Sui Client PTB CLI

The client ptb command allows you to specify the transactions for execution in a programmable transaction block (PTB) directly from your CLI or through bash scripts.

The examples in this document were tested using a bash shell environment. Your experience might vary depending on how your shell interprets the input values (for example, zsh requires quotes around passed values in brackets: "[]"; whereas bash accepts them without quotes). On Windows, you might need to add even more quotes around arguments passed (for example, --assign "forge @").

The following list itemizes all the available args for the sui client ptb command. Use the --help for a long help version that includes some examples on how to use this command.

The main philosophy behind the CLI PTB support is to enable a user to build and execute a PTB from the command line. Bash scripts can be used to construct and execute the PTB just as you would do from the command line, providing great flexibility when it comes to automating different tasks.

Besides using existing traditional PTB related concepts, we introduce a few new and important concepts for this command.

All the following examples were tested using a bash shell environment and your experience may vary depending on how your shell interprets the input values (e.g., zsh requires to pass values in brackets by adding quotes around it: "[]"; bash accepts them without quotes).

Sometimes, CLI PTBs require that you specify the type of a value or variable. For instance, in the following example you must provide the type when calling the 0x1::option::is_none function.

To pass in multiple types, delimit them with a comma:

CLI PTBs support string literals as inputs, which will be encoded as pure values that can be used as inputs to vector , std::ascii::String and std::string::String parameters. The following example previews a transaction block that passes the string "Hello, world" to a function m::f in a package $PKG (its ID is held in an environment variable).

Double-quoted string literals tend to also be valid syntax for shells (like bash), so when inputting PTBs on the command-line, remember to wrap the entire string in single-quotes so that its double-quotes are interpreted literally, as in the previous example.

You can pass literal addresses and objects IDs by prefixing them with '@'. This is needed to distinguish a hexadecimal value from an address in some situations.

For addresses that are in your local wallet, you can use their alias instead (passing them without '@', for example, --transfer-objects my_alias).

Here are some examples for transfer-objects and gas-coin :

Use the --assign argument to bind values to variables. There are two ways you can use it:

Let's look at the first case where you assign a value to a variable. You want to check if some variable's value is none . Call the 0x1::option::is_none function from the Move standard library, and pass in the variable name:

CLI PTB uses name resolution for common packages like sui , std , deepbook , so you can use them directly instead of their addresses: 0x2 , 0x1 , or 0xdee9 .

In the second case, if a previous command outputs some result, you can bound it to a variable for later access. Let's see an example where you want a new coin with 1000 MIST, which you can achieve by using the split-coins command. After you do that, you want to transfer the new coin to another address. Without the --assign argument, you couldn't instruct the CLI to transfer that new coin object as you would not have a way to refer to it.

If you build a complex PTB, use the --preview flag to display the PTB transaction list instead of executing it.

The following examples demonstrate how to use the client ptb command.

When a PTB is executed, the output contains all the relevant information (transaction data, gas cost, effects, object changes, and so on). Use --summary to get a short summary when you do not need all the data. For complex PTBs, you can use --preview to display the PTB transaction list instead of executing it.

When needing to execute a Move call, use the --move-call transaction to call a specific function from a package. The CLI PTB supports name resolution for common packages like sui , std , deepbook , so you can use both 0x1::option::is_none as well as std::option::is_none for passing the function name.

To call a specific function from a specific package, you can use the following call:

Publishing a package is one of the most important commands you need when working with Sui. While the CLI has a standalone publish command, PTBs also support publishing and upgrading packages. One main difference is that with sui client ptb , you must explicitly transfer the UpgradeCap object that is returned when creating a package, or destroy it with a call to make_immutable . Here is an example on how to publish a Move project on chain using the sui client ptb command. It makes a call to the sui::tx_context::sender to acquire the sender and assigns the result of that call to the sender variable, and then calls the publish command. The result of publish is bounded to upgrade_cap variable, and then this object is transferred to the sender.

The following example showcases how to split a gas coin into multiple coins, make a move call to destroy one or more of the new coins, and finally merge the coins that were not destroyed back into the gas coin. It also showcases how to use framework name resolution (for example, sui::coin instead of 0x2::coin) and how to refer to different values in an array using the . syntax.

This example creates three new coins from gas and transfers them to a different address.

You can also pass an alias (without the '@') instead of an address.

You cannot use the following words for variable names:

Append the --json flag to commands to format responses in JSON instead of the more human-friendly default Sui CLI output. This can be useful for extremely large datasets, for example, as those results can have a troublesome display on smaller screens. In these cases, the --json flag is useful.

Commands

The following list itemizes all the available args for the sui client ptb command. Use the --help for a long help version that includes some examples on how to use this command.

The main philosophy behind the CLI PTB support is to enable a user to build and execute a PTB from the command line. Bash scripts can be used to construct and execute the PTB just as you would do from the command line, providing great flexibility when it comes to automating different tasks.

Besides using existing traditional PTB related concepts, we introduce a few new and important concepts for this command.

All the following examples were tested using a bash shell environment and your experience may vary depending on how your shell interprets the input values (e.g., zsh requires to pass values in brackets by adding quotes around it: "[]"; bash accepts them without quotes).

Sometimes, CLI PTBs require that you specify the type of a value or variable. For instance, in the following example you must provide the type when calling the 0x1::option::is_none function.

To pass in multiple types, delimit them with a comma:

CLI PTBs support string literals as inputs, which will be encoded as pure values that can be used as inputs to vector , std::ascii::String and std::string::String parameters. The following example previews a transaction block that passes the string "Hello, world" to a function m::f in a package $PKG (its ID is held in an environment variable).

Double-quoted string literals tend to also be valid syntax for shells (like bash), so when inputting PTBs on the command-line, remember to wrap the entire string in single-quotes so that its double-quotes are interpreted literally, as in the previous example.

You can pass literal addresses and objects IDs by prefixing them with '@'. This is needed to distinguish a hexadecimal value from an address in some situations.

For addresses that are in your local wallet, you can use their alias instead (passing them without '@', for example, --transfer-objects my_alias).

Here are some examples for transfer-objects and gas-coin :

Use the --assign argument to bind values to variables. There are two ways you can use it:

Let's look at the first case where you assign a value to a variable. You want to check if some variable's value is none . Call the 0x1::option::is_none function from the Move standard library, and pass in the variable name:

CLI PTB uses name resolution for common packages like sui , std , deepbook , so you can use them directly instead of their addresses: 0x2 , 0x1 , or 0xdee9 .

In the second case, if a previous command outputs some result, you can bound it to a variable for later access. Let's see an example where you want a new coin with 1000 MIST, which you can achieve by using the split-coins command. After you do that, you want to transfer the new coin to another address. Without the --assign argument, you couldn't instruct the CLI to transfer that new coin object as you would not have a way to refer to it.

If you build a complex PTB, use the --preview flag to display the PTB transaction list instead of executing it.

The following examples demonstrate how to use the client ptb command.

When a PTB is executed, the output contains all the relevant information (transaction data, gas cost, effects, object changes, and so on). Use --summary to get a short summary when you do not need all the data. For complex PTBs, you can use --preview to display the PTB transaction list instead of executing it.

When needing to execute a Move call, use the --move-call transaction to call a specific function from a package. The CLI PTB supports name resolution for common packages like sui , std , deepbook , so you can use both 0x1::option::is_none as well as std::option::is_none for passing the function name.

To call a specific function from a specific package, you can use the following call:

Publishing a package is one of the most important commands you need when working with Sui. While the CLI has a standalone publish command, PTBs also support publishing and upgrading packages. One main difference is that with sui client ptb , you must explicitly transfer the UpgradeCap object that is returned when creating a package, or destroy it with a call to make_immutable . Here is an example on how to publish a Move project on chain using the sui client ptb command. It makes a call to the sui::tx_context::sender to acquire the sender and assigns the result of that call to the sender variable, and then calls the publish command. The result of publish is bounded to upgrade_cap variable, and then this object is transferred to the sender.

The following example showcases how to split a gas coin into multiple coins, make a move call to destroy one or more of the new coins, and finally merge the coins that were not destroyed back into the gas coin. It also showcases how to use framework name resolution (for example, sui::coin instead of 0x2::coin) and how to refer to different values in an array using the . syntax.

This example creates three new coins from gas and transfers them to a different address.

You can also pass an alias (without the '@') instead of an address.

You cannot use the following words for variable names:

Append the --json flag to commands to format responses in JSON instead of the more human-friendly default Sui CLI output. This can be useful for extremely large datasets, for example, as those results can have a troublesome display on smaller screens. In these cases, the --json flag is useful.

Design philosophy and concepts

The main philosophy behind the CLI PTB support is to enable a user to build and execute a PTB from the command line. Bash scripts can be used to construct and execute the PTB just as you would do from the command line, providing great flexibility when it comes to automating different tasks.

Besides using existing traditional PTB related concepts, we introduce a few new and important concepts for this command.

All the following examples were tested using a bash shell environment and your experience may vary depending on how your shell interprets the input values (e.g., zsh requires to pass values in brackets by adding quotes around it: "[]"; bash accepts them without quotes).

Sometimes, CLI PTBs require that you specify the type of a value or variable. For instance, in the following example you must provide the type when calling the 0x1::option::is_none function.

To pass in multiple types, delimit them with a comma:

CLI PTBs support string literals as inputs, which will be encoded as pure values that can be used as inputs to vector ,

std::ascii::String and std::string::String parameters. The following example previews a transaction block that passes the string "Hello, world" to a function m::f in a package $PKG (its ID is held in an environment variable).

Double-quoted string literals tend to also be valid syntax for shells (like bash), so when inputting PTBs on the command-line, remember to wrap the entire string in single-quotes so that its double-quotes are interpreted literally, as in the previous example.

You can pass literal addresses and objects IDs by prefixing them with '@'. This is needed to distinguish a hexadecimal value from an address in some situations.

For addresses that are in your local wallet, you can use their alias instead (passing them without '@', for example, --transfer-objects my_alias).

Here are some examples for transfer-objects and gas-coin :

Use the --assign argument to bind values to variables. There are two ways you can use it:

Let's look at the first case where you assign a value to a variable. You want to check if some variable's value is none . Call the 0x1::option::is_none function from the Move standard library, and pass in the variable name:

CLI PTB uses name resolution for common packages like sui , std , deepbook , so you can use them directly instead of their addresses: 0x2 , 0x1 , or 0xdee9 .

In the second case, if a previous command outputs some result, you can bound it to a variable for later access. Let's see an example where you want a new coin with 1000 MIST, which you can achieve by using the split-coins command. After you do that, you want to transfer the new coin to another address. Without the --assign argument, you couldn't instruct the CLI to transfer that new coin object as you would not have a way to refer to it.

If you build a complex PTB, use the --preview flag to display the PTB transaction list instead of executing it.

The following examples demonstrate how to use the client ptb command.

When a PTB is executed, the output contains all the relevant information (transaction data, gas cost, effects, object changes, and so on). Use --summary to get a short summary when you do not need all the data. For complex PTBs, you can use --preview to display the PTB transaction list instead of executing it.

When needing to execute a Move call, use the --move-call transaction to call a specific function from a package. The CLI PTB supports name resolution for common packages like sui , std , deepbook , so you can use both 0x1::option::is_none as well as std::option::is_none for passing the function name.

To call a specific function from a specific package, you can use the following call:

Publishing a package is one of the most important commands you need when working with Sui. While the CLI has a standalone publish command, PTBs also support publishing and upgrading packages. One main difference is that with sui client ptb , you must explicitly transfer the UpgradeCap object that is returned when creating a package, or destroy it with a call to make_immutable . Here is an example on how to publish a Move project on chain using the sui client ptb command. It makes a call to the sui::tx_context::sender to acquire the sender and assigns the result of that call to the sender variable, and then calls the publish command. The result of publish is bounded to upgrade_cap variable, and then this object is transferred to the sender.

The following example showcases how to split a gas coin into multiple coins, make a move call to destroy one or more of the new coins, and finally merge the coins that were not destroyed back into the gas coin. It also showcases how to use framework name resolution (for example, sui::coin instead of 0x2::coin) and how to refer to different values in an array using the . syntax.

This example creates three new coins from gas and transfers them to a different address.

You can also pass an alias (without the '@') instead of an address.

You cannot use the following words for variable names:

Append the --json flag to commands to format responses in JSON instead of the more human-friendly default Sui CLI output. This can be useful for extremely large datasets, for example, as those results can have a troublesome display on smaller screens. In these cases, the --json flag is useful.

Examples

The following examples demonstrate how to use the client ptb command.

When a PTB is executed, the output contains all the relevant information (transaction data, gas cost, effects, object changes, and so on). Use --summary to get a short summary when you do not need all the data. For complex PTBs, you can use --preview to display the PTB transaction list instead of executing it.

When needing to execute a Move call, use the --move-call transaction to call a specific function from a package. The CLI PTB supports name resolution for common packages like sui , std , deepbook , so you can use both 0x1::option::is_none as well as std::option::is_none for passing the function name.

To call a specific function from a specific package, you can use the following call:

Publishing a package is one of the most important commands you need when working with Sui. While the CLI has a standalone publish command, PTBs also support publishing and upgrading packages. One main difference is that with sui client ptb , you must explicitly transfer the UpgradeCap object that is returned when creating a package, or destroy it with a call to make_immutable . Here is an example on how to publish a Move project on chain using the sui client ptb command. It makes a call to the sui::tx_context::sender to acquire the sender and assigns the result of that call to the sender variable, and then calls the publish command. The result of publish is bounded to upgrade_cap variable, and then this object is transferred to the sender.

The following example showcases how to split a gas coin into multiple coins, make a move call to destroy one or more of the new coins, and finally merge the coins that were not destroyed back into the gas coin. It also showcases how to use framework name resolution (for example, sui::coin instead of 0x2::coin) and how to refer to different values in an array using the . syntax.

This example creates three new coins from gas and transfers them to a different address.

You can also pass an alias (without the '@') instead of an address.

You cannot use the following words for variable names:

Append the --json flag to commands to format responses in JSON instead of the more human-friendly default Sui CLI output. This can be useful for extremely large datasets, for example, as those results can have a troublesome display on smaller screens. In these cases, the --json flag is useful.

Reserved words

You cannot use the following words for variable names:

Append the --json flag to commands to format responses in JSON instead of the more human-friendly default Sui CLI output. This can be useful for extremely large datasets, for example, as those results can have a troublesome display on smaller screens. In these cases, the --json flag is useful.

JSON output

Append the --json flag to commands to format responses in JSON instead of the more human-friendly default Sui CLI output. This can be useful for extremely large datasets, for example, as those results can have a troublesome display on smaller screens. In these cases, the --json flag is useful.