec
[main] Version: 0.7.10-r789
[main] CMD: bwa mem -a -E3 -t 8 ref.fa[samopen] SAM header is present: 1 sequences.
reads.fq
[main] Real time: 0.056 sec; CPU: 0.019 sec
```

- This is a second output stream: standard error, ‘stderr’
- Not all tools use it
- To capture standard error, use `2>` (for stdout, `1>` equals `>)

```
milou-b: ~/course $ bwa mem -a -E3 -t 8 ref.fa reads.fq 2> aln.bam
[1]: [samopen] SAM header is present: 1 sequences.
milou-b: ~/course $ bwa mem -a -E3 -t 8 ref.fa reads.fq 2> bwa.stderr | samtools view
-Sb - > aln.bam
milou-b: ~/course $  
```

Putting stderr together with stdout

- Often you want to capture all output from a command, not just stdout or stderr
- You can reassign stderr to be directed to stdout (or vice versa) and then capture the combined output stream

```
command > file 2>&1
```

- when directing output to a file, this must come after `> file'
- When piping between tools, this is usually not a good idea because downstream tools usually expect one output stream or the other, but not both

```
milou-b: ~/course $ bwa mem -a -E3 -t 8 ref.fa reads.fq 2>&1 | samtools view -Sb - > aln.bam
[sam_read] missing header? Abort!
milou-b: ~/course $  
```

This sends both stdout and stderr through the pipe. For this, you can also use `|&' instead of `2>&1 |'.
Shell wildcards: ? *

- Using wildcards, filenames can be specified using expressions
- 0, 1 or more than 1 filename may match the expression
- Bash wildcards are similar but not identical to grep, sed, etc.

- ‘?’ matches any 1 character

- ‘*’ matches 0 or more of any character

Shell wildcards: character groups with [ ] - ^

- You can specify character groups using [ ] - ^

Match specific character: [a]  Two characters: [af]  Range: [a-f]

Anything but specific characters: [^a]  [^a-d]
Shell wildcards: groups of terms using { , }

- You can specify term groups using { , }
  
  ```
  milou-b: ~/course $ ls 
  a b c d e ee f ff
  ```

  There must be at least two terms: {e,f}
  
  ```
  milou-b: ~/course $ ls {e,f}
  e f
  ```

- Terms can contain wildcards: {c,??}
  
  ```
  milou-b: ~/course $ ls {c,??}
  c ee ff
  ```

Using terms can save a lot of typing:

  ```
  milou-b: ~ $ cp /some/really/long/directory/run-1.{log,out,err,pdf,sh} .
  ```

Man(ual) pages

- Uncertain about a command, or want to know what more it can do?
- Type ‘man command’
  
  man wc
Finding patterns: grep

- ‘grep’ is a very useful command that finds patterns
- Patterns can be literal (‘My name is Uppmax’) or can be regular expressions (‘My [a-z]+ is Uppmax’)
- Wildcards are regular expressions too, but in grep you can do a lot more
  - many tutorials available online
  - same syntax is also used in ‘sed’ and ‘awk’ and (a slight variant) ‘perl’
  - we will only use a tiny bit of what is available
How many sequences in a sequence file?

- Get example Fasta and FastQ files
  
  rackham3: ~/course $ cp /proj/g2017030/labs/linux2_additional-files/*.{fa,fq}.
  rackham3: ~/course $ ls
  a b c d e ee f ff reads.fq ref.fa

- Looking at their formats...
  
  rackham3: ~/course $ head -n 4 ref.fa reads.fq
  === ref.fa ===
  >seq1
  TGGCTCCTTTTTGATGTCATGACTGGCGGTTCAATATCAATTGCGCGCTCC
  TGGCTCCTTTTTGATGTCATGACTGGCGGTTCAATATCAATTGCGCGCTCC
  ATTTTCTGAAACGAGTCTAAGCTGGCTGACGCTCGCGTCGTAGACTGAAATAACTAAGAC
  
  === reads.fq ===
  @UQNPK:00025:00052
  GAAGAAACGACGGAA
  +
  BB7BB@CCCCCBBCC

- Lines holding Fasta sequence names begin with ‘>’
- ... FastQ ... begin with ‘@’

How many sequences: name lines begin with > or @

- ‘at beginning of line’ is indicated with ‘^’
- ‘at end of line’ is indicated with ‘$’
- The regular expression should be single- or double-quoted so bash does not become confused (we will be using ‘^’ !)

  milou2: ~/course $ grep '^>\.' ref.fa
  >seq1
  >seq2
  >seq3
  >seq4
  >seq5
  >seq6
  >seq7

  milou2: ~/course $ grep '^>\.' ref.fa | wc -l
  7

- How would you do this to count FastQ reads?
- Can you think of another way to count FastQ with ‘wc -l’?
Extracting pieces of output or files

• What if we want just the sequence names?

```bash
milou2: ~/course $ grep '->' ref.fa
>seq1
>seq2
>seq3
>seq4
>seq5
>seq6
>seq7
```

• ‘cut’ the sequence names out by (c)olumn!

```bash
milou2: ~/course $ grep '->' ref.fa | cut -c2-
seq1
seq2
seq3
e1
e2
e3
e4
e5
e6
e7
```

Find the line of a specific sequence

• ‘grep -n’ includes line (n)umbers

```bash
milou2: ~/course $ grep -n '->' ref.fa
1:$seq1
```

• for all matches

```bash
milou2: ~/course $ grep -n '->' ref.fa
1:$seq1
11:$seq1
19:$seq1
26:$seq4
34:$seq5
43:$seq6
52:$seq7
```

• Only line numbers? ‘cut’ a (f)ield using a (d)elimiter

```bash
milou2: ~/course $ grep -n '->' ref.fa | cut -f1 -d':'
1
19
26
34
43
52
```

```bash
milou2: ~/course $ grep -n '->' ref.fa | cut -f1 -d':'
| tail -n 1
1
```

What if we only want the line number for the last sequence?
Some differences with grep patterns

- `'.'` means any character, equivalent to `'?'` in the shell
- `'*` means ‘0 or more of the previous character’
- `'.*'` is equivalent to `'*'` in the shell

Some grep patterns can be specified more simply by providing the `–P` option (‘grep –P’ for (P)erl style patterns)
- `'+` means ‘1 or more of the previous character’ (‘\+’ w/o `–P`)
- Terms are easier, too

```
rackham3: ~/course $ grep '^[\(seq1\|seq7\)]$' ref.fa
>seq1
>seq7
```

```
rackham3: ~/course $ grep –P '^[\(seq1\|seq7\)]$' ref.fa
>seq1
>seq7
```

Other grep options

- `grep -i` : (i)gnore case in expression

```
milou2: ~/course $ grep –i 'SEQ1' ref.fa
>seq1
```

- `grep -v` : in(v)ert match, lines that do not match expression

```
milou2: ~/course $ grep –i 'SEQ[1-5]' ref.fa | grep –v '^[do]$$'
>seq1
>seq2
>seq3
```

- `grep -F` : (F)ixed expression, ignore wildcards

```
milou2: ~/course $ ls
* a b c d e e f f f f reads.fq ref.fa
milou2: ~/course $ ls –l | grep –F 'w'
-rw-rw-r-- 1 douglas douglas 0 Aug 25 15:11 *
milou2: ~/course $ ls –l "w"
-rw-rw-r-- 1 douglas douglas 0 Aug 25 15:11 *
```

- `grep --color` : use color in output

```
milou2: ~/course $ grep –i --color 'SEQ[^2-6]' ref.fa
>seq1
>seq7
```
Just a few more grep options

- **grep -c**: only print a (c)ount of the matching lines
  ```bash
  milou2: ~/course $ grep -c 'grep' ref.fa
  7
  milou2: ~/course $ grep -cv 'grep' ref.fa
  52
  ```

- **grep -m N**: stop output after N (m)atches
  ```bash
  milou2: ~/course $ grep -m 1 'q[367]' ref.fa
  >seq3
  ```

- **grep -H**: include the filename (default with >1 file)
  ```bash
  milou2: ~/course $ grep -Hn --color 'q[14]' ref.fa
  ref.fa:seq1
  ref.fa:seq26
  milou2: ~/course $ cat ref.fa | grep -Hn --color 'q[14]'
  (standard input):seq1
  (standard input):seq26
  ```

- **grep -l, -L**: only print filenames containing/(L)acking match
  ```bash
  milou2: ~/course $ grep -l 'seq1$' ref.fa reads.fq
  ref.fa
  milou2: ~/course $ grep -L 'seq1$' ref.fa reads.fq
  reads.fq
  ```

The last grep options, seriously

- **grep -B N**: include N lines (B)efore the match in output
  ```bash
  milou2: ~/course $ grep -B 1 '->seq2$' ref.fa
  TGTCAGGACCC
  >seq2
  ```

- **grep -A N**: include N lines (A)fter the match in output
  ```bash
  milou2: ~/course $ grep -A 3 '^@ONPK:00685:00805$' reads.fq
  @ONPK:00685:00805
  GAAGGACTGGACTATCCG
  +
  1:0-1:79895442444:4444999
  milou2: ~/course $ grep -A 3 '^@ONPK:00685:00805$' reads.fq | tail -n 1
  ```

- Just the sequence of that read? The quality string? The name of the next read?
  ```bash
  milou2: ~/course $ grep -A 1 '^@ONPK:00685:00805$' reads.fq | tail -n 1
  ```

  ```bash
  milou2: ~/course $ grep '^>' ref.fa | grep -A 1 '^>seq3$' | tail -n 1
  >seq4
  ```
Bash $( ... )

- $(< file ) replaces the whole $( ... ) with the contents of file
- $(command) replaces $( ... ) with the output of command

```bash
milou-b: ~/course $ cat > filelist

milou-b: ~/course $ grep -n \'longer\' $(< filelist)
ff:1: this file is a little longer
ff:3: this file is a little longer

milou-b: ~/course $ grep -n \'longer\' $(cat filelist)
ff:1: this file is a little longer
ff:3: this file is a little longer

milou-b: ~/course $ grep -n \'longer\' $(grep \'ef\' filelist)
ff:1: this file is a little longer
ff:3: this file is a little longer
```

```bash
milou-b: ~/course $ for F in $(cat filelist) ; do
  > grep -hn \'longer\' "$F"
  > done
ff:1: this file is a little longer
ff:3: this file is a little longer
```

Bash <( ... ) : anonymous named pipe

- <(command) creates a temporary file containing the output of command
- Useful for creating temporary files that don’t use extra space
  - for example, do some preliminary processing before use

```bash
diff <(sort file1.txt) <(sort file2.txt)

```
- decompressing files for commands that don’t handle compressed files

```bash
bwa mem ref.fa <(xzcat r1.fq.xz) <(xzcat r2.fq.xz) | ...
```
Augmenting your environment: `.bashrc`

- Wherever you are, save your position with `pushd .` and cd to your home directory. See the directory stack with `dirs`.
  ```bash
  milou2: ~/course $ pushd .
  ~/course ~/course
  milou2: ~/course $ cd
  milou2: ~ $ dirs
  ~ ~/course
  milou2: ~ $ dirs -v
  Ø ~
  1 ~/course
  ```

- Edit the `.bashrc` configuration file with nano, add the line
  ```bash
  alias rm='rm -i'
  ```

- Move back to previous location with `popd`.
  ```bash
  milou2: ~ $ popd
  ~/course
  milou2: ~/course $ dirs
  ~/course
  ```

Load an UPPMAX module with some tools

- the tinyutils module provides several useful tools
- search for module versions with `module spider`
  ```bash
  rackham3: ~/course $ module spider tinyutils
  ```

- load the module with `module load`
  ```bash
  rackham3: ~/course $ module load tinyutils/1.4
  rackham3: ~/course $ which hist
  /sw/apps/tinyutils/1.4/rackham/hist
  rackham3: ~/course $ which table
  /sw/apps/tinyutils/1.4/rackham/table
  ```
How long are my fasta sequences?

- Use the 'fastalength' tool from the exonerate module
  - module load bioinfo-tools
  - module load exonerate

```bash
milou-b: ~/course $ module load bioinfo-tools exonerate
milou-b: ~/course $ fastalength ref.fa
  493 seq1
  368 seq2
  356 seq3
  364 seq4
  461 seq5
  468 seq6
  383 seq7
```

What is the total length? Mean? Median?

- That’s what tinyutils are for

```bash
milou-b: ~/course $ fastalength ref.fa | cut -f1 -d' '
  493
  368
  356
  364
  461
  468
  383
milou-b: ~/course $ fastalength ref.fa | cut -f1 -d' ' | sum
  2893
milou-b: ~/course $ fastalength ref.fa | cut -f1 -d' ' | mean
  413.286
milou-b: ~/course $ fastalength ref.fa | cut -f1 -d' ' | median
  368
milou-b: ~/course $ fastalength ref.fa | cut -f1 -d' ' | max
  493
milou-b: ~/course $ fastalength ref.fa | cut -f1 -d' ' | min
  356
```
What is the length distribution of my reads?

- With a bit of awk (or the len tinyutil) to get lengths of lines

```bash
milou-b: ~/course $ grep "^[ACGTIN]*$" reads.fq | head -n 3
GAAGAACGCGACGGAA
GAAGAACGCGACGGAA
GAAGAACGCGACGGAA
```

```bash
milou-b: ~/course $ grep "^[ACGTIN]*$" reads.fq | awk '{ print length($0) }' | head -n 3
15
15
45
```

```bash
milou-b: ~/course $ grep "^[ACGTIN]*$" reads.fq | len | head -n 3
15
15
45
```

```bash
milou-b: ~/course $ grep "^[ACGTIN]*$" reads.fq | len | table
15
19
46
22
15
25
25
```

```bash
milou-b: ~/course $ grep "^[ACGTIN]*$" reads.fq | len | table | sort
15
19
22
46
25
15
25
```

Using ‘find’ to search a directory tree

- `find location list-of-file-attributes optional-actions`

```bash
rackhan3: ~/course $ mkdir directory
rackhan3: ~/course $ mv ?? directory/
rackhan3: ~/course $ find . -name d ./.directory/d
rackhan3: ~/course $ find . -name '?' ./.directory/ee ./.directory/ff
rackhan3: ~/course $ find . -iname B -ls
473861739 0 -rw-rw-r-- 1 douglas douglas 0 Jan 13 14:29 ./directory/b
rackhan3: ~/course $ find . -type d ./.directory
rackhan3: ~/course $ find . -type f -name '*.fa' -exec grep "^>seq3" {} \;
>seq3
rackhan3: ~/course $ find . -type f -name '*.fa' -exec grep -Hn "^>seq3" {} \\
./ref.fa:19:>seq3
```

- Most wildcards work, use within quotes
- The `-iname` option is case-insensitive
- `-ls` runs `ls -l` on each file
- Look for specific type of file
- Run commands on found files

- Other options for size, ownership, modification times, etc.
- See the (long) man page and online tutorials for more
Create symbolic links to clear things up

• Use `ln -s` ... do not forget the `-s`!
• Symbolic links indicate the location of another file/directory

```
milou-b: ~/course $ ln -s sf
milou-b: ~/course $ ls -li sf
1105098318 -rw-rw-r-- 1 douglas douglas 22 Jan 27 2015 f
1915648234 rwxrwxrwx 1 douglas douglas 1 Aug 22 11:37 sf -> f
```

‘Hard links’ (In without -s) are rarely necessary

• Hard links are truly another name for the same file

```
milou-b: ~/course $ ln hf
milou-b: ~/course $ ls -li hf hf sf
1105098318 -rw-rw-r-- 2 douglas douglas 22 Jan 27 2015 f
1105098318 -rw-rw-r-- 2 douglas douglas 22 Jan 27 2015 hf
1915648234 rwxrwxrwx 1 douglas douglas 1 Aug 22 11:37 sf -> f
milou-b: ~/course $ rm hf
rm: remove regular file 'f'? y
milou-b: ~/course $ ls -li hf hf sf
ls: cannot access f: No such file or directory
1105098318 -rw-rw-r-- 1 douglas douglas 22 Jan 27 2015 hf
1915648234 rwxrwxrwx 1 douglas douglas 1 Aug 22 11:37 sf -> f
milou-b: ~/course $ mv hf f
milou-b: ~/course $ ls -li hf hf sf
ls: cannot access hf: No such file or directory
1105098318 -rw-rw-r-- 1 douglas douglas 22 Jan 27 2015 f
1915648234 rwxrwxrwx 1 douglas douglas 1 Aug 22 11:37 sf -> f
```
Manipulating names in bash

- `name=value` assigns `value` to `name`
- `$name` and `${name}` produce the value of `name`
- `${name}` can be useful in some contexts
  - `${name}_suffix` prefixes the value of `name` to `_suffix`
  - `$name_suffix` looks in `name_suffix` for a value
- `${name%pattern}` removes `pattern` from end of `name`
  - F=file.fa; echo ${F%.*} produces ‘file’
  - F=f.file.fa; echo ${F%.*} produces ‘f’
- `${name#pattern}` removes `pattern` from beginning of `name`
  - F=/home/douglas/file.fa; echo ${F#*/} produces ‘home/douglas/file.fa’
  - F=/home/douglas/file.fa; echo ${F##*/} produces ‘file.fa’
- How might one get just the directory part?

Manipulating names in bash

- Save a result to a filename with a modified suffix

```
milou-b: ~/course $ F=ref.fa; grep -c ">" "$F" > ${F%.fa}.count
milou-b: ~/course $ cat ref.count
```

- basename and dirname can also be helpful to get filenames and directory names

```
milou-b: ~/course $ F=/home/douglas/file.fa
milou-b: ~/course $ basename $F
file.fa
milou-b: ~/course $ dirname $F
/home/douglas
```

- ‘man basename’ and ‘man dirname’
File conversions

- Mac, Windows and Linux text files use different line endings
  - Linux: Linefeed
  - Mac: Carriage-return
  - Windows: Carriage-return + Linefeed

```bash
milou-b: ~/course $ cp /proj/g2016011/labs/linux2_additional-files/Workbook1.txt .
milou-b: ~/course $ cat Workbook1.txt
3  6milou-b: ~/course $ dos2unix Workbook1.txt
dos2unix: converting file Workbook1.txt to UNIX format ...
milou-b: ~/course $ cat Workbook1.txt
3  6milou-b: ~/course $ mac2unix Workbook1.txt
dos2unix: converting file Workbook1.txt to UNIX format ...
milou-b: ~/course $ cat Workbook1.txt
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6milou-b: ~/course</td>
<td></td>
</tr>
</tbody>
</table>
```