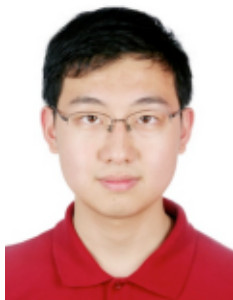


CS2103 PROJECT MANUAL

Command line task manager

```
c:\Documents and Settings\u0909122\My Documents\Visual Studio 2008\Projects\TaskM... Task Manager U0.2
< Ctrl-c >: exit < H >: show help
-----default-----
9      Number:      9      Deadline:      Wed Oct 27 10:45:00 2010
      Priority:      0      Status:      Finished
      Details:
      export bug: cannot export to replace existing file
-----
10     f default    Add a input module to handle user interaction. enabl...
11     f default    Enable pipe e.g ls -k Parser* ! edit -g Parser, this...
12     f default    Enable export -html
13     f default    Use XML standard entity references to cope with symb...
14     f default    Configuration reader.
15     f default    Enable args e.g. ./taskM
19     f default    Refractor the code: sepe
Oct 2010
Week  : Su Mo Tu We Th Fr Sa
39    : 26 27 28 29 30  1  2
40    :  3  4  5  6  7  8  9
41    : 10 11 12 13 14 15 16
42    : 17 18 19 20 21 22 23
43    : 24 25 26 27 28 29 30
44    : 31  1  2  3  4  5  6
+--Today is 6 Nov 2010----- 16 of 16 Tasks --+
ls
```



He Haocong

U099121H



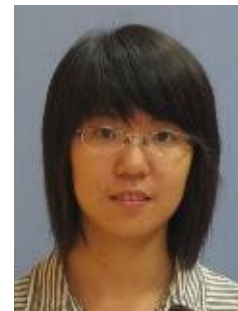
Liu Jialong

U099122U



Wang Xiangyu

U099120W



Zhou Biyan

U094837M

November 6, 2010

Contents

1	Introduction	2
2	User Guide	3
2.1	Quick Start	3
2.2	More commands	4
2.2.1	read, import and export	4
2.2.2	task	6
2.2.3	pri	6
2.2.4	edit	7
2.2.5	undo and redo	7
2.3	Using Options	7
2.3.1	add	7
2.3.2	ls	8
2.3.3	rm	9
2.4	The Text-Based Interactive User Interface	9
2.5	Advanced Usages	11
2.5.1	command piping	11
2.5.2	command mapping	11
2.5.3	taskManager script	12
2.5.4	startup script	13
2.5.5	talk to taskManager	13
2.6	Compilation and Installation	14
2.6.1	Microsoft Windows	14
2.6.2	Unix-like Operating Systems	14
3	Developer Guide	15
4	Milestones and Individual Work	16

1 Introduction

This is the manual for a command-line utility (referred to as taskManager) that manages one's to-do's.

2 User Guide

2.1 Quick Start

This section introduces you the minimum amount of commands to get started.

1. Start taskManager shell:

On Mac OS and GNU Linux (referred to as “*nix” later), taskManager can be started from shell by commands:

```
$ cd the/folder/containing/taskManager
$ ./taskManager1
```

On Microsoft Windows, taskManager can be launched in command prompt as well, or simply by double clicking “taskManager.exe”.

Once taskManager is launched, you will see a prompt like ">_", and you will start typing commands!

2. Add some tasks:

```
$ ./taskManager
> add "Sample task 1"
> add "Sample task 2"
TaskManager: This task is highly similiar to some existing task,
do you really want to add it? y
```

To add a task, simply use `add` command followed by the description of the task in a pair of quotation marks.

If no error messages are shown, the task is successfully added. TaskManager may prompt for confirmation if the task to be added is highly similar to some existing task(s) to help prevent people forget adding tasks, which is the case in the example above.

3. List the existing tasks:

```
> ls
1      Sample task 1
2      Sample task 2
```

To see the existing tasks, use `ls` command. By default, the taskManager shows the serial numbers and the descriptions of the tasks.

4. Mark a task as finished:

```
> finish 2
> ls
1      Sample task 1
2 f    Sample task 2
```

¹ *nix version can be run at any directory after installing taskManager – “`make install`”. See details in section “`Compilation and Installation`”.

To finish an existing task, use `finish` command followed by the serial number of the task to finish. Notice that for finished task, an 'f' is shown between serial number and task description.

5. Remove task(s):

```
> ls
1      Sample task 1
2 f    Sample task 2
> rm 1
TaskManager: Do you really want to remove this task permanently? y
> ls
2 f    Sample task 2
```

To remove an existing task, use `rm` command followed by the serial number of the task to remove. TaskManager will prompt for confirmation when removing tasks.

6. Exit from taskManager:

```
> exit
```

To quit from taskManager, use `exit` command. All changes to the existing tasks will be automatically saved.

2.2 More commands

2.2.1 read, import and export

TaskManager stores the tasks in an XML file which is by default `~/record.xml` on *nix, and `%USERPROFILE%\record.xml` on Windows.

TaskManager also supports importing/exporting the existing tasks from/to XML and HTML files. This is done by `read`, `import` and `export` commands.

`read` reads an XML file, list all the tasks it contains without affecting the current task list.

This is helpful when you only want to peek the content of an xml file without really importing it.

```
> ls
1      Sample task 2
> read midterms.xml2
1 f    CS2103 midTerm Sep 29 06:30 - 07:30 pm MPSH 1B
2 f    CS3230 midTerm Oct 15 06:00 pm
3 f    CS3241 midTerm Oct 07 lecture
4 f    CS3244 midTerm Oct 04 lecture
5 f    ST2132 midTerm Oct 08 LT33 12:15 - 1:30 pm
> ls
1      Sample task 2
```

`import` is similar to `read` command. It reads the content of the XML file and appends all the tasks in it to current task list.

```
> ls
1   Sample task 2
> import mytasks.xml
1   Sample task 2 3
2 f  CS2103 midTerm Sep 29 06:30 - 07:30 pm MPSH 1B
3 f  CS3230 midTerm Oct 15 06:00 pm
4 f  CS3241 midTerm Oct 07 lecture
5 f  CS3244 midTerm Oct 04 lecture
6 f  ST2132 midTerm Oct 08 LT33 12:15 - 1:30 pm
```

`export` exports the current task list to an XML or HTML file.

```
> export sampletasks.xml4
> exit
$ cat sampletasks.xml
<taskList>
<task>
<serialNumber> 1 </serialNumber>
<deadline> 1288473083 </deadline>
<priority> 0 </priority>
<description> Sample task 2 </description>
<group> default </group>
<isFinished> 0 </isFinished>
</task>
</taskList>
```

`export` can also be used to generate an HTML file which is more visually pleasant in your favourite browser.

```
> export -html sampletasks.html5
> exit
```

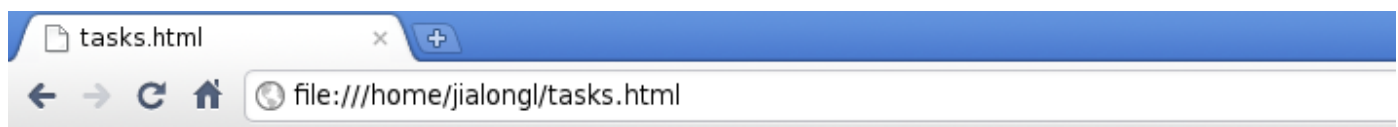
² The file name is not quoted. If the file name contains space, please quote it with a pair of quotation marks.

³ Task 1 is still in task list. Importing tasks will not erase existing tasks.

⁴ Currently `export` does not support environmental variables in path. i.e. `export ~/abc.xml` will not export the file to user's home directory `/home/username/`.

⁵ If the name of the HTML file to export is not specified, the default file name "`record.html`" is used.

⁶ Page may not render correctly in IE 6 or its earlier versions.



NUMBER	PRIORITY	GROUP	STATUS	DEADLINE	DETAILS
1	0	default	Doing	Wed Oct 27 10:41:24 2010	Enable Help/Man
2	0	default	Finished	Thu Oct 28 19:16:18 2010	Enable shell style auto-completion
3	0	default	Doing	Wed Oct 27 10:41:56 2010	Test and strengthen Parser to str
4	0	default	Finished	Wed Oct 27 10:42:12 2010	Parser strip double quote from tok
5	0	default	Doing	Wed Oct 27 10:42:58 2010	Parser enable slash double quote t
6	0	default	Finished	Wed Oct 27 10:43:28 2010	Parser enable single token to be w
7	0	default	Doing	Wed Oct 27 10:43:50 2010	Resolve memory leak and security
8	0	default	Doing	Wed Oct 27 10:44:55 2010	Enable man taskManager for linux
9	0	default	Doing	Wed Oct 27 10:45:18 2010	Fix export bug: cannot export to r
10	0	default	Doing	Wed Oct 27 10:45:33 2010	Add a input module to handle user

Figure 1: tasks exported as webpage ⁶

2.2.2 task

```
> task 1
Number: 1   Deadline: Sun Oct 31 05:11:23 2010
Priority: 0   Status: Doing
Group: default
Details:
Sample task 2 7
```

To show detail information of a task, use `task` command followed by serial number of the task.

2.2.3 pri

```
> pri 1 10
> task 1
Number: 1   Deadline: Sun Oct 31 05:11:23 2010
Priority: 10   Status: Doing
Group: default
Details:
Sample task 2
```

⁷ Adding tasks with detailed information is covered in section 2.3.1. In this example, default values are shown.

To change the priority of a task, use `pri` command followed by the serial number of a task and its new priority. Priority is typically a number between -20 and 20. By default, the priority of a newly added task is 0.

2.2.4 edit

```
> edit 1 -d "Sample task 3" -p 12 -t 1d8 -g SampleGroup -f yes
> task 1
Number:    1    Deadline:  Mon Nov 1 05:47:59 2010
Priority:  12    Status:    Finished
Group:     SampleGroup
Details:
    Sample task 3
```

To edit a task, `edit` command which has the form: `edit taskSerialNumber -d newDescription -p newPriority -t newDeadline -g newGroup -f finished_or_not`. Only `taskSerialNumber` is compulsory. Besides, to finish a task, `finish 1` is equivalent to `edit 1 -f yes`.

2.2.5 undo and redo

```
> undo
```

would undo the last command. Note that it has no effect on commands like `ls`, `export`, `tui` and `undo`.

```
> redo
```

re-does the last undo. It can be executed until all the undo's are re-done.

2.3 Using Options

Like `edit`, some of the commands come with options to support more functionality. In this section they are introduced in great detail.

2.3.1 add

`add` has three flags: `-t` to specify deadline, `-p` to set priority and `-g` to put the group name for a task.

`-t` adds a task with a deadline:

```
> add "74 hours from now" -t 3d2h
> add "Tomorrow 2359" -t b2d
> add "due at Jan 2, 1970 00:00 UTC" -t 86400
```

⁸ `-t 1d` means setting the deadline to be 1 day later. Time formats that `taskManager` accept are discussed in section 2.3.1.

TaskManager support 3 types of time format:

"plus" format "Plus" format specifies the how much time left for the task, and it has form `?w?d?h?m`⁹, where each question mark stands for a number (not a digit). For example, `3d2h` means the task will due after 3 days 2 hours the moment the command is executed – you have 3 days and 2 hours to finish it.

"by" format "By" format has a similar form of "plus" format. It is in the form of `b?w?d?h?m`¹⁰, where each question mark stands for a number (not a digit). For example:

<code>b0d22h</code>	by 10:00pm today.
<code>b2d</code>	by 23:59 tomorrow.
<code>b1w</code>	by the end of this week. i.e. 23:59 of the Saturday ¹¹ of this week.
<code>b0w5d</code>	by 23:59 on Friday of this week.
<code>b2w3d8h</code>	by 8:00am on the Wednesday of the next next week.

Unix timestamp "Unix timestamp" means the number of seconds elapsed since Jan 1, 1970 00:00:00. It is not recommended for users, but rather used as a lower-order procedure for developers.

`-p` adds a task with a priority:

```
> add "some important task" -p 20
```

`-g` specifies a group for a task:

```
> add "the task with group"12 -g SampleGroup
```

Options are not compulsory. Different options can be used together. For example:

```
> add "CS2103 final exam" -p 10 -g "finals" -t b3w2d
```

would add a task "CS2103 final exam" which has a priority of 10, belongs to group "finals" and is held on the Tuesday 3 weeks from now.

2.3.2 ls

`ls` has four flags: `-s` to sort tasks, `-k` to search tasks, `-f` to view only (un)finished tasks and `-g` to filter tasks by group name.

`-s` sorts the existing tasks:

```
> ls -s "deadline priority"
```

A more general format is: `ls -s "keyword1 keyword2" ...`

The listed tasks will be sorted by keyword1 and then keyword2 ...

Available columns are: deadline, priority and serialnumber. Prefix of a keyword is also acceptable. e.g. `ls -s "p"` will sort the tasks by their priorities.

Examples:

⁹ At least one of letters w/d/h/m should be specified.

¹⁰ At least one of letters w/d/h/m should be specified.

¹¹ Sunday is the first day of a week.

¹² If group name contains spaces, use a pair of quotation marks to quote it.

```
> ls -s "p d"
1    task 1 highest priority.    3 Sun Oct 31 06:49:09 2010
2    task 2 high priority.       2 Mon Nov 1 06:54:42 2010
3    task 3 default priority.    0 Tue Nov 2 06:54:37 2010
```

- k filters tasks with a keyword¹³ where ? means any single character, * means a string of any length (including 0 length).

Take command `ls -k *Sam?le*task` as an example, "This is a sample with an important task" will match `*Sam?le*task` as the first * matches "This is a", ? matches 'p' and the second * matches " with an important ".

"samqleTask" will also match `*sam?le*task` as both * maps empty string and ? to be 'q'.

- f shows finished/unfinished tasks:

`ls -f yes` shows only finished tasks.

`ls -f no` shows only unfinished tasks.

- g shows tasks of a specific group: `ls -g SampleTask` makes tasks only from SampleTask group shown.

Important:

As said, different options can be used together when issuing commands. When more then one restrictive options are present, *conjunction* of these restrictions are used. e.g.

```
ls -g SampleTask -f y
```

will show tasks that are finished AND from "SampleTask" group.

2.3.3 rm

`rm` has basically two usages: remove tasks by a group name or serial number.

Use `-g` option to remove a group of tasks: `rm -g SampleTask` removes the entire SampleTask group.

`rm` can be used remove several tasks once as well. e.g. `rm 1 2 3` removes tasks 1, 2 and 3.

2.4 The Text-Based Interactive User Interface

The text-based user interface (referred to as TUI later) makes interaction more intuitive by visualizing the deadlines, groups and simplifying manipulation from issuing commands to moving cursors and pressing keys.

TUI can be launched using command `tui` in command-line mode:

¹³ keyword is case insensitive.

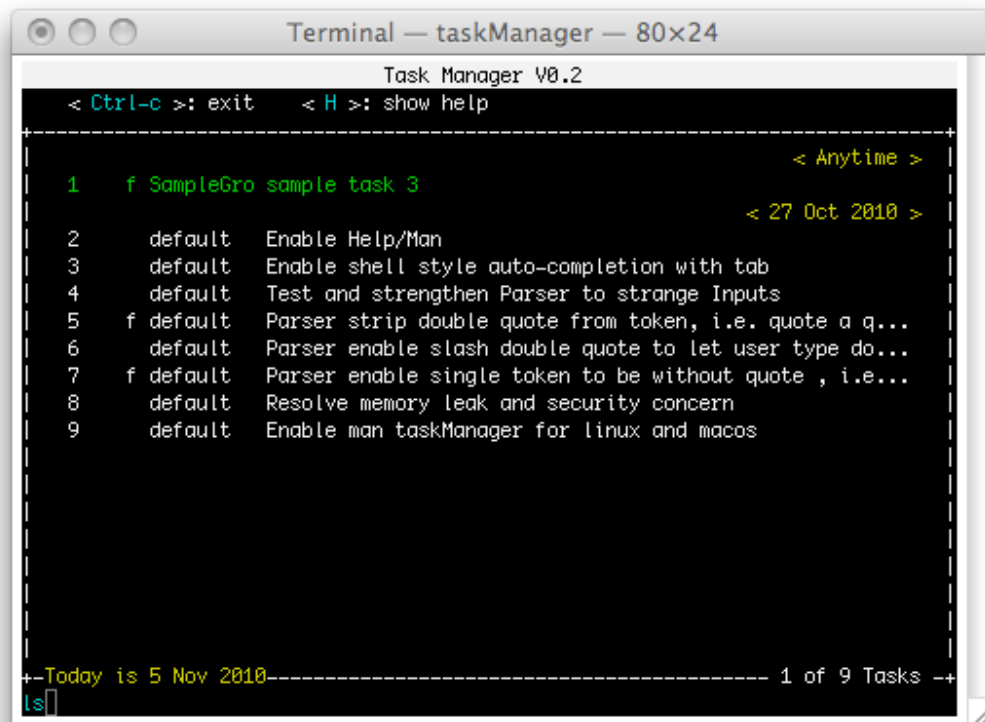


Figure 2: The first screen of TUI

As seen, the tasks are grouped according to deadlines. If a task's deadline is "Anytime", it means the task has no specific deadline – you can finish it at ease.

Besides, TUI provides some features that are not available in the command line mode. For example auto-completion using TAB key, a small calendar at the bottom right corner with the dates that has tasks due highlighted, incremental search of tasks by keyword.

This is a summary of the key bindings:

SPACE or ENTER	show details of the selected task
↓ or j	select the task below
↑ or k	select the task above
g	select the first task
G	select the last task
number selectionKey ¹⁴	select number tasks above/below
p	previous page/page up
n	next page/page down
u	undo
R	redo
c	show the small calendar
C	issue a command without leaving TUI
a	add a task
e	edit a task
d	remove a task
f	finish a task
s	search tasks by keywords

You can always press H to read the key bindings.

2.5 Advanced Usages

2.5.1 command piping

```
> ls | rm
```

TaskManager supports command piping for most commands though it is a bit different from traditional Unix pipe. Piping means if one command selects some tasks, then the selected tasks will be passed to the next command as input. The tasks after the last command will be shown as output. Piping in taskManager is done with symbol `|`. When a pipe signs appear in a command, the broken-up commands (separated by pipes) are executed one by one from left to right. For example:

1. finish all tasks:

```
> ls | finish
```

2. remove all finished tasks:

```
> ls -f yes | rm
```

3. import from a file and replace current task list:

```
> ls | rm | import newTasks.xml
```

4. import all CS2103 group tasks from a file:

```
> read newTasks.xml | ls -g CS2103 | add
```

¹⁴ It means pressing a number key and then press `↑/↓/j/k` key.

5. export all CS2103 related tasks to a html file:

```
> ls -k *CS2103* | export -html cs2103tasks.html
```

6. show details of CS2103 tasks, sort by priority:

```
> ls -g CS2103 | sort "pri" | task
```

2.5.2 command mapping

```
> map "ls" "ls -f no"
```

TaskManager supports custom command mapping/aliasing so you can alias the taskManager commands to the format you are more comfortable with.

General format of `map` is: `map "new command" "original command"`

A simple mapping is like the previous example, which maps `ls` to `ls -f no`, which effectively hides finished tasks when doing `ls`. To retain the original `ls` command, use `.`. The command that `ls` maps to will not be executed then.

When several `map` commands are executed in sequence :

```
> map "lsa" "ls"
> map "ls" "ls -f no"
```

Then `ls` will show the unfinished tasks only, while `lsa` lists all the tasks.

The order of mapping matters as commands are executed one by one. Thus reversing the order of `map` commands MIGHT NOT work.

More complex mapping makes use of symbol `$`. There are two kinds of `$` symbols:

`$0` matches all characters from the current position.

`$1`, `$2`, `$3` ... matches one token (i.e. a word or several words).

Examples:

1.

```
> map "tomorrow $1" "add $1 -t 1d"
> tomorrow "Finish user guide"
```

The latter command will be parsed as `add "Finish user guide" -t 1d`, and a new task “Finish user guide” will be added with the deadline of 24 hours.

2.

```
> map "do $1 at $2" "add $1 -t $2"
> do "Laundry" at 4h
```

The latter command will be parsed as `add "Laundry" -t 4h`, and a new task called “Laundry” will be added with the deadline of 4hours.

3.

```
> map "ls $0" "ls -f no $0" > ls > ls -g cs2103
```

The second command will be parsed as `ls -f no`, and will list out all unfinished tasks. The third command will be parsed as `ls -f no -g cs2103`, and will list out all unfinished cs2103 tasks.

Important: TUI uses `ls` to retrieve tasks. Mapping `ls` to something else will affect behaviour of TUI, which may or may not be the case you want.

2.5.3 taskManager script

Task manager commands can be saved in a text file and be executed using `run` command.

```
$ cat tmscript
ls
map "ls" "ls -f no"
ls
$ ./taskManager
Task Manager V 0.2
type " exit" to quit, "help" for more instructions
=====
> run tmscript
1 f   Sample task 1.   This also has high priority
2     Sample task 2.   This has high priority
3     Sample task 3.   This is the latest

2     Sample task 2.   This has high priority
3     Sample task 3.   This is the latest
```

The first 3 tasks are the result of the first `ls` in the script. The last 2 tasks are the result of the second `ls` in the script. Because `ls` is mapped to `ls -f no`, finished tasks are not shown by the second `ls`.

2.5.4 startup script

By default, taskManager executes a special script everytime when it is started. This script is `~/.tmrc` on *nix and `%USERPROFILE%\tmrc.txt` on Windows. This file can be edited to include customized settings.

Examples:

1. To switch to the interactive user interface by default, add this line into `tmrc`:

```
tui
```

2. To save a backup file when taskManager is started:

```
export /tmp/backupTasklist.xml
```

3. To show tasks when taskManager is started:

```
ls
```

4. To remove finished tasks when taskManager is started:

```
ls -f yes | rm
```

5. To run a script with all self-defined mappings when taskManager is started:

```
run /home/myusername/mymappings
```

2.5.5 talk to taskManager

For all inputs that cannot be recognized by taskManager as a command, it will be treated as natural language sentence. TaskManager will try its best to recognize it and give correct response. For example:

```
> what do I do today?
```

will list all the tasks due today.

2.6 Compilation and Installation

This section discusses compilation and installation of taskManager. One thing to note is that the TUI is build by default, which requires PDCurses library. It is free and can be downloaded here: <http://sourceforge.net/projects/pdcurses/files/>.

Unix-like Operating Systems are likely to have curses library, of which PDCurses is the cross-platform version, installed already. For Windows' users convenience, the necessary library files for compilation are included in the zip.

2.6.1 Microsoft Windows

On Windows, taskManager can be built with Visual Studio 2008[®] ¹⁵ in the following steps:

1. Start Visual Studio with "C++ Development Settings".
2. Create an *empty win32 console project* called "taskManager".
3. Drag all the .h and .cpp files (including those in the subdirectories) into the solution folder. Files should be automatically categorized into header files and source files.
4. Press **Alt + F7** to edit the project properties. Under "Configuration Properties" => "General", set "Character Set" to be "Use Multi-Byte Character Set"; Under "Linker" => "Input", add pdcurses.lib to "Additional Dependencies" and add the folder containing pdcurses.lib to "Additional Library Directories".
5. Copy pdcurses.dll to %WINDIR%\system32\ or the directory your executable will be generated (which would be the "Debug" folder).

6. Build the solution.
7. Copy the `tmrc` to your home directory (e.g. `C:\Documents and Settings\John Doe\` on Windows XP) and rename it to `tmrc.txt` in order to use the commands we pre-customized. This step is optional.

2.6.2 Unix-like Operating Systems

On Unix-like operating systems such as GNU Linux and Mac OS X: start a shell; unzip the zip archive; change directory properly and type:

```
$ make
$ sudo make install
```

“make install” is optional. It just makes `taskManager` available system wide by copying the executable and man page to cooresponding directories.

Another option is to copy the `tmrc` to your home directory and rename it to `.tmrc` in order to use the commands we pre-customized.

The curses library should ship with most Linux distributions and Mac OS. If not, it can be installed with the package manager (`apt-get`, `yum`, `pacman` on various Linux distributions and `port` on Mac).

¹⁵ Menu names mentioned below may vary among different versions of Visual Studio.

3 Developer Guide

4 Milestones and Individual Work

1. Decide project topic and brainstorming.

Date: Sept 17

Members attended: All

We picked task manager as the project topic. There were conflicts among opinions as some of us do not like development on Windows and thus the GUI board game, while others want to have a taste of GUI programming.

2. Design detailed architecture.

Date: Sept 17

Members attended: All

We designed the Shell-Parser-CommmandExecutor architecture. We thought it was an intuitive design and easy to split up work.

3. Integrate v0.1

Date: Oct 4 Oct 8

4. Refactor the code base for Windows Compilation (overnight session)

Date: Oct 22 - 23

Members attended: He Haocong, Liu Jialong

The purpose of this session is to extract the method declaration to .h and definition to .cpp files to make the code adhere to standard.

As we were developping on *nix, we realized that squeezing all the classes in the .h files will fail to compile on Windows using Visual Stdio. After refactoring, we are able to compile taskManager on Windows and write a better Makefile by the way – the previous Makefile contains only one rule and one command, which is something like

```
g++ -o taskManager main.cpp Shell.h Parser.h MainCommandExecutor.h ...
```

We got to know better of C++ compilation process by writing the new Makefile and solving the linking problems and it was worth the stay-up.

5. At media lab ...

Date: Oct 29

Members attended: All

6. Feature close shop, testing and documenting. (overnight session)

Date: Nov 4 - 5

Members attended: He Haocong, Liu Jialong

7. Polishing the last bits for submission.

Date: Nov 6

Members attended: All

We met up to do the developer guide, compilation on Windows, usability testing and video.