# UNIX Utilities

Thomas Zwinger

CSC – Tieteen tietotekniikan keskus Oy
CSC – IT Center for Science Ltd.

# Contents

- Basics
  - `more` or `less`
  - pipes
  - `stderr`, `stdout`
  - redirecting
- How to find stuff
  - regular expressions
  - `grep`, `fgrep`, `egrep`
  - `find`
- Simple text utilities:
  - `paste`, `cut`
  - `wc` counting lines/ words/chars
  - `cat` with `tail` and `head`
  - `sort`, `uniq`

# Before we start

- This course is based on **bash**
- CSC's default login shell is **tcsh**[1)]
- Ways of changing on CSC's servers:
  1 Temporarily invoke bash:
  ```
  > /bin/bash -l
  ```
  2 Make bash your default:
  ```
  > chsh
  ```

```
yourid@hippu2:~> chsh
Changing shell for yourid.
Password: yourunixpassword
New shell [/bin/tcsh]: /bin/bash
Shell changed.
```

1) please, do not ask why

# More or less

- `more` *`filename`*
  - Displays the contents of a file screen-by-screen
  - Press `h` for the help-screen
  - Searching for regular expression:

    `/`*`regexp`* and `n` (next occurrence)
  - Space or *z* for next screen
  - Return for forward line scroll
  - b or Ctrl.+b for back
  - q for quitting (else, automatic after last screen)

# More or less

- `less` *`filename1 filename2`*
  - `less` is more than `more`
  - `h` for help screen
  - Searching: `/`*`pattern`* or `?`*`pattern`*
  - Continue search: `n, N` or `ESC-N`
  - Matching lines: &*`pattern`*
  - Goto line number: *`linenumber`* `g`
  - Multiple files: `:n` (next) `:p` (previous)

# Pipes

- In UNIX/LINUX commands can be put into a chain (i.e., output of the earlier is provided to the next):

```
command1|command2
```

- An arrangement of commands like the one above is called a **pipe**, the  symbol | is called the pipe-symbol

- Example: amount of files/directories in `/dev`[1]:

```
ls -d /dev/* |wc -w
```

1) detailed introduction to wc, see later

# Redirection

- Redirection of stdout: `command 1> filename`
  - creates/overwrites *`filename`*
  - Writes stdout to *`filename`* and stderr to screen
  - Same result: `command > filename`

- Redirection of stderr: `command 2> filename`
  - creates/overwrites *`filename`*
  - Writes stderr to *`filename`* and stdout to screen

# Redirection

- Redirection/splitting of stderr and stdout into separate files:

```
command 1> filename.out 2> filename.err
```

  - stdout to `filename.out`
  - stderr to `filename.err`

- Combining of stderr and stdout into single file:

```
command &> filename.errout
```

# Redirection

- Adding instead of overwriting:

```
cat filename.out >> filename.err
```

  – Appends contents from (hopefully existing) `filename.out` to (existing or else new created) `filename.err`

- Hint: Elegantly getting rid of unneeded/-wanted output:

```
command 1> /dev/null
```

  – Only displays errors and disposes stdout

# Regular Expressions

- For searching (e.g., `{e,f,r}grep`)
- Any word is a regular expression: e.g.,

```
ls -d /dev/*|grep 'sda'
```

  - Will find any (non-hidden) files/directories in /dev that contain ***sda*** in their name
- Brackets [] describes a  character list:

```
ls -d /dev/*|egrep '[1_]'
```

  - Will find any (non-hidden) files/directories in /dev that contain either **1** or _ in their name

# Regular Expressions

- Single character wildcard:

```
ls -d /dev/*|egrep '.da'
```

  - Will find any files in /dev that contain a 3 character expression which ends with *da* (e.g., *s**da**1*, *h**da**1*, *...*)

- The ^ operator excludes (in [] brackets):

```
ls -d /dev/*|grep tty|egrep '[^S]'
```

  - Finds files in /dev including sequence **tty** but only highlights everyting but **S**

# Regular Expressions

- Beginning of the line ^:

```
ls -d /dev/*|egrep '^/dev/s'
```

  – Will find any files/directories in /dev that start with a ***s***

- End of the line $:

```
ls -d /dev/*|egrep 'a$'
```

  – Will find any files/directories in /dev that end with an *a* (e.g., */dev/sd**a***)

- * preceding character 1 or more times:

```
ls -d /dev/*|egrep '[o*]'
```

  – Displays all files in /dev containing *o*, *oo*, ...

# Regular Expressions

- Character classes:

| Expression | Purpose |
|---|---|
| [:alpha:] [:digit] | Any alphabetic/digit character |
| [:lower] [:upper:] | Anny lower/upper case character |
| [:space:] | Any white-space character (space, tab, ...) |
| [:print:] [:cntrl:] | Any printable/non-printable character |

```
cat file|egrep '[[:upper:]]'
```

  – displays all lines of *file* with upper-case letters

# The `grep`-family

- **`grep`**: matches patterns in a file/stdout and outputs the corresponding line
  - We already had a fair amount of examples
  - Some options:
    - `-i` ignores upper/lowercase
    - `-v` invert match, e.g.,

```
ls -d /dev/*|grep -v 'sd'
```

      all filenames not containing ***sd***
    - `-f patternfile` obtain the pattern from a file
    - `-w` only lines that contain the pattern as whole words

```
ls -d /dev/*|grep -w 'sda'
```

# The grep-family

- **`egrep = grep -E`**
  - `-E` option for multiple patterns, e.g.,

```
ls -d /dev/*|grep -E '(a$|sd)'
```

    searches for files, which either end with an ***a*** or contain ***sd*** in their names

- **`fgrep = grep -F`**
  - Interprets the pattern as string (so no regular expressions). Spot the difference:

```
ls /etc|grep ..conf
ls /etc|grep -F ..conf
```

# The grep-family

- **`rgrep = grep -r`**
  - `-r` option for recursive search, e.g.,

```
rgrep network /etc/init.d
```

searches recursively for file containing the pattern network (`grep` would not find a single occurrence, as all files are in sub-directories of `/etc/init.d`)

# How can I `find` my files

- **`find /path/to/start/from expression`**
  - `-name` *`pattern`* filename search pattern, e.g.,

```
find /etc/ -name \*.conf 2> /dev/null
```

    searches recursively in `/etc` for files with suffix `.conf`
  - Options:
    - `-mindepth/-maxdepth` *`n`* minimum/maximum depth *`n`* in search tree relative to the path given as first argument
    - `-fprint` *`file`* / `-print` print to file ore stdout (default)
    - `-exec` *`command`* `{} \;` executes command

```
find /etc/ -name \*.conf -exec ls -l {} \;
```

# How can I `find` my files

- **`find + xargs`**
  - Finding certain files and piping the result to another command, e.g., `fgrep` to search for strings in them
  - List can be too long to fit into buffer of shell
  - xargs takes care and eventually chops the data into suitable pieces

```
find /etc/ |xargs fgrep network
```

  - Checks all files in /etc for the keyword
    (just like the earlier shown rgrep command)

# Paste

- **paste** [-d *del* -s] *file1 file2* ...

  merge lines of files separated by tabs

  -d *del* use other delimiter than tabs

  -s sequential (if more than two input files) processing: [(file1 + file2) + file 3] + file4

```
paste count.txt sheep.txt > sheep_tab.txt
```

merges count and sheep using tabs as delimiters

```
paste -d ' ' count.txt sheep.txt > sheep_space.txt
```

same, but with space as delimiters

# Cut

- **`cut`** `[-b -c -f -d` *`del`* `-s]` *`file1 file2`*

  `...`

  remove sections from each line of files

  `-b` byte list for selection

  `-c` character list for selection

  `-f` select fields

  `-d` *`del`* use other delimiter del than tabs

  `-s` skip lines not containing delimiters

```
cut -d ' ' -f 1 sheep_space.txt
```

prints 1st column (i.e, original count.txt file contents)

# Counting words, lines, bytes and chars [and sheep]

- **wc [-l -w -c -m]** *filename*

  `-l` for counting lines:

```
wc -l sheep_space.txt
```

  how many (lines in) sheep?

  `-w` counts the words

  `-c` counts the bits

  `-m` counts the characters

# Cats

- **`cat`** `[-n -E -v -T -s]` *`file1 file2`*

  concatenates files and prints to stdout

  `-n` numbering output lines (e.g., source-code listing)

  `-E` indicate ends with a $

  `-v` show non-printing

  `-T` indicate tabs

  `-s` squeeze blanks (remove repeatedly empty lines)

```
cat -n sheep_space.txt|cut -f 1| paste
sheep_space.txt - > sheep_lines.txt
```

numbers the lines in sheep_space.txt and adds the column to the new file

# Tails and Heads

- **`tail`** **`[-n`** ***`N`*** **`-f --pid`** ***`pid`*** **`]`** *`filename`*

  output the end of a file

  - `-n` *`N`* for display *N* last lines
  - `-f` continuously display file while being appended (very useful if following log-files)
  - `-pid` *`pid`* send tail command after process with ID *pid* has ended

- **`head`** **`[-n`** ***`N`*** **`]`** *`filename`*

  output head of a file

  - `-n` *`N`* for display *N* first lines

# Sort

- **sort** **[-d -f -g ]** *file1 file2* **...**

  sort lines of text files

  `-d` dictionary (alphanumeric) order

  `-f` ignore upper/lower case

  `-g` general numeric

  – Spot the difference:

```
sort -d sheep_line.txt
```

```
sort -g sheep_line.txt
```

# Uniq

- **uniq** **[-c -f -s -w ]** *file1 file2* ...

  filter adjacent lines

  `-c` prefix lines by number of their occurrence

  `-f` *N* avoid comparing the first *N* fields

  `-s` *N* same with chars

  `-w N` general numeric

```
uniq -c -f 1 -w 10 sheeps_line.txt
```

skips the first column (the previously inserted numbers) and matches in max. 10 characters (i.e., avoiding the later columns) and prefixes the number of occurrence