COMP09024 Unix System Administration

Lecture 5: Redirection, Pipes and Filters

Duncan Thomson/Hector Marco

UWS

Trimester 1 2020/21

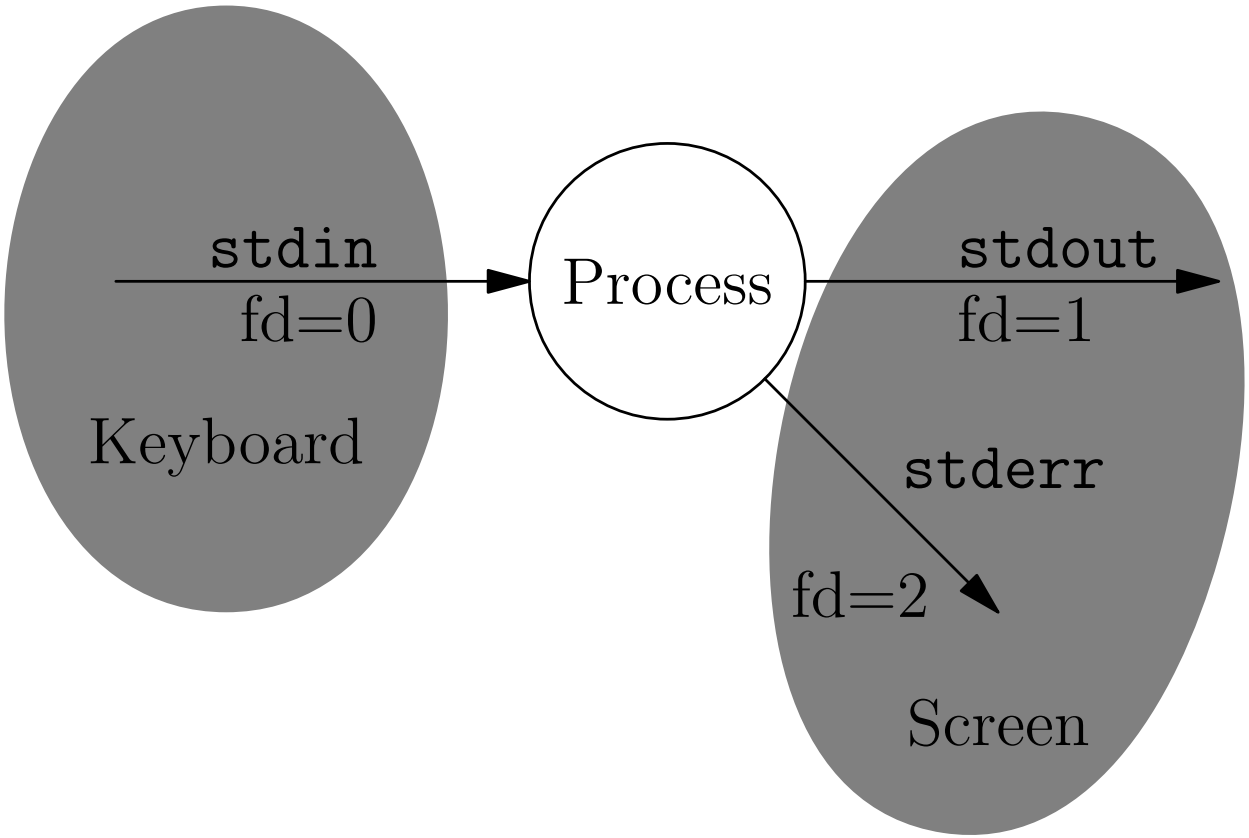
Outline

5.1 Redirecting I/O

Process Input and Output

- Most programs expect to:
  - Take some kind of input (maybe from a file)
  - Process the data
  - Send the results to some kind of output (perhaps a file)
- Some processes may also produce error messages
- Files — or other inputs or outputs in Unix (everything is a file!) — are identified by a *file descriptor*
- By default, each processes is started with three *standard streams*:
  - *Standard input* (`stdin`), identified by file descriptor 0
  - *Standard output* (`stdout`), identified by file descriptor 1
  - *Standard error* (`stderr`), identified by file descriptor 2
- By default, these are the keyboard and the screen

Standard Streams



Redirection of Streams

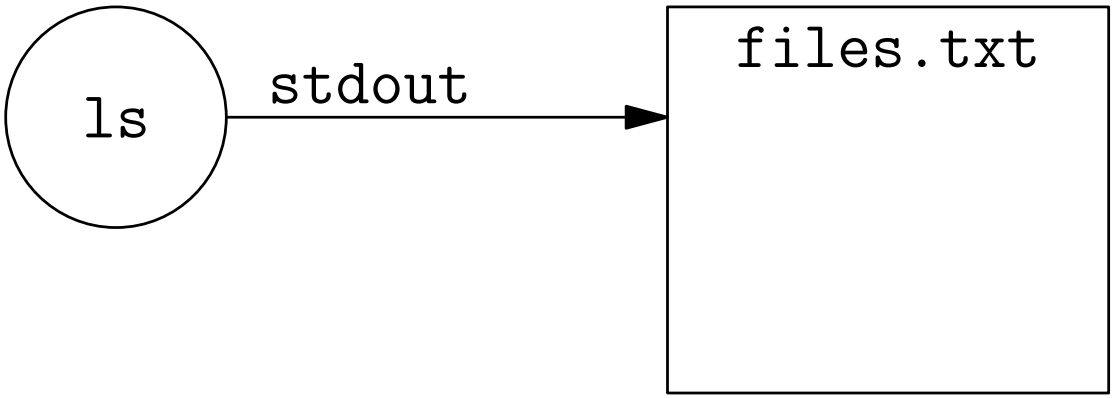
- A key concept of Unix is that these standard streams can be *redirected*
- These can be redirected to/from anything which can be described by a file descriptor, including:
  - Sending `stdout` output to a file
  - Sending `stderr` output to a file
  - Taking `stdin` input from a file
  - Taking `stdin` input from `stdout` (or `stderr`) of another process
  - Sending `stdout` (or `stderr`) output to `stdin` of another process
- Many programs are written specifically to allow chaining together in this way, and are often known as *filters*

5.2 Stream Redirection

Redirecting `stdout` to a File

- To redirect `stdout` to a file, the `>` operator is used
- Example: redirecting the output of `ls` to the file `files.txt`

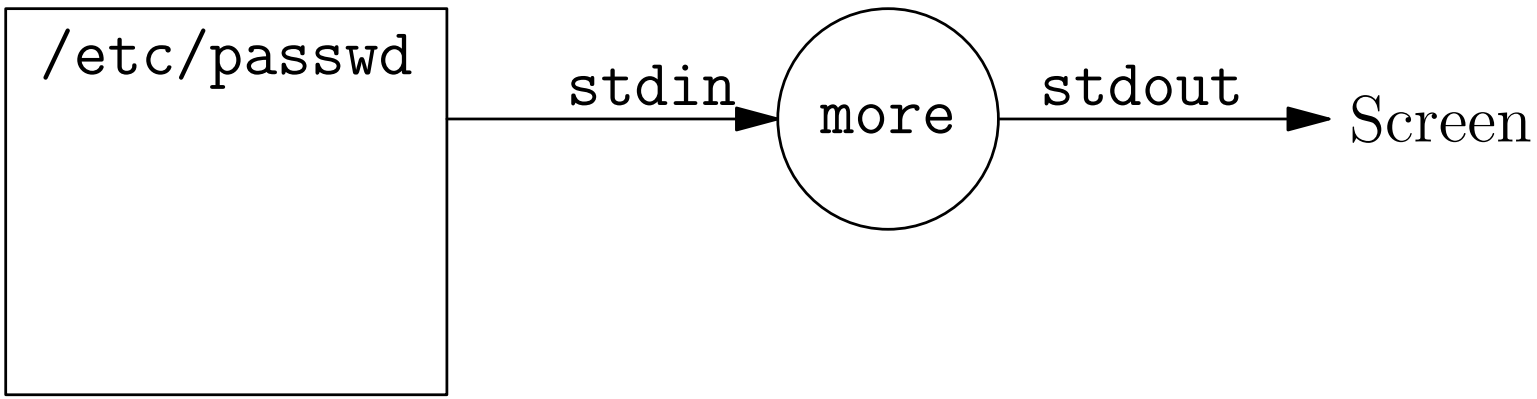
```
user@debian:~$ ls -l >files.txt
```



Redirecting stdin from a File

- To redirect stdin from a file, the < operator is used
- Example: redirecting the input of more from the file /etc/passwd (bit of a silly example, since we could just do more /etc/passwd)

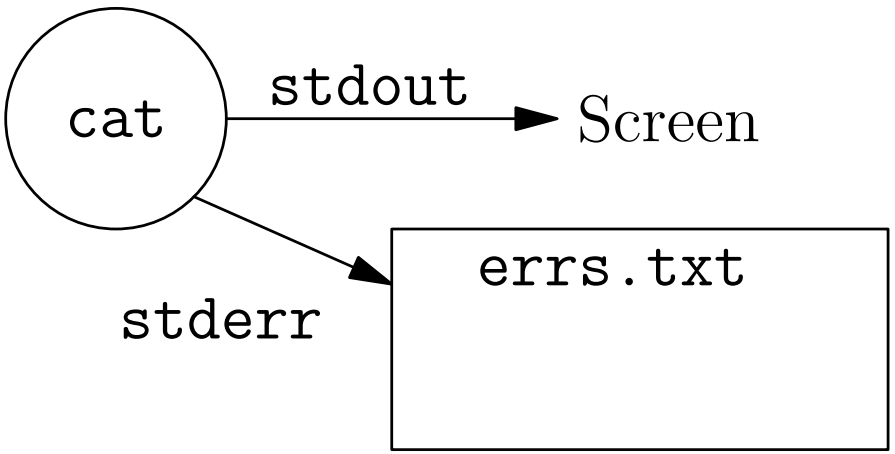
```
user@debian:~$ more </etc/passwd
```



Redirecting `stderr` to a File

- To redirect `stderr` to a file, the `2>` operator is used
  - Here, the `2` references `stderr`
- Example: redirecting errors from the `cat /etc/*` command to the file `errs.txt`

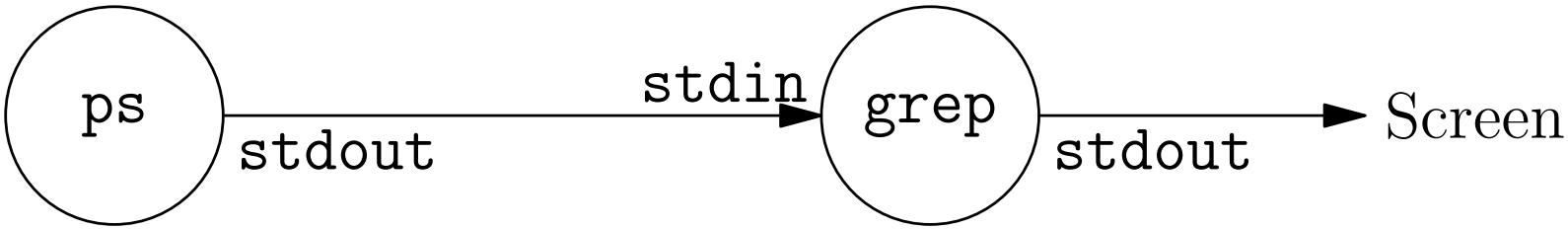
```
user@debian:~$ cat /etc/* 2>errs.txt
```



The Pipe Operator

- The | operator connects stdout of the first process to stdin of the second
- Example: search for the process name init in output from ps
  - The grep command searches for a string

```
user@debian:~$ ps -ef | grep init
```



Advanced Redirection

- Additional redirections may be possible, eg `bash` allows:
  - Appending `stdout` to end of an existing file with `>>`
  - Redirecting `stderr` into `stdout` with `2>&1`
  - Redirecting `stderr` to another command with `|&`
  - Redirecting both `stdout` and `stderr` to a file with `&>`
  - A 'here-document': taking `stdin` of a command from `stdin` (ended by a specified string, eg 'EOF') with `<<EOF`
- There are also some special device files, for example:
  - `/dev/null` discards anything sent to it
  - `/dev/zero` provides en endless string of zeros
  - `/dev/random` provides (pseudo)-random data
- Possible to combine redirections in a command string, eg:

```
user@debian:~$ ps -ef | grep init | more
user@debian:~$ who | grep alice >alice.txt
```

5.3 Commands and Exit Status

Command Strings

- Sequential execution of commands one after another (`command2` is run after `command1` has completed)

```
command1 ; command2
```

- Parallel execution of commands (`command1` in background):

```
command1 & command2
```

- Pipelined commands (moving `stdout` of `command1` to `stdin` of `command2`):

```
command 1 | command2
```

Exit Status

- Every process when it terminate has an *exit status* — an integer indicating the outcome of the process
- This can be accessed using the variable `$?`
- The value is usually `0` for 'success', and may take other values for other occurences
- For example:

```
user@debian:~$ grep -sq root /etc/passwd ; echo $?
0
user@debian:~$ grep -sq raat /etc/passwd ; echo $?
1
user@debian:~$ grep -sq root /etc/pass ; echo $?
2
```

Conditional Command Strings

- There are also two ways of combining commands 'conditionally' — depending on the exit status of the first
- 'AND' command combination uses `&&`:
  - `command1 && command2`
  - `command2` is only run if `command1` has a zero exit status
- 'OR' command combination uses `||`:
  - `command1 || command2`
  - `command2` is only run if `command1` has a non-zero exit status
- Overall exit status in both cases is status of last command run

5.4 Filters

Filters

- A large number of Unix tools are designed to operate by taking data on standard input, transforming it in some way, and sending it out to standard output
- Such programs are known as *filters*
- Generally follow the Unix philosophy of 'do one thing well'
- We have already come across a couple:
  - `cat` — takes standard input (or a list of files) and concatenates them onto standard output
  - `grep` — searches for a string in standard input, prints matches on standard output (we'll examine this more closely in a moment)
- We'll now look at some Unix filters

Searching with grep

- grep (stands for General Regular Expression Parser) searches for strings (or *regular expressions*) in standard input (or files)
- Matching lines are sent to standard output
- A wide range of options are available, including:
  - -v prints non-matching lines
  - -i matches case-insensitively
  - -F or -E match fixed strings or *extended* regexps
  - -l (or -L) only list filenames with (or without) matches
  - -c only output count of matching lines (per file)
  - -q (and -s) suppress output (or error messages)
  - -A and -B include context after and before matches

Regular Expressions

- (Basic or extended) *regular expressions* (regexp or regex) are used for matching patterns in strings using special purpose characters (see `man grep`)
- `.` matches any single character
- `[`*chars*`]` matches any character listed (which may include ranges like `[a-z]` or `[0-9]`)
- `[^`*chars*`]` matches any character except those listed
- `^` and `$` match beginning and end of line respectively
- `?` matches 0 or 1 of preceding character
- `*` matches 0 or more of preceding character
- `+` matches 1 or more of preceding character
- `{`*n*`}` matches eactly *n* of preceding character
- `(`*re1*`|`*re2*`)` matches either of the subexpressions

Examples of Regular Expressions

- `abcde` — matches the string `abcde`
- `abcd.` — matches any 5-char string starting with `abcd`
- `Thomp?son` — matches either Thomson or Thompson
- `Gra(eme|ham)` — matches either Graeme or Graham
- `[0-9A-F]` — matches any (uppercase) hexadecimal digit
- `^[^a-z]` matches any line not starting with a lower case letter
- `[Ww]ee+` — matches ‘Weeee’, ‘wee’, ‘weeeeeeee’ (but not ‘we’ ‘woooo’ or ‘Wii’)
- `07[0-9]{9}` — matches a UK mobile phone number
- `[A-Z][A-Z]?[1-9][0-9]? [0-9][A-Z]{2}` — matches UK postcode

Examples using grep

Some examples using grep:

```
user@debian:~$ cat file.txt
s1 newa e1
s2 newb e2
s3 newc e3
s4 newd e4
user@debian:~$ grep new[ac] file.txt
s1 newa e1
s3 newc e3
user@debian:~$ grep new[a-c] file.txt
s1 newa e1
s2 newb e2
s3 newc e3
user@debian:~$ grep -E "(e1|e2)" file.txt
s1 newa e1
s2 newb e2
```

Selecting text with `cut`

- The `cut` command picks specific characters or fields
- It either operate in byte, character or field mode:
- In character mode (`-c`), a list of characters is provided:
  - Ranges are specified by using the `-` symbol
  - More than one range may be given, spearated by commas
  - Example: `dpkg -l | cut -c5-40` shows a list of package names (only) installed on Debian
- In field mode, (`-f`) a list of fields is given
  - Field separator is TAB or provided with `-d`
  - Several ranges may be provided, as with `-c`
  - Example: `cut -d: -f1,5 /etc/passwd` shows usernames and with full user's name and contact details

sort and uniq

- sort sorts lines in stdin (or a file)
  - -a performs an alphabetical sort
  - -g performs an general numerical sort
  - -R sorts randomly
  - -M sorts by month name
  - -r reverses order of sort
  - -k specifies which 'key' (field) to sort on
  - -t specifies field separator character
  - Example: `sort -t: -k3 -g /etc/passwd` — sorts /etc/passwd numerically by UID field
- uniq searches for unique or duplicate lines in sorted input (and removes duplicates by default)
  - -d only prints duplicates
  - -u only prints unique lines

head, tail and tac

- The head command by default prints the first 10 lines of stdin or a file
  - The number of lines can be given with -
- The tail command by default prints the last 10 lines of stdin or a file
  - The number of lines can be given with -
  - The -f allows 'following' of a file as it is added to
- The tac command prints the lines of stdin (or file) in reverse order
  - -s specifies a separator other than newlines
- Example: head /etc/passwd | tail -5 prints lines 6-10 of the /etc/passwd file

wc and tee and tr

- The wc command counts characters, words and/or lines
  - -c prints a count of characters
  - -w prints a count of words
  - -l prints a count of lines
- The tee command copies stdin to stdout, sending a copy into a file specified (like a T-piece in a pipe)
- tr allows translating from one set of character to another
  - -d deletes specified characters instead
- Example: `tr a-z A-Z <file.txt | tee upper.txt | wc -w` puts an uppercase version of file.txt into upper.txt and prints a word count of the result

- `sed` allows arbitrary edits to be made to data
- A very powerful tool, but not easy to understand
- Commands can be provided to it either
  - Using `-e` to give a command on the command line
  - Using `-f` to specify a file containing commands
- Each command consists of:
  - An optional address or address range specifying which lines to edit
  - A command, which is executed for all matching lines
- A `sed` script may include multiple commands
- An example `sed` command: `sed -e "/hello/y/a-z/A-Z/"` — this capitalises all letters on lines which contain the word ‘hello’

sed Addresses and Commands

- Some methods of specifying lines include:
  - Line number, eg `10`
  - Line number range with a comma eg `10,20`
  - A regular expression using `//` as delimiters, eg `/^[0-9]/` will match all lines starting with a digit
- Commands which can be executed include:
  - `a` appends text after matching lines
  - `i` inserts text before matching lines
  - `c` changes lines for new text
  - `d` delete lines
  - `s/`*`regexp`*`/`*`string`*`/` substitutes *string* in place of *regexp*
  - `y/`*`chars`*`/`*`chars2`*`/` substitutes characters in *chars2* for those in *chars1* (like `tr`)

5.5 More Unix Commands

Finding Files with `find`

- The `find` command an find files matching specific criteria
- The syntax is: `find` *`path criteria action`*
- The path is the starting point in the filesystem
- Criteria can be combined, and include:
  - `-name` to match on a filename
  - `-atime`, `-ctime` or `-mtime` to match access, change or modification times ('equal', 'after' or 'before' a number of days; there are other variations)
  - `-user` or `-uid` to match the owner
  - `-group` or `-gid` to match a group owner
  - `-perm` to match on file permissions
  - `-size` to match on file size
  - `-type` to match by type (directory, link, etc)
- Actions can include `-print` or executing commands with `-exec`

Archiving with `tar`

- The `tar` command allows creation of archives of multiple files (from Tape ARchive)
- `f` options specifies archive file (almost always used)
- Other options include:
  - `c` to create an archive (from the list of files given)
  - `x` to extract files from an archive
  - `t` to print a table of contents
- `tar` archive files normally end in `.tar`
- `tar` is only one of many archival programs, which include `dd`, `cpio` and others depending on system
- Example: tar xzf archive.tgz uncompresses and extracts all files from the file `archive.tgz`

Compressing Files: `gzip`

- A number of compression utilities are available
- `gzip` is one of the most popular
- Default is to compress the given file (giving it a `.gz` extension)
- Level of compression can be controlled with `-1` to `-9`
- Compressed files are uncompressed with `gunzip`
- There are also some special variants of commands which operate directly on compressed files, eg `zcat`, `zmore`, `zgrep`
- Some systems may have other compression commands, for example:
  - `compress` and `uncompress` (the original Unix one)
  - `bzip` and `bunzip`

Comparing Files

- A number of commands are available to compare files
- `diff` is the most widely used — reports on differences between text files
- Can provide context lines
- Is widely used to produce 'patch' files (which can be used to update source code using the `patch` command)
- The `cmp` utility also compares files

Calculations

- There are two command line calculators widely available in Unix
- `dc` is a reverse Polish notation (stack-based) calculator
- `bc` is more convenient-to-use calculator
- Both of these provide the user with command-line interface to the calculator
- `dc` also allows providing a script directly on the command line
- The `expr` command is more useful for shorter calculations such as might be used in shell scripts (next week)
- Example: `expr 2 + 2` gives the answer 4

Summary

- Process input and output
- Standard process streams: `stdin`, `stdout` and `stderr`
- Redirection
- File redirection: `>`, `<`, `>>`, `2>` and `<<`
- Pipelines and the pipe operator `|`
- Command exit status
- Chaining commands: `;`, `&`, `&&` and `||`
- `grep` and regular expressions
- Filters: `cut`, `head`, `tail`, `tee`, `sort`, `sed`, `tr` and others
- Finding files matching criteria with `find`
- Archiving and compressing: `tar` and `gzip`
- Finding differences in files: `diff` and `patch`
- Calculations with `expr`, `dc` and `bc`