UniCa$h

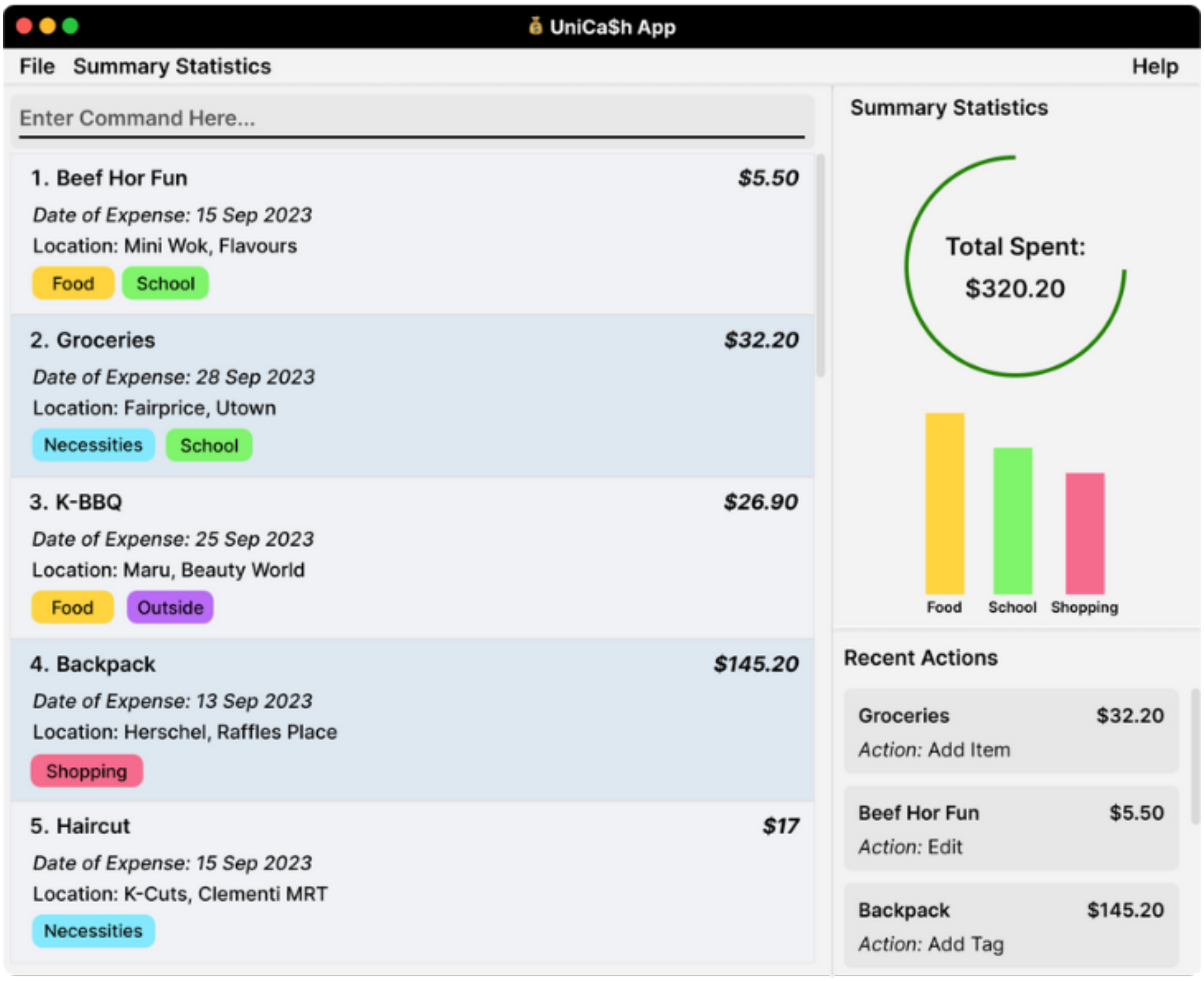
User Guide

UniCa$h is a **is a desktop application used for university students who want to be more financially conscious, optimized for use via a Command Line Interface** (CLI) while still having the benefits of a Graphical User Interface (GUI). If you can type fast, UniCa$h can get your contact management tasks done faster than traditional GUI apps.

Quick start

1. Ensure you have Java `11` or above installed in your Computer.
2. Download the latest `unicash.jar` from coming soon.
3. Copy the file to the folder you want to use as the *home folder* for your UniCa$h.
4. Open a command terminal, `cd` into the folder you put the jar file in, and use the `java -jar unicash.jar` command to run the application.

   A GUI similar to the below should appear in a few seconds. Note how the app contains some sample data.



1. Beef Hor Fun $5.50
Date of Expense: 15 Sep 2023
Location: Mini Wok, Flavours
Food School

2. Groceries $32.20
Date of Expense: 28 Sep 2023
Location: Fairprice, Utown
Necessities School

3. K-BBQ $26.90
Date of Expense: 25 Sep 2023
Location: Maru, Beauty World
Food Outside

4. Backpack $145.20
Date of Expense: 13 Sep 2023
Location: Herschel, Raffles Place
Shopping

5. Haircut $17
Date of Expense: 15 Sep 2023
Location: K-Cuts, Clementi MRT
Necessities

Summary Statistics

Total Spent: $320.20

Recent Actions

Groceries *Action:* Add Item	$32.20
Beef Hor Fun *Action:* Edit	$5.50
Backpack *Action:* Add Tag	$145.20

5. Type the command in the command box and press Enter to execute it. e.g. typing `help` and pressing Enter will open the help window.
   Some example commands you can try:
   - `commands coming soon!`
6. Refer to the Features below for details of each command.

Features

Create Expense [coming soon]

Allows a user to create an expense and all information associated with that expense including the name, amount, category (defaults to "Others"), location (optional), and date (defaults to the current date) of the expense.

Command: `create <name> -amount <expense amount> [-category <category of expense>] [-date <date of expense>] [-location <location of expense>]`

Command Argument: `name` represents the name of the expense to be added.

Command Options:

Option Name	Optional?	Purpose
-amount	No	Amount of expense. Currency is SGD.
-category	Yes	Category/type of expense, used to group and filter expenses. Defaults to "Others" if not specified.
-date	Yes	Date of when the expense was made. Follows format `dd/MM/yyyy`. Defaults to date of creation if not specified.
-location	Yes	Location where expense was made. Defaults to `NULL` if not specified.

Expected Outputs

Successful Execution

Example 1

> ***Case***: *Create expense with name, amount, date, location, and category*
>
> ***Input***: `create buy food -amount 7.50 -date 19/09/2023 -location`

`Food Clique -category Food`

Output:

```
Successfully created expense "buy food" of category "Food"!
```

Remark: *The expense will be dated 19/09/2023.*

Example 2

Case: *Create expense with name, amount, location, and category but without date*

Input: `create buy groceries -amount 14.30 -category Food -location Fairprice`

Output:

```
Successfully created expense "buy groceries" of category "Food"!
```

Remark: *The expense will be dated whenever the* `create` *command was executed.*

Example 3

Case: *Create expense with name, amount, and category but without date and location*

Input: `create buy stuff -amount 13.00 -category Misc`

Output:

```
Successfully created expense "buy stuff" of category "Misc"!
```

Remark: *The expense will be dated whenever the* `create` *command was executed and have a* `NULL` *location.*

Example 4

> ***Case***: *Create expense with name and amount but without date, location, or category*
>
> ***Input***: `create buy things –amount 10.00`
>
> ***Output***:
>
> ```
> Successfully created expense "buy things" of category "Others"!
> ```
>
> ***Remark***: *The expense will be dated whenever the* `create` *command was executed, have a* `NULL` *location, and be assigned to the "Others" category by default.*

Failed Execution

Example 1

> ***Case***: *Missing* `name` *of expense*
>
> ***Input***: `create`
>
> ***Output***:
>
> ```
> Cannot create expense without expense name. Please specify the expe
> ```

Example 2

> ***Case***: *Missing* `amount` *option of expense*
>
> ***Input***: `create buy something!`
>
> ***Output***:
>
> ```
> Cannot create expense without amount of expenditure. Please specify
> ```

Example 3

> ***Case***: *Invalid `amount` option value.*
>
> ***Input***: `create buy something! -amount hi`
>
> ***Output***:
>
> ```
> Failed to create expense as amount is invalid, ensure that it is a
> ```

Example 4

> ***Case***: *Invalid `date` option value.*
>
> ***Input***: `create buy something! -amount 14.30 -date today`
>
> ***Output***:
>
> ```
> Failed to create expense as date is invalid, ensure that it is the
> ```

Edit Expense [coming soon]

Allows a user to make edits to an existing expense, and all associated information.

Command: `edit <expense_id> -<name of attribute 1> <new attribute 1 value> [-<name of attribute N> <new attribute N value> …]`

Command Argument: `expense_id` is the ID of the expense to edit.

Command Options:

Option Name	Optional?	Purpose
		The attribute to make edits to. Possible values: `name`, `amount`, `category`, `date`, `location` Note: If `name of attribute` is date, then the associated

-name of attribute	No	`new attribute value` must be in format: `dd/MM/yyyy`. Note: If `name of attribute` is amount, then the associated `new attribute value` must be a number. Note: If `name of attribute` is not name or amount, then the associated `new attribute value` can be empty if the user wants to reset the attribute to the default value (NULL for location and Others for category).
…	Yes	More `name of attribute` `new attribute value` pairs to make more edits to the same expense

Expected Outputs

Successful Execution

Example 1

> ***Case***: *Editing one attribute of expense 3*
>
> ***Input***: `edit 3 -location online`
>
> ***Output***:
>
> ```
> Successful edits to expense 3:
> location : online
> ```

Example 2

> ***Case***: *Setting the expense's category to be default of "Others"*
>
> ***Input***: `edit 2 -category -location frontier pasta express -amount 5.8`
>
> ***Output***:
>
> ```
> Successful edits to expense 1:
> category : "Others"
> ```

```
location : frontier pasta express
amount : $5.80
```

Failed Execution

Example 1

> **Case**: No attributes to edit
>
> **Input**: `edit 1`
>
> **Output**:
>
> ```
> Please input an attribute to edit, and the new value to change the
> Syntax: edit <integer> -<name of attribute> <new attribute value>
> ```

Example 2

> **Case**: New attribute value for `name` is empty
>
> **Input**: `edit 1 -name`
>
> **Output**:
>
> ```
> Attributes “name” and “amount” cannot be empty
> ```

Example 3

> **Case**: There are only 10 expenses in the list, but user tries to edit expense 100000
>
> **Input**: `edit 100000 -location online`
>
> **Output**:
>
> ```
> There are only 10 expenses. Please provide an integer between 1 and
> ```

Example 4

> ***Case***: *Wrong input format for “date” and “amount” attribute*
>
> ***Input***: `edit 2 -date yesterday -amount 5.80.`
>
> ***Output***:
>
> ```
> Attribute “date” must be of the form dd/MM/yyyy (received: yesterda
> Attribute “amount” must be a number (received: 5.80.)
> ```

Delete Expense [coming soon]

Allows a user to delete a previously added expense and all information associated with that expense.

Command: `delete <name>`

Command Argument: `name` represents the exact name of the expense intended to be deleted. Has to exactly match a given expense, or else the command will do nothing, so as to ensure the integrity of user data.

Expected Outputs

Successful Execution

Example 1

> ***Case***: *Delete expense named “friday mcdonalds”*
>
> ***Input***: `delete "friday mcdonalds"`
>
> ***Output***:
>
> ```
> Successfully deleted expense “friday mcdonalds”!
> ```
>
> ***Remark***: *The expense will be removed from file*

Unsuccessful Execution

Example 1

> ***Case**: Delete expense command entered with no argument provided*
>
> ***Input**: `delete`*
>
> ***Output**:*
>
> ```
> No expense deleted. Delete command must be followed with an expense
> ```
>
> ***Remark**: No expenses will be removed and no changes made to file.*

Example 2

> ***Case**: Delete expense command entered with no matching expense name*
>
> ***Input**: `delete asdf`*
>
> ***Output**:*
>
> ```
> No expense deleted. Delete command must be followed with a valid ex
> ```
>
> ***Remark**: No expenses will be removed and no changes made to file.*

Mass Delete Expense [coming soon]

Allows a user to delete all added expenses, and all associated information.

Command: `delete_all_expenses`

Command Argument: No arguments are needed for this command. The command is intentionally lengthy to ensure that mass deletion of all expenses is done intentionally.

Remarks: Confirmation for mass deletion to be implemented at a later date.

Expected Outputs

Successful Execution

Example 1

> ***Case***: *Delete all expenses*
>
> ***Input***: `delete_all_expenses`
>
> ***Output***:
>
> ```
> Successfully deleted all expenses!
> ```
>
> ***Remark***: *All expenses will be removed from file*

Unsuccessful Execution

Example 1

> ***Case***: *Mass deletion command entered improperly*
>
> ***Input***: `delete_all`
>
> ***Output***:
>
> ```
> Invalid command.
> ```
>
> ***Remark***: *No expenses will be removed and no changes made to file.*

Example 2

> ***Case***: *Wrong delete command entered*
>
> ***Input***: `delete`
>
> ***Output***:

> ```
> No expense deleted. Delete command must be followed with an expense
> ```
>
> ***Remark***: *No expenses will be removed and no changes made to file. The above error is the same as the one for the simple "delete" function. In the above example, the delete_all_expenses functionality is intentionally obfuscated to prevent the user from accidental mass deletions. The rationale is that a user unsure of a basic command like delete is probably a new user, and a new user should not be directed to mass delete information. There are other, more proper ways to convey this information, such as this User Guide.*

List Expenses [coming soon]

Allows a user to retrieve a list of all their past expenses with details on where it was spent, type of spending and how much was spent.

Command: `list`

Expected Outputs

Successful Execution

Example 1

> ***Case***: *Calling the command when there are no existing expenses.*
>
> ***Input***: `list`
>
> ***Output***:
>
> ```
> You have no expenses!
> ```

Example 2

> ***Case***: *Calling the command with existing expenses.*

> ***Input***: `list`
>
> ***Output***:
>
> ```
> 1. buy groceries 23/09/23 - $15.20 (groceries)
> 2. lunch at fc 23/09/23 - $5.50 (meals)
> ```

Failed Execution

Example 1

> ***Case***: *Calling the command with any parameters*
>
> ***Input***: `list 5`
>
> ***Output***:
>
> ```
> Command not recognised. Try using the command “list” instead.
> ```

Find Expenses [coming soon]

Allows a user to retrieve the expense(s) that contain/matches any of the given keywords.

Command: `find <keyword>`

Command Parameters: `<keyword>` is the keyword to look for in any of the stored expenses, it can be a single word or multiple words separated by spaces.

Expected Outputs

Successful Execution

Example 1

> ***Case***: *Calling the command when there are no matching expenses.*

> ***Input***: `find chicken`
>
> ***Output***:
>
> ```
> You have no matching expenses!
> ```

Example 2

> ***Case***: *Calling the command with keywords that match existing expenses.*
>
> ***Input***: `find lunch`
>
> ***Output***:
>
> ```
> 2 expenses found containing the word(s) “groceries”:
>
> 1. lunch at holland 16/09/23 - $15.20 (groceries)
> 4. lunch at fc 23/09/23 - $5.50 (meals)
> ```
>
> ***Note:*** *Index of retrieved list is respective to the order of the full expense list so index of 4 is the 4th expense stored in the system.*

Failed Execution

Example 1

> ***Case***: *Calling the command without any parameters*
>
> ***Input***: `find`
>
> ***Output***:
>
> ```
> The “find” command requires at least one word to search.
> ```

Tabulate Total Expense [coming soon]

Allows a user to view their total expenditure, filtered by category of spending or by month.

Command: `total [-category <category>] [-month <month>]`

Command Options:

Option Name	Optional?	Purpose
-category	Yes	Category / type of expense. Defaults to accounting for all categories if not specified.
-month	Yes	Month of expenditure. Can either be the shorthand of the name like Sep or full name like September. Defaults to all months if not specified.

Expected Outputs

Successful Execution

Example 1

> ***Case**: Calling the command with no options.*
>
> ***Input**: `total`*
>
> ***Output**:*
>
> ```
> Your total expenditure recorded is $1388.
> ```

Example 2

> ***Case**: Calling the command with a specified category.*
>
> ***Input**: `total -category food`*
>
> ***Output**:*

> ```
> Your total expenditure recorded for food is $780.
> ```

Example 3

> ***Case***: *Calling the command with a specified month.*
>
> ***Input***: `total -month June`
>
> ***Output***:
>
> ```
> Your total expenditure recorded for June is $400.
> ```

Example 4

> ***Case***: *Calling the command with a specified category and month.*
>
> ***Input***: `total -category food -month June`
>
> ***Output***:
>
> ```
> Your total expenditure recorded for food in June is $230.
> ```

Failed Execution

Example 1

> ***Case***: *Calling the command with a category that doesn't exist.*
>
> ***Input***: `total -category chicken -month june`
>
> ***Output***:
>
> ```
> The category "chicken" doesn't exist.
> ```

Example 2

> ***Case***: *Calling the command with a month that doesn't exist.*

> ***Input***: `total -category food -month juely`
>
> ***Output***:
>
> ```
> The month "juely" doesn't exist.
> ```

Example 3

> ***Case***: *Calling the command with a category and month that doesn't exist.*
>
> ***Input***: `total -category chicken -month juely`
>
> ***Output***:
>
> ```
> The category "chicken" and month "juely" doesn't exist.
> ```

Create Income

Allows a user to register an inflow of money (income) into the application. Our application will store an income based on the name, value, date.

Command: `create_income <name> [-value <value of income>] [-date <date of expense>]`

Command Argument: `name` represents the name of the income to be added.

Command Options:

Option Name	Optional?	Purpose
-value	No	Value of expense. Represents a positive number (integer/float).
-date	Yes	Date of when the expense was made. Follows format dd/MM/yyyy

		Defaults to date of creation if not specified.

Expected Outputs

Successful Execution

Example 1

> ***Case***: *Create "work at lifo" income dated 19/09/2023 with value of 900.*
>
> ***Input***: `create_income work at liho -date 19/09/2023 -value 900`
>
> ***Output***: *Successfully created income "work at liho"!*
>
> ***Remark***: *The income will be dated 19/09/2023.*

Failed Execution

Example 1

> ***Case***: *Missing* `name` *of income*
>
> ***Input***: `create_income`
>
> ***Output***: *Cannot create income without income name. Please specify the income name as such:* `create_income <name> -value <value>`

Example 2

> ***Case***: *Missing* `value` *of income*
>
> ***Input***: `create_income working`
>
> ***Output***: *Cannot create income without income value. Please specify the income name as such:* `create_income <name> -value <value>`

Example 3

> ***Case***: *Invalid `value` form (not positive number)*
>
> ***Input***: `create_income working -value hi`
>
> ***Output***: *Cannot create income due to invalid income value type. Ensure that it is a positive number.*

Example 4

> ***Case***: *Invalid `date` of income*
>
> ***Input***: `create_income working -value 1300 -date today`
>
> ***Output***: *Cannot create income due to invalid date format. Ensure that it follows dd/MM/yyyy.*

Delete Income

Allows a user to delete an income previously added into the application.

Command: `delete_income <name>`

Expected Outputs

Successful Execution

Example 1

> ***Case***: *Delete "work at liho" income.*
>
> ***Input***: `delete_income work at liho`
>
> ***Output***: *Successfully deleted expense "work at liho"*

Failed Execution

Example 1

> ***Case***: *Missing `name` of income*
>
> ***Input***: `delete_income`
>
> ***Output***: *Cannot delete income without income `name`. Please specify the income name as such: `delete_income <name>`*

Find Income

Allows a user to search for an income(s) that was previously entered. User can find income(s) through name.

Command: `find_income [-name <name of income>] [-value_more <value of income>] [-value_less <value of income>] [-date <date of income>]`

Command Options:

Option Name	Optional?	Purpose
-name	Yes	Name of income to find.
-value_more	Yes	Value of income, used to filter income more than value.
-value_less	Yes	Value of income, used to filter income less than value.
-date	Yes	Date of when the income was made. Follows format dd/MM/yyyy. Filters income added on that date.

💡 **Note:** If no options are specified, all income is returned.

Expected Outputs

Successful Execution

Example 1

> ***Case***: *Find "work at liho" income.*
>
> ***Input***: `find_income work at liho`
>
> ***Output***: *Successfully found income "work at liho". Display information related to the income*

Failed Execution

Example 1

> ***Case***: *Missing* `name` *of income*
>
> ***Input***: `find_income`
>
> ***Output***: *Cannot find income without income name. Please specify the income name as such:* `find_income <name>`

Example 2

> ***Case***: *Invalid* `date` *format*
>
> ***Input***: `find_income work at liho -date tomorrow`
>
> ***Output***: *Cannot find income due to invalid date format. Ensure that it follows dd/MM/yyyy.*

Archiving data files `[coming in v2.0]`

Details coming soon ...

FAQ

Q: How do I transfer my data to another Computer?

A: Install the app in the other computer and overwrite the empty data file it creates with the file that contains the data of your previous UniCa$h home folder.

Known issues

1. Currently no known issues!

Command summary

Action	Format, Examples	
Create Expense	`create <name> -amount <expense amount> [-category <category of expense>] [-date <date of expense>] [-location <location of expense>]` e.g., `create buy food -amount 7.50 -date 19/09/2023 -location Food Clique -category Food`	
Delete Expense	`delete <name>` e.g., `delete grabfood_lunch`	
Mass Delete Expenses	`delete_all_expenses`	
	`edit <expense_id> -<name of attribute 1> <new`	

Edit Expenses	`attribute 1 value> [-<name of attribute N> <new attribute N value> …]` e.g., `edit 3 -location online`	
List Expenses	`list`	
Find Expenses	`find <keyword>` e.g., `find lunch`	
Tabulate Total Expense	`total [-category <category>] [-month <month>]` e.g., `total -category Food -month June`	
Create Income	`create_income <name> [-value <value of income>] [-date <date of expense>]` e.g., `create_income work at liho -date 19/09/2023 -value 900`	
Delete Income	`delete_income <name>`	
Find Income	`find_income <name> [-value_more <value of income>] [-value_less <value of income>] [-date <date of income>]` e.g., `find_income work at liho`	