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

```bash
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
```

- `cat`
  - `con”cat”enate`: dumps the contents of a file
  - can also be used to quickly create a short file

```bash
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
```
Redirecting input:  

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

```
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

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

Redirecting input:  

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

```
imou-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 is a short file
this file is a little longer
```

– command line files can be more flexible

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

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

Here, cat first reads f, then
reads stdin, which has e
Redirecting output:  > and  >>

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

```
$ cat e > ee
this is a short file
$ ls -l e ee
-rw-r--r-- 1 douglas douglas 21 Aug 24 13:14 e
-rw-r--r-- 1 douglas douglas 21 Aug 24 17:00 ee
$ cat ee
```

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

```
$ cat f >> ee
this file is a little longer
$ ls -l e f ee
-rw-r--r-- 1 douglas douglas 21 Aug 24 13:14 e
-rw-r--r-- 1 douglas douglas 29 Aug 24 13:47 f
$ cat ee
```

Connecting stdout to stdin with the pipe:  |

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

```
$ cat f ee | cat > ff
this file is a little longer
this is a short file
this is a little longer
$ cat f | wc -l
1
$ cat ee | wc -l
2
$ 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

```
$ 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:

```bash
milou-b: ~/course $ bwa mem -a -E3 -t 8 ref.fa reads.fq | samtools view -Sb - > aln.bam
[m::main_mem] read 32 sequences (4016 bp)...
[m::main_mem] 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 `>`) and then capture the combined output stream with `>` file

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

  ```bash
  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

```bash
milou-b: ~/course $ bwa mem -a -E3 -t 8 ref.fa reads.fq 2> &1 | samtools view
  -Sb - > aln.bam
[samopen] no gSQ lines in the header.
[warn] sam_read1 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 ’.
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.

```
milou2: ~course $ ls
a b c d e ee f ff
```

- ‘?’ matches any 1 character

```
milou2: ~course $ ls ?
* a b c d e
milou2: ~course $ ls ?
ff
```

- ‘*’ matches 0 or more of any character

```
milou2: ~course $ ls *
* a b c d e ee f ff
```

Quote to match literally:
```
milou2: ~course $ touch "*" milou2: ~course $ ls "*"
* 
```

Wildcards: character groups with [ ] - ^

- You can specify character groups using [ ] - ^

```
milou2: ~course $ ls
a b c d e ee f ff
```

Match specific character: [a] Two characters: [af] Range: [a-f]
```
milou2: ~course $ ls [a]
a
milou2: ~course $ ls [af]
a f
milou2: ~course $ ls [a-f]
a b c d e f
milou2: ~course $ ls [a-f]?
ee ff
```

Anything but specific characters: [^a] [^a-d]
```
milou2: ~course $ ls [^a]
* b c d e f
milou2: ~course $ ls [^a-d]
* e f
milou2: ~course $ ls [^a-d]?
ee ff
```
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

```bash
milou-b: ~/course $ cp /proj/g2016011/labs/linux2_additional-files/ref*.fa
milou-b: ~/course $ ls reads fq ref fa
```

- Looking at their formats...

```bash
milou2: ~/course $ head -n 4 ref fa
>seq1
TGCTCTCTTTTGGTGCTAGGTGACTGCTGGGGTTCCAATCAATTGGGGCTTTG
TGCTTTTTGGGGGGCTAGCTACGGGTTGAGGGTTGCTTTTCAACACATCTCAAACT
ATTTTCTGAAGACAGTTCTACTGCTGCTTCGGCGGCTGAGACTGAATAAACTAAGAC
milou2: ~/course $ head -n 4 reads fq
@EU0PK:00025:00052
GAAGAACGCGGAAA
@BB7EBQQCCCCCBBCC
```

- 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 must be single-quoted so bash does not become confused (e.g., 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?

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

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

```
milou2: ~/course $ grep '^>' ref.fa | cut -c2-
seq1
seq2
seq3
seq4
seq5
seq6
seq7
milou2: ~/course $ grep '^>' ref.fa | cut -c4-5
e1
e2
e3
e4
e5
e6
e7
```
Find the line of a specific sequence

- ‘grep -n’ includes line (n)umbers
  milou2: ~/course $ grep -n "^>seq1" ref.fa
  1:>seq1
- for all matches
  milou2: ~/course $ grep -n "^>" ref.fa
  1:>seq1
  11:>seq2
  19:>seq3
  26:>seq6
  34:>seq5
  43:>seq6
  52:>seq7
- Only line numbers? ‘cut’ a (f)ield using a (d)elimiter
  milou2: ~/course $ grep -n "^>" ref.fa | cut -f1 -d':'
    1
    11
    19
    26
    34
    43
    52
  milou2: ~/course $ grep -n "^>" ref.fa | cut -f1 -d':' | tail -n 1
    52

What if we only want the line number for the last sequence?

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 '[457]'
  >seq1
  >seq2
  >seq3
- grep -F : (F)ixed expression, ignore wildcards
  milou2: ~/course $ ls
  * a b c d e e f ff reads.fq ref.fa
  milou2: ~/course $ ls -l | grep -F 'w'
  -rw-r--r-- 1 douglas douglas 0 Aug 25 15:11 *
  milou2: ~/course $ ls -l "w"
  -rw-r--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
A few more grep options

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

- grep -m N : stop output after N (m)atches
  
  ```bash
  milou2: ~/course $ grep -m 1 'q[0-9]' ref.fa >seq3
  ```

- grep -H : include the filename (default with >1 file)
  
  ```bash
  milou2: ~/course $ grep -Hm --color 'q[14]' ref.fa
  ref.fa:1<=seq1
  ref.fa:26:=>seq4
  milou2: ~/course $ cat ref.fa | grep -Hm --color 'q[14]' [standard input]:1<=seq1
  [standard input]:26:=>seq4
  ```

- grep -l, -L : only print fi(l)enames 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
  ```

Just a couple more grep options

- grep -B N : include N lines (B)efore the match in output
- grep -A N : include N lines (A)fter the match in output

```bash
milou2: ~/course $ grep -B 1 '^>' seq2$ ref.fa
TGTCAGGAGCCC
>seq2
milou2: ~/course $ grep -A 3 '^[@]QKNP:00685:00085$' reads.fq
@QKNP:00685:00085
GAGGATCTGATACATCCTGCA
+1@179895442444:4444999
```

- Just the sequence of that read? The quality string? The name of the next read?

```bash
milou2: ~/course $ grep -A 1 '^[@]QKNP:00685:00085$' reads.fq | tail -n 1
GAGGATCTGATACATCCTGCA
milou2: ~/course $ grep -A 3 '^[@]QKNP:00685:00085$' reads.fq | tail -n 1
1@179895442444:4444999
milou2: ~/course $ grep -A 4 '^[@]QKNP:00685:00085$' reads.fq | tail -n 1
@QKNP:01068:00786
```

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

- $(<\text{file})$ replaces the whole $( ... )$ with the contents of file
- $(\text{command})$ replaces $( ... )$ with the output of command

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

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
  0 ~
  1 ~/course
```

- Edit the ‘.bashrc’ configuration file with nano, add the line

```bash
alias rm='rm -i'
```

A similar line may already be there, check first!

- Move back to previous location with ‘popd’

```bash
milou2: ~ $ popd
~/course
milou2: ~/course $ dirs
~/course
```
Load an UPPMAX module with some tools

```
milou-b: ~ $ module load tinyutils
milou-b: ~ $ which hist
/sw/apps/tinyutils/1.3/milou/hist
milou-b: ~ $ which table
/sw/apps/tiny_utils/1.3/milou/table
```

How long are my fasta sequences?

- Use the ‘fastalength’ tool from the exonerate module
  - module load bioinfo-tools
  - module load exonerate

```
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

```
milou-b: ~/course $ fastlength ref.fa | cut -f1 -d' '
milou-b: ~/course $ fastlength ref.fa | cut -f1 -d' ' | sum
milou-b: ~/course $ fastlength ref.fa | cut -f1 -d' ' | mean
milou-b: ~/course $ fastlength ref.fa | cut -f1 -d' ' | median
milou-b: ~/course $ fastlength ref.fa | cut -f1 -d' ' | max
milou-b: ~/course $ fastlength ref.fa | cut -f1 -d' ' | min
```

What is the length distribution of my reads?

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

```
milou-b: ~/course $ grep "^[ACGTN]$" reads.fq | head -n 3
GAAGACGGCGAGGAA
GGAGAAGCCGGCGAGGAA
GGAGAAGCCTCTCTGCTGATGATGGAAGAACCGCGGAA
milou-b: ~/course $ grep "^[ACGTN]$" reads.fq | awk '{ print length($0) }' | head -n 3
15
15
45
milou-b: ~/course $ grep "^[ACGTN]$" reads.fq | len | head -n 3
15
15
45
milou-b: ~/course $ grep "^[ACGTN]$" reads.fq | len | table
45 7
46 1
19 1
37 1
47 2
22 1
15 25
25 1
milou-b: ~/course $ grep "^[ACGTN]$" reads.fq | len | table | sort
15 25
19 1
22 1
25 1
37 1
45 7
46 1
47 2
```
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 lwxrwxrwx 1 douglas douglas 1 Aug 22 11:37 sf -> f
```

‘Hard links’ (ln 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 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 lwxrwxrwx 1 douglas douglas 1 Aug 22 11:37 sf -> f
milou-b: ~/course $ rm f
rm: remove regular file ’f’? y
milou-b: ~/course $ ls -li hf sf
ls: cannot access hf: No such file or directory
1105098318 -rw-rw-r-- 1 douglas douglas 22 Jan 27 2015 hf
1915648234 lwxrwxrwx 1 douglas douglas 1 Aug 22 11:37 sf -> f
milou-b: ~/course $ mv hf f
milou-b: ~/course $ ls -li hf sf
ls: cannot access hf: No such file or directory
1105098318 -rw-rw-r-- 1 douglas douglas 22 Jan 27 2015 f
1915648234 lwxrwxrwx 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%.fa}` 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

```
milou-b: ~/course $ cp /proj/g2016011/labs/linux2_additional-files/Workbook1.txt .
milou-b: ~/course $ cat Workbook1.txt
3
milou-b: ~/course $ dos2unix Workbook1.txt
dos2unix: converting file Workbook1.txt to UNIX format ...
milou-b: ~/course $ cat Workbook1.txt
3
milou-b: ~/course $ mac2unix Workbook1.txt
dos2unix: converting file Workbook1.txt to UNIX format ...
milou-b: ~/course $ cat Workbook1.txt
```

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>