Bitburner Documentation

Release 0.47.0

Bitburner

Jul 16, 2019

Contents:

Bitburner is a programming-based incremental game that revolves around hacking and cyberpunk themes. The game is currently in the early beta stage of development. It can be played here.

CHAPTER 1

What is Bitburner?

Bitburner is a cyberpunk-themed incremental RPG where you, the player, take the role of an unknown hacker in a dark, dystopian world. When a mysterious hacker called jump3R messages you, he/she confirms your suspicions that there is something wrong with the world around you. Now, aided by jump3R, you embark on a quest to gain money and power by any means necessary, in the hopes that this will lead to to uncover the secrets that you've been searching for.

1.1 Netscript

Netscript is the programming language used in the world of Bitburner.

When you write scripts in Bitburner, they are written in the Netscript language. Netscript is simply a subset of JavaScript. This means that Netscript's syntax is identical to that of JavaScript, but it does not implement some of the features that JavaScript has.

If you have any requests or suggestions to improve the Netscript language, feel free to reach out to the developer!

1.1.1 Learn to Program in Netscript

Netscript is simply a subset of JavaScript, with some additional functions added in to allow interaction with the game.

1.1.1.1 For Beginner Programmers

If you have little to no programming experience, that's okay! You don't need to be a great programmer in order to enjoy or play this game. In fact, this game could help you learn some basic programming concepts.

Here are some good tutorials for learning programming/JavaScript as a beginner:

- Learn-JS
- **Speaking JavaScript** This is a bit on the longer side. You can skip all of the historical background stuff. Recommended chapters: 1, 7-18

1.1.1.2 For Experienced Programmers

The following section lists several good tutorials/resources for those who have experience programming but who have not worked extensively with JavaScript before.

Before that, however, it's important to clarify some terminology about the different versions of JavaScript. These are summarized in this article:

WTF is ES6, ES8, ES2017, ECMAScript...

An important takeaway from this article is that ES6, also known as ES2015, introduced many major features that are commonly seen in modern JavaScript programming. However, this means that ES5 engines and interpreters will fail if they encounters these ES6 features. You'll see why this is important further down.

- MDN Introduction to JS
- **Eloquent JavaScript (ES6+)** Recommended Chapters: Introduction, 1-6
- **Modern Javascript Tutorial (ES6+)** Recommended Chapters: 2, 4-6

1.1.1.3 Netscript 1.0 vs Netscript 2.0

There are two versions of Netscript:

- *Netscript 1.0*
- *NetscriptJS (Netscript 2.0)*

Visit the pages above to get more details about each version. If you are new to programming or unfamiliar with JavaScript, I would recommend starting out with *Netscript 1.0*. Experienced web developers can use *NetscriptJS (Netscript 2.0)* to take advantage of faster speeds and additional features.

Here is a short summary of the differences between Netscript 1.0 and Netscript 2.0:

Netscript 1.0

- ES5
- Some ES6 features implemented with polyfills
- Slow compared to NetscriptJS (interpreter runs at the "Netscript Exec Time" speed configured in options)
- Compatible with all browsers

Netscript JS (Netscript 2.0)

- Supports (almost) all features of modern JavaScript
- Extremely fast - code is executed as an Async Function
- Currently only works with Google Chrome browser
- Each script becomes a module and therefore all instances of that script can easily share data between each other (essentially global/static variables)

1.1.2 Netscript 1.0

Netscript 1.0 is implemented using a modified version of Neil Fraser's JS-Interpreter.

This is an ES5 JavaScript interpreter. This means that (almost) any JavaScript feature that is available in ES5 is also available in Netscript 1.0. However, this also means that the interpreter does not natively support any JavaScript features introduced in versions ES6 or after.

If you are confused by the ES5/ES6/etc. terminology, consider reading this: WTF is ES6, ES8, ES2017, ECMAScript...

Netscript 1.0 scripts end with the ".script" extension in their filenames.

1.1.2.1 Which ES6+ features are supported?

Netscript 1.0 is a ES5 interpreter, but the following features from versions ES6 and above are supported as well.

If there is an additional ES6+ feature you would like to see implemented with a polyfill, feel free to open an issue (and provide the polyfill if possible).

- import - See *Importing Functions*
- **Array**
  - find()
  - findIndex()
  - includes()
- **String**
  - endsWith()
  - includes()
  - startsWith()

1.1.3 NetscriptJS (Netscript 2.0)

Netscript 2.0, or Netscript JS, is the new and improved version of Netscript that allows users to write (almost) full-fledged Javascript code in their scripts, while still being able to access the Netscript functions.

NetscriptJS was developed primarily by Github user jaguilar

On top of having almost all of the features and capabilities of JavaScript, NetscriptJS is also significantly faster than Netscript 1.0.

This documentation will not go over any of the additional features of NetscriptJS, since there is plenty of documentation on Javascript available on the web.

1.1.3.1 Browser compatibility

As of the time of writing this, a few browsers do not support dynamic import functionality and therefore cannot run NetscriptJS scripts. These browsers will thus only be capable of using Netscript 1.0.

1.1.3.2 How to use NetscriptJS

Working with NetscriptJS scripts is the same as Netscript 1.0 scripts. The only difference is that NetscriptJS scripts use the ".ns" or ".js" extension rather than ".script". E.g.:

```
$ nano foo.ns
$ run foo.ns -t 100 arg1 arg2 arg3
exec("foo.ns", "purchasedServer1", "100", "randomArg");
```

The caveat when using NetscriptJS to write scripts is that your code must be asynchronous. Furthermore, instead of using the global scope and executing your code sequentially, NetscriptJS uses a `main()` function as an entry point.

Furthermore, the "Netscript environment" must be passed into a NetscriptJS script through the main function. This environment includes all of the pre-defined Netscript functions (`hack()`, `exec`, etc.) as well as the arguments you pass to the script.

Therefore, the signature of the `main()` function must be:

```
export async function main(ns) {
    ns.print("Starting script here");
    await ns.hack("foodnstuff"); //Use Netscript hack function
    ns.print(ns.args);           //The script arguments must be prefaced with ns as␣
↪well
}
```

Here is a summary of all rules you need to follow when writing Netscript JS code:

- Write `await` before any call to the following Netscript functions:
  - hack
  - grow
  - weaken
  - sleep
  - prompt
  - wget
- Any function that contains `await` must be declared as `async`
- Always `await` any function that is marked as `async`
- Any functions that you want to be visible from other scripts must be marked with `export`.
- **Do not write any infinite loops without using a `sleep` or one of the timed Netscript functions like `hack`.** Doing so will crash your game.
- Any global variable declared in a NetscriptJS script is shared between all instances of that script. For example, assume you write a script *foo.ns* and declared a global variable like so:

  ```
  //foo.ns
  let globalVariable;

  export async function main(ns) {
      globalVariable = ns.args.length;
      while(true) {
          ns.tprint(globalVariable);
          await ns.sleep(3000);
      }
  }
  ```

  Then, you ran multiple instances of *foo.ns*:

  ```
  $ run foo.ns 1
  $ run foo.ns 1 2 3
  $ run foo.ns 1 2 3 4 5
  ```

Then all three instances of foo.ns will share the same instance of `globalVariable`. (In this example, the value of `globalVariable` will be set to 5 because the last instance of *foo.ns* to run has 5 arguments. This means that all three instances of the script will repeatedly print the value 5).

These global variables can be thought of as C++ static class members, where a NetscriptJS script is a class and a global variable is a static member within that class.

1.1.3.3 Warnings

The NetscriptJS evaluation engine works by converting your code into a blob URL and then using a dynamic import to load your code as a module. Every unique NetscriptJS script is loaded as its own module. This means that making a small edit to a NetscriptJS script results in a new module being generated.

At this point, we have been unable to find a method for deleting modules from browsers so that they get garbage collected.

The result is that these modules from NetscriptJS scripts accumulate in your browser, using memory that never gets released. Over time, this results in a memory-leak type situation that can slow down your computer.

Therefore, there are two recommendations for those who decide to use NetscriptJS:

1. Every now and then, close and re-open the game. This will clear all of the modules. To be safe, I recommend **completely** closing the game's tab and then re-opening it. Depending on your browser, a refresh or page reload does not always clear the modules.

2. Only use NetscriptJS scripts when needed. It is very unlikely that NetscriptJS is needed for very simple scripts. By doing this, you will reduce the number of modules that are loaded.

1.1.3.4 Examples

DOM Manipulation (tprintColored.ns)

Directly alter the game's terminal and print colored text:

```
export function tprintColored(txt, color) {
    let terminalInput   = document.getElementById("terminal-input");
    let rowElement      = document.createElement("tr");
    let cellElement     = document.createElement("td");

    rowElement.classList.add("posted");
    cellElement.classList.add("terminal-line");
    cellElement.style.color = color;
    cellElement.innerText = txt;

    rowElement.appendChild(cellElement);
    terminalInput.before(rowElement);
}

export async function main(ns) {
    tprintColored("Red Text!", "red");
    tprintColored("Blue Text!", "blue");
    tprintColored("Use Hex Codes!", "#3087E3");
}
```

Script Scheduler (scriptScheduler.ns)

This script shows some of the new functionality that is available in NetscriptJS, including objects and object constructors, changing an object's prototype, and importing other NetscriptJS scripts:

```
import {tprintColored} from "tprintColored.ns"; //Importing from other NetscriptJS␣
↪scripts works!

function ScriptJob(params) {
    if (params.fn == null) {
        throw new Error("No Filename (fn) passed into ScriptJob ctor");
    }

    this.fn         = params.fn;
    this.threads    = params.threads ? params.threads : 1;
    this.args       = params.args    ? params.args : [];
}

ScriptJob.prototype.run = async function(ns) {
    let runArgs = [this.fn, this.threads].concat(this.args);
    await ns.run.apply(this, runArgs);
    tprintColored("Running " + this.fn + " on " + ns.getHostname(), "blue");
}

ScriptJob.prototype.exec = async function(ns, target) {
    ns.scp(this.fn, target);

    let execArgs = [this.fn, target, this.threads].concat(this.args);
    await ns.exec.apply(this, execArgs);

    tprintColored("Executing " + this.fn + " on " + target, "blue");
}

export async function main(ns) {
    tprintColored("Starting scriptScheduler.ns", "red");
    try {
        let job = new ScriptJob({
            fn:         "test.js",
            threads:    1,
            args:       ["foodnstuff"]
        });
        await job.run(ns);
        await job.exec(ns, "foodnstuff");
    } catch (e) {
        ns.tprint("Exception thrown in scriptScheduler.ns: " + e);
    }
}
```

1.1.3.5 Final Note

NetscriptJS opens up a lot of possibilities when scripting. I look forward to seeing the scripts that people come up with. Just remember that the power and capabilities of NetscriptJS come with risks. Please backup your save if you're going to experiment with NetscriptJS and report any serious exploits.

With great power comes great responsibility

Happy hacking

1.1.4 Netscript Script Arguments

Arguments passed into a script can be accessed in Netscript using a special array called *args*. The arguments can be accessed using a normal array using the [] operator (args[0], args[1], etc...).

For example, let's say we want to make a generic script 'generic-run.script' and we plan to pass two arguments into that script. The first argument will be the name of another script, and the second argument will be a number. This generic script will run the script specified in the first argument with the amount of threads specified in the second element. The code would look like:

```
run(args[0], args[1]);
```

It is also possible to get the number of arguments that was passed into a script using:

```
args.length
```

WARNING: Do not try to modify the args array. This will break the game. I will do my best to prevent players from doing this.

1.1.5 Netscript Basic Functions

This page contains the complete documentation for all functions that are available in Netscript. This includes information such as function signatures, what they do, and their return values.

1.1.5.1 hack() Netscript Function

hack (*hostname/ip*[, *opts={}*])

Arguments

- **hostname/ip** (*string*) – IP or hostname of the target server to hack
- **opts** (*object*) – Optional parameters for configuring function behavior. Properties:
  - threads (*number*) - Number of threads to use for this function. Must be less than or equal to the number of threads the script is running with.
  - stock (*boolean*) - If true, the function can affect the stock market. See *Player Actions Influencing Stocks*

Returns The amount of money stolen if the hack is successful, and zero otherwise

RAM cost 0.1 GB

Function that is used to try and hack servers to steal money and gain hacking experience. The runtime for this command depends on your hacking level and the target server's security level. In order to hack a server you must first gain root access to that server and also have the required hacking level.

A script can hack a server from anywhere. It does not need to be running on the same server to hack that server. For example, you can create a script that hacks the 'foodnstuff' server and run that script on any server in the game.

A successful hack() on a server will raise that server's security level by 0.002.

Example:

```
hack("foodnstuff");
hack("10.1.2.3");
hack("foodnstuff", { threads: 5 }); // Only use 5 threads to hack
```

1.1.5.2 grow() Netscript Function

grow(*hostname/ip*[, *opts={}*])

Arguments

- **hostname/ip** (*string*) – IP or hostname of the target server to grow
- **opts** (*object*) – Optional parameters for configuring function behavior. Properties:
  - threads (*number*) - Number of threads to use for this function. Must be less than or equal to the number of threads the script is running with.
  - stock (*boolean*) - If true, the function can affect the stock market. See *Player Actions Influencing Stocks*

Returns The number by which the money on the server was multiplied for the growth

RAM cost 0.15 GB

Use your hacking skills to increase the amount of money available on a server. The runtime for this command depends on your hacking level and the target server's security level. When grow() completes, the money available on a target server will be increased by a certain, fixed percentage. This percentage is determined by the target server's growth rate (which varies between servers) and security level. Generally, higher-level servers have higher growth rates. The getServerGrowth() function can be used to obtain a server's growth rate.

Like hack(), grow() can be called on any server, regardless of where the script is running. The grow() command requires root access to the target server, but there is no required hacking level to run the command. It also raises the security level of the target server by 0.004.

Example:

```
grow("foodnstuff");
grow("foodnstuff", { threads: 5 }); // Only use 5 threads to grow
```

1.1.5.3 weaken() Netscript Function

weaken(*hostname/ip*[, *opts={}*])

Arguments

- **hostname/ip** (*string*) – IP or hostname of the target server to weaken
- **opts** (*object*) – Optional parameters for configuring function behavior. Properties:
  - threads (*number*) - Number of threads to use for this function. Must be less than or equal to the number of threads the script is running with.

Returns The amount by which the target server's security level was decreased. This is equivalent to 0.05 multiplied by the number of script threads

RAM cost 0.15 GB

Use your hacking skills to attack a server's security, lowering the server's security level. The runtime for this command depends on your hacking level and the target server's security level. This function lowers the security level of the target server by 0.05.

Like hack() and grow(), weaken() can be called on any server, regardless of where the script is running. This command requires root access to the target server, but there is no required hacking level to run the command.

Example:

```
weaken("foodnstuff");
weaken("foodnstuff", { threads: 5 }); // Only use 5 threads to weaken
```

1.1.5.4 hackAnalyzeThreads() Netscript Function

hackAnalyzeThreads(*hostname/ip*, *hackAmount*)

Arguments

- **hostname/ip** (*string*) – IP or hostname of server to analyze
- **hackAmount** (*number*) – Amount of money you want to hack from the server

Returns The number of threads needed to hack() the server for *hackAmount* money

RAM cost 1 GB

This function returns the number of script threads you need when running the *hack()* command to steal the specified amount of money from the target server.

If *hackAmount* is less than zero or greater than the amount of money available on the server, then this function returns -1.

For example, let's say the *foodnstuff* server has $10m and you run:

```
hackAnalyzeThreads("foodnstuff", 1e6);
```

If this function returns 50, this means that if your next *hack()* call is run on a script with 50 threads, it will steal $1m from the *foodnstuff* server.

Warning: The value returned by this function isn't necessarily a whole number.

Warning: It is possible for this function to return `Infinity` or `NaN` in certain uncommon scenarios. This is because in JavaScript:

- `0 / 0 = NaN`
- `N / 0 = Infinity` for 0 < N < Infinity.

For example, if a server has no money available and you want to hack some positive amount from it, then the function would return `Infinity` because this would be impossible.

1.1.5.5 hackAnalyzePercent() Netscript Function

hackAnalyzePercent(*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – IP or hostname of target server

Returns The percentage of money you will steal from the target server with a single hack

RAM cost 1 GB

Returns the percentage of the specified server's money you will steal with a single hack. This value is returned in **percentage form, not decimal (Netscript functions typically return in decimal form, but not this one).**

For example, assume the following returns 1:

```
hackAnalyzePercent("foodnstuff");
```

This means that if hack the *foodnstuff* server, then you will steal 1% of its total money. If you *hack()* using N threads, then you will steal N% of its total money.

1.1.5.6 hackChance() Netscript Function

hackChance(*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – IP or hostname of target server

Returns The chance you have of successfully hacking the target server

RAM cost 1 GB

Returns the chance you have of successfully hacking the specified server. This returned value is in decimal form, not percentage.

1.1.5.7 growthAnalyze() Netscript Function

growthAnalyze(*hostname/ip*, *growthAmount*)

Arguments

- **hostname/ip** (*string*) – IP or hostname of server to analyze
- **growthAmount** (*number*) – Multiplicative factor by which the server is grown. Decimal form. Must be >= 1.

Returns The amount of grow() calls needed to grow the specified server by the specified amount

RAM cost 1 GB

This function returns the number of "growths" needed in order to increase the amount of money available on the specified server by the specified amount.

The specified amount is multiplicative and is in decimal form, not percentage.

For example, if you want to determine how many *grow()* calls you need to double the amount of money on *foodnstuff*, you would use:

```
growthAnalyze("foodnstuff", 2);
```

If this returns 100, then this means you need to call *grow()* 100 times in order to double the money (or once with 100 threads).

Warning: The value returned by this function isn't necessarily a whole number.

1.1.5.8 sleep() Netscript Function

sleep(*n*)

Arguments

- **n** (*number*) – Number of milliseconds to sleep

RAM cost 0 GB

Suspends the script for n milliseconds.

1.1.5.9 print() Netscript Function

print(*x*)

Arguments

- **x** – Value to be printed

RAM cost 0 GB

Prints a value or a variable to the script's logs.

1.1.5.10 tprint() Netscript Function

tprint(*x*)

Arguments

- **x** – Value to be printed

RAM cost 0 GB

Prints a value or a variable to the Terminal

1.1.5.11 clearLog() Netscript Function

clearLog()

RAM cost 0 GB

Clears the script's logs

1.1.5.12 disableLog() Netscript Function

disableLog(*fn*)

Arguments

- **fn** (*string*) – Name of function for which to disable logging

RAM cost 0 GB

Disables logging for the given function. Logging can be disabled for all functions by passing 'ALL' as the argument.

Note that this does not completely remove all logging functionality. This only stops a function from logging when the function is successful. If the function fails, it will still log the reason for failure.

Notable functions that cannot have their logs disabled: run, exec, exit

1.1.5.13 enableLog() Netscript Function

enableLog(*fn*)

Arguments

- **fn** (*string*) – Name of function for which to enable logging

RAM cost 0 GB

Re-enables logging for the given function. If 'ALL' is passed into this function as an argument, then it will revert the effects of disableLog('ALL')

1.1.5.14 isLogEnabled() Netscript Function

isLogEnabled(*fn*)

Arguments

- **fn** (*string*) – Name of function to check

RAM cost 0 GB

Returns a boolean indicating whether or not logging is enabled for that function (or 'ALL')

1.1.5.15 getScriptLogs() Netscript Function

getScriptLogs([*fn*][, *hostname/ip=current ip*][, *args...*])

Arguments

- **fn** (*string*) – Optional. Filename of script to get logs from.
- **ip** (*string*) – Optional. IP or hostname of the server that the script is on
- **args...** – Arguments to identify which scripts to get logs for

RAM cost 0 GB

Returns a script's logs. The logs are returned as an array, where each line is an element in the array. The most recently logged line is at the end of the array.

Note that there is a maximum number of lines that a script stores in its logs. This is configurable in the game's options.

If the function is called with no arguments, it will return the current script's logs.

Otherwise, the *fn*, *hostname/ip,* and *args...* arguments can be used to get the logs from another script. Remember that scripts are uniquely identified by both their names and arguments.

Examples:

```
// Get logs from foo.script on the current server that was run with no args
getScriptLogs("foo.script");

// Get logs from foo.script on the foodnstuff server that was run with no args
getScriptLogs("foo.script", "foodnstuff");

// Get logs from foo.script on the foodnstuff server that was run with the␣
↪arguments [1, "test"]
getScriptLogs("foo.script", "foodnstuff", 1, "test");
```

1.1.5.16 tail() Netscript Function

tail([*fn*][, *hostname/ip=current ip*][, *...args*])

Arguments

- **fn** (*string*) – Optional. Filename of script to get logs from.
- **ip** (*string*) – Optional. IP or hostname of the server that the script is on
- **args...** – Arguments to identify which scripts to get logs for

RAM cost 0 GB

Opens a script's logs. This is functionally the same as the *tail* Terminal command.

If the function is called with no arguments, it will open the current script's logs.

Otherwise, the *fn*, *hostname/ip,* and *args...* arguments can be used to get the logs from another script. Remember that scripts are uniquely identified by both their names and arguments.

Examples:

```
// Open logs from foo.script on the current server that was run with no args
tail("foo.script");

// Open logs from foo.script on the foodnstuff server that was run with no args
tail("foo.script", "foodnstuff");

// Open logs from foo.script on the foodnstuff server that was run with the␣
↪arguments [1, "test"]
tail("foo.script", "foodnstuff", 1, "test");
```

1.1.5.17 scan() Netscript Function

scan(*hostname/ip=current ip*[, *hostnames=true*])

Arguments

- **hostname/ip** (*string*) – IP or hostname of the server to scan
- **boolean** – Optional boolean specifying whether the function should output hostnames (if true) or IP addresses (if false)

RAM cost 0.2 GB

Returns an array containing the hostnames or IPs of all servers that are one node way from the specified target server. The hostnames/IPs in the returned array are strings.

1.1.5.18 nuke() Netscript Function

nuke(*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – IP or hostname of the target server

RAM cost 0 GB

Runs the NUKE.exe program on the target server. NUKE.exe must exist on your home computer.

Example:

```
nuke("foodnstuff");
```

1.1.5.19 brutessh() Netscript Function

brutessh (*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – IP or hostname of the target server

RAM cost 0 GB

Runs the BruteSSH.exe program on the target server. BruteSSH.exe must exist on your home computer.

Example:

```
brutessh("foodnstuff");
```

1.1.5.20 ftpcrack() Netscript Function

ftpcrack (*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – IP or hostname of the target server

RAM cost 0 GB

Runs the FTPCrack.exe program on the target server. FTPCrack.exe must exist on your home computer.

Example:

```
ftpcrack("foodnstuff");
```

1.1.5.21 relaysmtp() Netscript Function

relaysmtp (*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – IP or hostname of the target server

RAM cost 0 GB

Runs the relaySMTP.exe program on the target server. relaySMTP.exe must exist on your home computer.

Example:

```
relaysmtp("foodnstuff");
```

1.1.5.22 httpworm() Netscript Function

httpworm (*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – IP or hostname of the target server

RAM cost 0 GB

Runs the HTTPWorm.exe program on the target server. HTTPWorm.exe must exist on your home computer.

Example:

```
httpworm("foodnstuff");
```

1.1.5.23 sqlinject() Netscript Function

sqlinject (*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – IP or hostname of the target server

RAM cost 0 GB

Runs the SQLInject.exe program on the target server. SQLInject.exe must exist on your home computer.

Example:

```
sqlinject("foodnstuff");
```

1.1.5.24 run() Netscript Function

run (*script*[, *numThreads=1*][, *args...*])

Arguments

- **script** (*string*) – Filename of script to run
- **numThreads** (*number*) – Optional thread count for new script. Set to 1 by default. Will be rounded to nearest integer
- **args...** – Additional arguments to pass into the new script that is being run. Note that if any arguments are being passed into the new script, then the second argument *numThreads* must be filled in with a value.

RAM cost 1 GB

Run a script as a separate process. This function can only be used to run scripts located on the current server (the server running the script that calls this function).

If the script was successfully started, then this functions returns the PID of that script. Otherwise, it returns 0.

Note: PID stands for Process ID. The PID is a unique identifier for each script. The PID will always be a positive integer.

Warning: Running this function with a *numThreads* argument of 0 will return 0 without running the script. However, running this function with a negative *numThreads* argument will cause a runtime error.

The simplest way to use the *run* command is to call it with just the script name. The following example will run 'foo.script' single-threaded with no arguments:

```
run("foo.script");
```

The following example will run 'foo.script' but with 5 threads instead of single-threaded:

```
run("foo.script", 5);
```

This next example will run 'foo.script' single-threaded, and will pass the string 'foodnstuff' into the script as an argument:

```
run("foo.script", 1, 'foodnstuff');
```

1.1.5.25 exec() Netscript Function

exec (*script*, *hostname/ip*[, *numThreads=1*][, *args...*])

Arguments

- **script** (*string*) – Filename of script to execute
- **hostname/ip** (*string*) – IP or hostname of the 'target server' on which to execute the script
- **numThreads** (*number*) – Optional thread count for new script. Set to 1 by default. Will be rounded to nearest integer
- **args...** – Additional arguments to pass into the new script that is being run. Note that if any arguments are being passed into the new script, then the third argument *numThreads* must be filled in with a value.

RAM cost 1.3 GB

Run a script as a separate process on a specified server. This is similar to the *run* function except that it can be used to run a script on any server, instead of just the current server.

If the script was successfully started, then this functions returns the PID of that script. Otherwise, it returns 0.

Note: PID stands for Process ID. The PID is a unique identifier for each script. The PID will always be a positive integer.

Warning: Running this function with a *numThreads* argument of 0 will return 0 without running the script. However, running this function with a negative *numThreads* argument will cause a runtime error.

The simplest way to use the *exec* command is to call it with just the script name and the target server. The following example will try to run *generic-hack.script* on the *foodnstuff* server:

```
exec("generic-hack.script", "foodnstuff");
```

The following example will try to run the script *generic-hack.script* on the *joesguns* server with 10 threads:

```
exec("generic-hack.script", "joesguns", 10);
```

This last example will try to run the script *foo.script* on the *foodnstuff* server with 5 threads. It will also pass the number 1 and the string "test" in as arguments to the script:

```
exec("foo.script", "foodnstuff", 5, 1, "test");
```

1.1.5.26 spawn() Netscript Function

spawn (*script*, *numThreads*[, *args...*])

Arguments

- **script** (*string*) – Filename of script to execute
- **numThreads** (*number*) – Number of threads to spawn new script with. Will be rounded to nearest integer
- **args...** – Additional arguments to pass into the new script that is being run.

RAM cost 2 GB

Terminates the current script, and then after a delay of about 10 seconds it will execute the newly-specified script. The purpose of this function is to execute a new script without being constrained by the RAM usage of the current one. This function can only be used to run scripts on the local server.

Because this function immediately terminates the script, it does not have a return value.

The following example will execute the script 'foo.script' with 10 threads and the arguments 'foodnstuff' and 90:

```
spawn('foo.script', 10, 'foodnstuff', 90);
```

1.1.5.27 kill() Netscript Function

kill(*script*, *hostname/ip*[, *args...*])

Arguments

- **script** (*string*) – Filename of the script to kill
- **hostname/ip** (*string*) – IP or hostname of the server on which to kill the script
- **args...** – Arguments to identify which script to kill

RAM cost 0.5 GB

Kills the script on the target server specified by the script's name and arguments. Remember that scripts are uniquely identified by both their name and arguments. For example, if *foo.script* is run with the argument 1, then this is not the same as *foo.script* run with the argument 2, even though they have the same code.

If this function successfully kills the specified script, then it will return true. Otherwise, it will return false.

Examples:

The following example will try to kill a script named *foo.script* on the *foodnstuff* server that was ran with no arguments:

```
kill("foo.script", "foodnstuff");
```

The following will try to kill a script named *foo.script* on the current server that was ran with no arguments:

```
kill("foo.script", getHostname());
```

The following will try to kill a script named *foo.script* on the current server that was ran with the arguments 1 and "foodnstuff":

```
kill("foo.script", getHostname(), 1, "foodnstuff");
```

kill(*scriptPid*)

Arguments

- **scriptPid** (*number*) – PID of the script to kill

RAM cost 0.5 GB

Kills the script with the specified PID. Killing a script by its PID will typically have better performance, especially if you have many scripts running.

If this function successfully kills the specified script, then it will return true. Otherwise, it will return false.

Examples:

The following example will try to kill the script with the PID 10:

```
if (kill(10)) {
    print("Killed script with PID 10!");
}
```

1.1.5.28 killall() Netscript Function

killall(*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – IP or hostname of the server on which to kill all scripts

RAM cost 0.5 GB

Kills all running scripts on the specified server. This function returns true if any scripts were killed, and false otherwise. In other words, it will return true if there are any scripts running on the target server.

1.1.5.29 exit() Netscript Function

exit()

RAM cost 0 GB

Terminates the current script immediately

1.1.5.30 scp() Netscript Function

scp(*files*[, *source*], *destination*)

Arguments

- **files** (*string/array*) – Filename or an array of filenames of script/literature files to copy
- **source** (*string*) – Hostname or IP of the source server, which is the server from which the file will be copied. This argument is optional and if it's omitted the source will be the current server.
- **destination** (*string*) – Hostname or IP of the destination server, which is the server to which the file will be copied.

RAM cost 0.6 GB

Copies a script or literature (.lit) file(s) to another server. The *files* argument can be either a string specifying a single file to copy, or an array of strings specifying multiple files to copy.

Returns true if the script/literature file is successfully copied over and false otherwise. If the *files* argument is an array then this function will return true if at least one of the files in the array is successfully copied.

Examples:

```
//Copies hack-template.script from the current server to foodnstuff
scp("hack-template.script", "foodnstuff");

//Copies foo.lit from the helios server to the home computer
scp("foo.lit", "helios", "home");

//Tries to copy three files from rothman-uni to home computer
files = ["foo1.lit", "foo2.script", "foo3.script"];
scp(files, "rothman-uni", "home");
```

1.1.5.31 ls() Netscript Function

ls (*hostname/ip*[, *grep*])

Arguments

- **hostname/ip** (*string*) – Hostname or IP of the target server
- **grep** (*string*) – a substring to search for in the filename

RAM cost 0 GB

Returns an array with the filenames of all files on the specified server (as strings). The returned array is sorted in alphabetic order

1.1.5.32 ps() Netscript Function

ps (*hostname/ip=current ip*)

Arguments

- **ip** (*string*) – Hostname or IP address of the target server. If not specified, it will be the current server's IP by default

RAM cost 0.2 GB

Returns an array with general information about all scripts running on the specified target server. The information for each server is given in an object with the following structure:

```
{
    filename:   Script name,
    threads:    Number of threads script is running with,
    args:       Script's arguments
}
```

Example usage (using *NetscriptJS (Netscript 2.0)*):

```
export async function main(ns) {
    const ps = ns.ps("home");
    for (let i = 0; i < ps.length; ++i) {
        ns.tprint(ps[i].filename + ' ' + ps[i].threads);
        ns.tprint(ps[i].args);
    }
}
```

1.1.5.33 hasRootAccess() Netscript Function

hasRootAccess(*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – Hostname or IP of the target server

RAM cost 0.05 GB

Returns a boolean indicating whether or not the player has root access to the specified target server.

Example:

```
if (hasRootAccess("foodnstuff") == false) {
    nuke("foodnstuff");
}
```

1.1.5.34 getHostname() Netscript Function

getHostname()

RAM cost 0.05 GB

Returns a string with the hostname of the server that the script is running on

1.1.5.35 getHackingLevel() Netscript Function

getHackingLevel()

RAM cost 0.05 GB

Returns the player's current hacking level

1.1.5.36 getHackingMultipliers() Netscript Function

getHackingMultipliers()

RAM cost 4 GB

Returns an object containing the Player's hacking related multipliers. These multipliers are returned in decimal forms, not percentages (e.g. 1.5 instead of 150%). The object has the following structure:

```
{
    chance: Player's hacking chance multiplier,
    speed: Player's hacking speed multiplier,
    money: Player's hacking money stolen multiplier,
    growth: Player's hacking growth multiplier
}
```

Example of how this can be used:

```
mults = getHackingMultipliers();
print(mults.chance);
print(mults.growth);
```

1.1.5.37 getHacknetMultipliers() Netscript Function

getHacknetMultipliers()

RAM cost 4 GB

Returns an object containing the Player's hacknet related multipliers. These multipliers are returned in decimal forms, not percentages (e.g. 1.5 instead of 150%). The object has the following structure:

```
{
    production: Player's hacknet production multiplier,
    purchaseCost: Player's hacknet purchase cost multiplier,
    ramCost: Player's hacknet ram cost multiplier,
    coreCost: Player's hacknet core cost multiplier,
    levelCost: Player's hacknet level cost multiplier
}
```

Example of how this can be used:

```
mults = getHacknetMultipliers();
print(mults.production);
print(mults.purchaseCost);
```

1.1.5.38 getServerMoneyAvailable() Netscript Function

getServerMoneyAvailable(*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – Hostname or IP of target server

RAM cost 0.1 GB

Returns the amount of money available on a server. **Running this function on the home computer will return the player's money.**

Example:

```
getServerMoneyAvailable("foodnstuff");
getServerMoneyAvailable("home"); //Returns player's money
```

1.1.5.39 getServerMaxMoney() Netscript Function

getServerMaxMoney(*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – Hostname or IP of target server

RAM cost 0.1 GB

Returns the maximum amount of money that can be available on a server

1.1.5.40 getServerGrowth() Netscript Function

getServerGrowth(*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – Hostname or IP of target server

RAM cost 0.1 GB

Returns the server's instrinsic "growth parameter". This growth parameter is a number between 1 and 100 that represents how quickly the server's money grows. This parameter affects the percentage by which the server's money is increased when using the *grow()* function. A higher growth parameter will result in a higher percentage increase from *grow()*.

1.1.5.41 getServerSecurityLevel() Netscript Function

getServerSecurityLevel(*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – Hostname or IP of target server

RAM cost 0.1 GB

Returns the security level of the target server. A server's security level is denoted by a number, typically between 1 and 100 (but it can go above 100).

1.1.5.42 getServerBaseSecurityLevel() Netscript Function

getServerBaseSecurityLevel(*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – Hostname or IP of target server

RAM cost 0.1 GB

Returns the base security level of the target server. This is the security level that the server starts out with. This is different than *getServerSecurityLevel()* because *getServerSecurityLevel()* returns the current security level of a server, which can constantly change due to *hack()*, *grow()*, and *weaken()*, calls on that server. The base security level will stay the same until you reset by installing an Augmentation(s).

1.1.5.43 getServerMinSecurityLevel() Netscript Function

getServerMinSecurityLevel(*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – Hostname or IP of target server

RAM cost 0.1 GB

Returns the minimum security level of the target server

1.1.5.44 getServerRequiredHackingLevel() Netscript Function

getServerRequiredHackingLevel(*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – Hostname or IP of target server

RAM cost 0.1 GB

Returns the required hacking level of the target server

1.1.5.45 getServerNumPortsRequired() Netscript Function

getServerNumPortsRequired(*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – Hostname or IP of target server

RAM cost 0.1 GB

Returns the number of open ports required to successfully run NUKE.exe on the specified server.

1.1.5.46 getServerRam() Netscript Function

getServerRam(*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – Hostname or IP of target server

RAM cost 0.1 GB

Returns an array with two elements that gives information about a server's memory (RAM). The first element in the array is the amount of RAM that the server has total (in GB). The second element in the array is the amount of RAM that is currently being used on the server (in GB).

Example:

```
res = getServerRam("helios");
totalRam = res[0];
ramUsed = res[1];
```

1.1.5.47 serverExists() Netscript Function

serverExists(*hostname/ip*)

Arguments

- **hostname/ip** (*string*) – Hostname or IP of target server

RAM cost 0.1 GB

Returns a boolean denoting whether or not the specified server exists

1.1.5.48 fileExists() Netscript Function

fileExists(*filename*[, *hostname/ip*])

Arguments

- **filename** (*string*) – Filename of file to check
- **hostname/ip** (*string*) – Hostname or IP of target server. This is optional. If it is not specified then the function will use the current server as the target server

RAM cost 0.1 GB

Returns a boolean indicating whether the specified file exists on the target server. The filename for scripts is case-sensitive, but for other types of files it is not. For example, *fileExists("brutessh.exe")* will work fine, even though the actual program is named "BruteSSH.exe".

If the *hostname/ip* argument is omitted, then the function will search through the current server (the server running the script that calls this function) for the file.

Examples:

```
fileExists("foo.script", "foodnstuff");
fileExists("ftpcrack.exe");
```

The first example above will return true if the script named *foo.script* exists on the *foodnstuff* server, and false otherwise. The second example above will return true if the current server contains the *FTPCrack.exe* program, and false otherwise.

1.1.5.49 isRunning() Netscript Function

isRunning(*filename*, *hostname/ip*[, *args...*])

Arguments

- **filename** (*string*) – Filename of script to check. This is case-sensitive.
- **hostname/ip** (*string*) – Hostname or IP of target server
- **args...** – Arguments to specify/identify which scripts to search for

RAM cost 0.1 GB

Returns a boolean indicating whether the specified script is running on the target server. Remember that a script is uniquely identified by both its name and its arguments.

Examples:

In this first example below, the function call will return true if there is a script named *foo.script* with no arguments running on the *foodnstuff* server, and false otherwise:

```
isRunning("foo.script", "foodnstuff");
```

In this second example below, the function call will return true if there is a script named *foo.script* with no arguments running on the current server, and false otherwise:

```
isRunning("foo.script", getHostname());
```

In this next example below, the function call will return true if there is a script named *foo.script* running with the arguments 1, 5, and "test" (in that order) on the *joesguns* server, and false otherwise:

```
isRunning("foo.script", "joesguns", 1, 5, "test");
```

1.1.5.50 getPurchasedServerCost() Netscript Function

getPurchasedServerCost(*ram*)

RAM cost 0.25 GB

Arguments

- **ram** (*number*) – Amount of RAM of a potential purchased server. Must be a power of 2 (2, 4, 8, 16, etc.). Maximum value of 1048576 (2^20)

Returns the cost to purchase a server with the specified amount of *ram*.

Examples:

```
for (i = 1; i <= 20; i++) {
    tprint(i + " -- " + getPurchasedServerCost(Math.pow(2, i)));
}
```

1.1.5.51 purchaseServer() Netscript Function

purchaseServer(*hostname*, *ram*)

Arguments

- **hostname** (*string*) – Hostname of the purchased server
- **ram** (*number*) – Amount of RAM of the purchased server. Must be a power of 2. Maximum value of *getPurchasedServerMaxRam()*

RAM cost 2.25 GB

Purchased a server with the specified hostname and amount of RAM.

The *hostname* argument can be any data type, but it will be converted to a string and have whitespace removed. Anything that resolves to an empty string will cause the function to fail. If there is already a server with the specified hostname, then the function will automatically append a number at the end of the *hostname* argument value until it finds a unique hostname. For example, if the script calls *purchaseServer("foo", 4)* but a server named "foo" already exists, the it will automatically change the hostname to "foo-0". If there is already a server with the hostname "foo-0", then it will change the hostname to "foo-1", and so on.

Note that there is a maximum limit to the amount of servers you can purchase.

Returns the hostname of the newly purchased server as a string. If the function fails to purchase a server, then it will return an empty string. The function will fail if the arguments passed in are invalid, if the player does not have enough money to purchase the specified server, or if the player has exceeded the maximum amount of servers.

Example:

```
ram = 64;
hn = "pserv-";
for (i = 0; i < 5; ++i) {
    purchaseServer(hn + i, ram);
}
```

1.1.5.52 deleteServer() Netscript Function

deleteServer(*hostname*)

Arguments

- **hostname** (*string*) – Hostname of the server to delete

RAM cost 2.25 GB

Deletes one of your purchased servers, which is specified by its hostname.

The *hostname* argument can be any data type, but it will be converted to a string. Whitespace is automatically removed from the string. This function will not delete a server that still has scripts running on it.

Returns true if successful, and false otherwise.

1.1.5.53 getPurchasedServers() Netscript Function

getPurchasedServers([*hostname=true*])

Arguments

- **hostname** (*boolean*) – Specifies whether hostnames or IP addresses should be returned. If it's true then hostnames will be returned, and if false then IPs will be returned. If this argument is omitted then it is true by default

RAM cost 2.25 GB

Returns an array with either the hostnames or IPs of all of the servers you have purchased.

1.1.5.54 getPurchasedServerLimit() Netscript Function

getPurchasedServerLimit()

RAM cost 0.05 GB

Returns the maximum number of servers you can purchase

1.1.5.55 getPurchasedServerMaxRam() Netscript Function

getPurchasedServerMaxRam()

RAM cost 0.05 GB

Returns the maximum RAM that a purchased server can have

1.1.5.56 write() Netscript Function

write(*port/fn*, *data=""*, *mode="a"*)

Arguments

- **port/fn** (*string/number*) – Port or text file/script that will be written to
- **data** (*string*) – Data to write
- **mode** (*string*) – Defines the write mode. Only valid when writing to text files or scripts.

RAM cost 1 GB

This function can be used to either write data to a port, a text file (.txt), or a script (.script, .js, .ns)

If the first argument is a number between 1 and 20, then it specifies a port and this function will write *data* to that port. Read about how *Netscript Ports* work here. The third argument, *mode*, is not used when writing to a port.

If the first argument is a string, then it specifies the name of a text file or script and this function will write *data* to that text file/script. If the specified text file/script does not exist, then it will be created. The third argument *mode*, defines how the data will be written. If *mode* is set to "w", then the data is written in "write" mode which means that it will overwrite all existing data on the text file/script. If *mode* is set to any other value then the data will be written in "append" mode which means that the data will be added at the end of the file.

1.1.5.57 tryWrite() Netscript Function

tryWrite(*port*, *data=""*)

Arguments

- **port** (*number*) – Port to be written to
- **data** (*string*) – Data to try to write

Returns True if the data is successfully written to the port, and false otherwise

RAM cost 1 GB

Attempts to write data to the specified Netscript Port. If the port is full, the data will not be written. Otherwise, the data will be written normally

1.1.5.58 read() Netscript Function

read(*port/fn*)

Arguments

- **port/fn** (*string/number*) – Port or text file to read from

RAM cost 1 GB

This function is used to read data from a port, a text file (.txt), or a script (.script, .js, .ns)

If the argument *port/fn* is a number between 1 and 20, then it specifies a port and it will read data from that port. Read about how *Netscript Ports* work here. A port is a serialized queue. This function will remove the first element from that queue and return it. If the queue is empty, then the string "NULL PORT DATA" will be returned.

If the argument *port/fn* is a string, then it specifies the name of a text file or script and this function will return the data in the specified text file/script. If the text file does not exist, an empty string will be returned.

1.1.5.59 peek() Netscript Function

peek(*port*)

Arguments

- **port** (*number*) – Port to peek. Must be an integer between 1 and 20

RAM cost 1 GB

This function is used to peek at the data from a port. It returns the first element in the specified port without removing that element. If the port is empty, the string "NULL PORT DATA" will be returned.

Read about how *Netscript Ports* work here

1.1.5.60 clear() Netscript Function

clear(*port/fn*)

Arguments

- **port/fn** (*string/number*) – Port or text file to clear

RAM cost 1 GB

This function is used to clear data in a *Netscript Port* or a text file.

If the *port/fn* argument is a number between 1 and 20, then it specifies a port and will clear it (deleting all data from the underlying queue).

If the *port/fn* argument is a string, then it specifies the name of a text file (.txt) and will delete all data from that text file.

1.1.5.61 getPortHandle() Netscript Function

getPortHandle (*port*)

Arguments

- **port** (*number*) – Port number

RAM cost 10 GB

Get a handle to a Netscript Port. See more details here: *Netscript Ports*

WARNING: Port Handles only work in *NetscriptJS (Netscript 2.0)*. They will not work in *Netscript 1.0*.

1.1.5.62 rm() Netscript Function

rm (*fn*[, *hostname/ip=current server*])

Arguments

- **fn** (*string*) – Filename of file to remove. Must include the extension
- **hostname/ip** (*string*) – Hostname or IP Address of the server on which to delete the file. Optional. Defaults to current server

Returns True if it successfully deletes the file, and false otherwise

RAM cost 1 GB

Removes the specified file from the current server. This function works for every file type except message (.msg) files.

1.1.5.63 scriptRunning() Netscript Function

scriptRunning (*scriptname*, *hostname/ip*)

Arguments

- **scriptname** (*string*) – Filename of script to check. This is case-sensitive.
- **hostname/ip** (*string*) – Hostname or IP of target server

RAM cost 1 GB

Returns a boolean indicating whether any instance of the specified script is running on the target server, regardless of its arguments.

This is different than the *isRunning()* function because it does not try to identify a specific instance of a running script by its arguments.

Examples:

The example below will return true if there is any script named *foo.script* running on the *foodnstuff* server, and false otherwise:

```
scriptRunning("foo.script", "foodnstuff");
```

The example below will return true if there is any script named "foo.script" running on the current server, and false otherwise:

```
scriptRunning("foo.script", getHostname());
```

1.1.5.64 scriptKill() Netscript Function

scriptKill(*scriptname*, *hostname/ip*)

Arguments

- **scriptname** (*string*) – Filename of script to kill. This is case-sensitive.
- **hostname/ip** (*string*) – Hostname or IP of target server

RAM cost 1 GB

Kills all scripts with the specified filename on the target server specified by *hostname/ip*, regardless of arguments. Returns true if one or more scripts were successfully killed, and false if none were.

1.1.5.65 getScriptName() Netscript Function

getScriptName()

RAM cost 0 GB

Returns the current script name

1.1.5.66 getScriptRam() Netscript Function

getScriptRam(*scriptname*[, *hostname/ip*])

Arguments

- **scriptname** (*string*) – Filename of script. This is case-sensitive.
- **hostname/ip** (*string*) – Hostname or IP of target server the script is located on. This is optional, If it is not specified then the function will set the current server as the target server.

RAM cost 0.1 GB

Returns the amount of RAM required to run the specified script on the target server. Returns 0 if the script does not exist.

1.1.5.67 getHackTime() Netscript Function

getHackTime(*hostname/ip*[, *hackLvl=current level*])

Arguments

- **hostname/ip** (*string*) – Hostname or IP of target server
- **hackLvl** (*number*) – Optional hacking level for the calculation. Defaults to player's current hacking level

RAM cost 0.05 GB

Returns the amount of time in seconds it takes to execute the *hack()* Netscript function on the target server.

The function takes in an optional *hackLvl* parameter that can be specified to see what the hack time would be at different hacking levels.

Note: For Hacknet Servers (the upgraded version of a Hacknet Node), this function will return `Infinity`.

1.1.5.68 getGrowTime() Netscript Function

`getGrowTime` (*hostname/ip*[, *hackLvl=current level*])

Arguments

- **`hostname/ip`** (*`string`*) – Hostname or IP of target server
- **`hackLvl`** (*`number`*) – Optional hacking level for the calculation. Defaults to player's current hacking level

RAM cost 0.05 GB

Returns the amount of time in seconds it takes to execute the *grow()* Netscript function on the target server.

The function takes in an optional *hackLvl* parameter that can be specified to see what the grow time would be at different hacking levels.

Note: For Hacknet Servers (the upgraded version of a Hacknet Node), this function will return `Infinity`.

1.1.5.69 getWeakenTime() Netscript Function

`getWeakenTime` (*hostname/ip*[, *hackLvl=current level*])

Arguments

- **`hostname/ip`** (*`string`*) – Hostname or IP of target server
- **`hackLvl`** (*`number`*) – Optional hacking level for the calculation. Defaults to player's current hacking level

RAM cost 0.05 GB

Returns the amount of time in seconds it takes to execute the *weaken()* Netscript function on the target server.

The function takes in an optional *hackLvl* parameter that can be specified to see what the weaken time would be at different hacking levels.

Note: For Hacknet Servers (the upgraded version of a Hacknet Node), this function will return `Infinity`.

1.1.5.70 getScriptIncome() Netscript Function

`getScriptIncome` ([*scriptname*][, *hostname/ip*][, *args...*])

Arguments

- **scriptname** (*string*) – Filename of script
- **hostname/ip** (*string*) – Server on which script is running
- **args...** – Arguments that the script is running with

RAM cost 0.1 GB

Returns the amount of income the specified script generates while online (when the game is open, does not apply for offline income). Remember that a script is uniquely identified by both its name and its arguments. So for example if you ran a script with the arguments "foodnstuff" and "5" then in order to use this function to get that script's income you must specify those same arguments in the same order in this function call.

This function can also be called with no arguments. If called with no arguments, then this function will return an array of two values. The first value is the total income ($ / second) of all of your active scripts (scripts that are currently running on any server). The second value is the total income ($ / second) that you've earned from scripts since you last installed Augmentations.

1.1.5.71 getScriptExpGain() Netscript Function

getScriptExpGain([*scriptname*][, *hostname/ip*][, *args...*])

Arguments

- **scriptname** (*string*) – Filename of script
- **hostname/ip** (*string*) – Server on which script is running
- **args...** – Arguments that the script is running with

RAM cost 0.1 GB

Returns the amount of hacking experience the specified script generates while online (when the game is open, does not apply for offline experience gains). Remember that a script is uniquely identified by both its name and its arguments.

This function can also return the total experience gain rate of all of your active scripts by running the function with no arguments.

1.1.5.72 getTimeSinceLastAug() Netscript Function

getTimeSinceLastAug()

RAM cost 0.05 GB

Returns the amount of time in milliseconds that have passed since you last installed Augmentations

1.1.5.73 sprintf() Netscript Function

sprintf()

RAM cost 0 GB

See this link for details.

1.1.5.74 vsprintf() Netscript Function

vsprintf()

RAM cost 0 GB

See this link for details.

1.1.5.75 nFormat() Netscript Function

nFormat(*n*, *format*)

Arguments

- **n** (*number*) – Number to format
- **format** (*string*) – Formatter

Converts a number into a string with the specified formatter. This uses the numeraljs library, so the formatters must be compatible with that.

This is the same function that the game itself uses to display numbers.

Examples:

```
nFormat(1.23e9, "$0.000a"); // Returns "$1.230b"
nFormat(12345.678, "0,0");  // Returns "12,346"
nFormat(0.84, "0.0%");      // Returns "84.0%
```

1.1.5.76 prompt() Netscript Function

prompt(*txt*)

Arguments

- **txt** (*string*) – Text to appear in the prompt dialog box

RAM cost 0 GB

Prompts the player with a dialog box with two options: "Yes" and "No". This function will return true if the player click "Yes" and false if the player clicks "No". The script's execution is halted until the player selects one of the options.

1.1.5.77 wget() Netscript Function

wget(*url*, *target*[, *hostname/ip=current ip*])

Arguments

- **url** (*string*) – URL to pull data from
- **target** (*string*) – Filename to write data to. Must be script or text file
- **ip** (*string*) – Optional hostname/ip of server for target file.

RAM cost 0 GB

Retrieves data from a URL and downloads it to a file on the specified server. The data can only be downloaded to a script (.script, .ns, .js) or a text file (.txt). If the file already exists, it will be overwritten by this command.

Note that it will not be possible to download data from many websites because they do not allow cross-origin resource sharing (CORS). Example:

```
wget("https://raw.githubusercontent.com/danielyxie/bitburner/master/README.md",
↪"game_readme.txt");
```

IMPORTANT: This is an asynchronous function that returns a Promise. The Promise's resolved value will be a boolean indicating whether or not the data was successfully retrieved from the URL. Because the function is async and returns a Promise, it is recommended you use `wget` in *NetscriptJS (Netscript 2.0)*.

In NetscriptJS, you must preface any call to `wget` with the `await` keyword (like you would `hack` or `sleep`).

`wget` will still work in *Netscript 1.0*, but the functions execution will not be synchronous (i.e. it may not execute when you expect/want it to). Furthermore, since Promises are not supported in ES5, you will not be able to process the returned value of `wget` in Netscript 1.0.

1.1.5.78 getFavorToDonate() Netscript Function

`getFavorToDonate``()`

RAM cost 0.1 GB

Returns the amount of Faction favor required to be able to donate to a faction.

1.1.6 Netscript Advanced Functions

These Netscript functions become relevant later on in the game. They are put on a separate page because they contain spoilers for the game.

Warning: This page contains spoilers for the game

1.1.6.1 getBitNodeMultipliers() Netscript Function

`getBitNodeMultipliers``()`

Returns an object containing the current BitNode multipliers. This function requires Source-File 5 in order to run. The multipliers are returned in decimal forms (e.g. 1.5 instead of 150%). The multipliers represent the difference between the current BitNode and the original BitNode (BitNode-1). For example, if the *CrimeMoney* multiplier has a value of 0.1, then that means that committing crimes in the current BitNode will only give 10% of the money you would have received in BitNode-1.

The structure of the returned object is subject to change as BitNode multipliers get added to the game. Refer to the source code here to see the name of the BitNode multipliers.

Example:

```
mults = getBitNodeMultipliers();
print(mults.ServerMaxMoney);
print(mults.HackExpGain);
```

1.1.6.2 getHackTime(), getGrowTime(), & getWeakenTime()

The *getHackTime()*, *getGrowTime()*, and *getWeakenTime()* all take an additional third optional parameter for specifying a specific intelligence level to see how that would affect the hack/grow/weaken times. This parameter defaults to your current intelligence level.

(Intelligence is unlocked after obtaining Source-File 5).

The function signatures are then:

```
getHackTime(hostname/ip[, hackLvl=current level, intLvl=current level])
getGrowTime(hostname/ip[, hackLvl=current level, intLvl=current level])
getWeakenTime(hostname/ip[, hackLvl=current level, intLvl=current level])
```

1.1.7 Netscript Hacknet Node API

Warning: Not all functions in the Hacknet Node API are immediately available. For this reason, the documentation for this API may contains spoilers for the game.

Netscript provides the following API for accessing and upgrading your Hacknet Nodes through scripts.

Note that none of these functions will write to the script's logs. If you want to see what your script is doing you will have to print to the logs yourself.

Hacknet Node API functions must be accessed through the hacknet namespace

In Netscript 1.0:

```
hacknet.purchaseNode();
hacknet.getNodeStats(3).level;
```

In *NetscriptJS (Netscript 2.0)*:

```
ns.hacknet.purchaseNode();
ns.hacknet.getNodeStats(3).level;
```

1.1.7.1 numNodes() Netscript Function

numNodes()
: Returns the number of Hacknet Nodes you own.

1.1.7.2 purchaseNode() Netscript Function

purchaseNode()
: Purchases a new Hacknet Node. Returns a number with the index of the Hacknet Node. This index is equivalent to the number at the end of the Hacknet Node's name (e.g The Hacknet Node named 'hacknet-node-4' will have an index of 4).

  If the player cannot afford to purchase a new Hacknet Node then the function will return -1.

1.1.7.3 getPurchaseNodeCost() Netscript Function

getPurchaseNodeCost()
Returns the cost of purchasing a new Hacknet Node

1.1.7.4 getNodeStats() Netscript Function

Warning: This page contains spoilers for the game

getNodeStats(*i*)

Arguments

- **i** (*number*) – Index/Identifier of Hacknet Node. *See here for details*

Returns an object containing a variety of stats about the specified Hacknet Node:

```
{
    name:               Node's name ("hacknet-node-5"),
    level:              Node's level,
    ram:                Node's RAM,
    cores:              Node's number of cores,
    cache:              Cache level. Only applicable for Hacknet Servers
    hashCapacity:       Hash Capacity provided by this Node. Only applicable for␣
↪Hacknet Servers
    production:         Node's production per second
    timeOnline:         Number of seconds since Node has been purchased,
    totalProduction:    Total amount that the Node has produced
}
```

Note: Note that for Hacknet Nodes, production refers to the amount of money the node generates. For Hacknet Servers (the upgraded version of Hacknet Nodes), production refers to the amount of hashes the node generates.

1.1.7.5 upgradeLevel() Netscript Function

upgradeLevel(*i*, *n*)

Arguments

- **i** (*number*) – Index/Identifier of Hacknet Node. *See here for details*
- **n** (*number*) – Number of levels to purchase. Must be positive. Rounded to nearest integer

Tries to upgrade the level of the specified Hacknet Node by *n*.

Returns true if the Hacknet Node's level is successfully upgraded by *n* or if it is upgraded by some positive amount and the Node reaches its max level.

Returns false otherwise.

1.1.7.6 upgradeRam() Netscript Function

upgradeRam(*i*, *n*)

Arguments

- **i** (*number*) – Index/Identifier of Hacknet Node. *See here for details*
- **n** (*number*) – Number of times to upgrade RAM. Must be positive. Rounded to nearest integer

Tries to upgrade the specified Hacknet Node's RAM *n* times. Note that each upgrade doubles the Node's RAM. So this is equivalent to multiplying the Node's RAM by 2^n.

Returns true if the Hacknet Node's RAM is successfully upgraded *n* times or if it is upgraded some positive number of times and the Node reaches it max RAM.

Returns false otherwise.

1.1.7.7 upgradeCore() Netscript Function

upgradeCore(*i*, *n*)

Arguments

- **i** (*number*) – Index/Identifier of Hacknet Node. *See here for details*
- **n** (*number*) – Number of cores to purchase. Must be positive. Rounded to nearest integer

Tries to purchase *n* cores for the specified Hacknet Node.

Returns true if it successfully purchases *n* cores for the Hacknet Node or if it purchases some positive amount and the Node reaches its max number of cores.

Returns false otherwise.

1.1.7.8 upgradeCache() Netscript Function

Warning: This page contains spoilers for the game

upgradeCache(*i*, *n*)

Arguments

- **i** (*number*) – Index/Identifier of Hacknet Node. *See here for details*
- **n** (*number*) – Number of cache levels to purchase. Must be positive. Rounded to nearest integer

Note: This function is only applicable for Hacknet Servers (the upgraded version of a Hacknet Node).

Tries to upgrade the specified Hacknet Server's cache *n* times.

Returns true if it successfully upgrades the Server's cache *n* times, or if it purchases some positive amount and the Server reaches its max cache level.

Returns false otherwise.

1.1.7.9 getLevelUpgradeCost() Netscript Function

getLevelUpgradeCost(*i*, *n*)

Arguments

- **i** (*number*) – Index/Identifier of Hacknet Node. *See here for details*
- **n** (*number*) – Number of levels to upgrade. Must be positive. Rounded to nearest integer

Returns the cost of upgrading the specified Hacknet Node by *n* levels.

If an invalid value for *n* is provided, then this function returns 0. If the specified Hacknet Node is already at max level, then Infinity is returned.

1.1.7.10 getRamUpgradeCost() Netscript Function

getRamUpgradeCost(*i*, *n*)

Arguments

- **i** (*number*) – Index/Identifier of Hacknet Node. *See here for details*
- **n** (*number*) – Number of times to upgrade RAM. Must be positive. Rounded to nearest integer

Returns the cost of upgrading the RAM of the specified Hacknet Node *n* times.

If an invalid value for *n* is provided, then this function returns 0. If the specified Hacknet Node is already at max RAM, then Infinity is returned.

1.1.7.11 getCoreUpgradeCost() Netscript Function

getCoreUpgradeCost(*i*, *n*)

Arguments

- **i** (*number*) – Index/Identifier of Hacknet Node. *See here for details*
- **n** (*number*) – Number of times to upgrade cores. Must be positive. Rounded to nearest integer

Returns the cost of upgrading the number of cores of the specified Hacknet Node by *n*.

If an invalid value for *n* is provided, then this function returns 0. If the specified Hacknet Node is already at the max number of cores, then Infinity is returned.

1.1.7.12 getCacheUpgradeCost() Netscript Function

Warning: This page contains spoilers for the game

getCacheUpgradeCost(*i*, *n*)

Arguments

- **i** (*number*) – Index/Identifier of Hacknet Node. *See here for details*
- **n** (*number*) – Number of times to upgrade cache. Must be positive. Rounded to nearest integer

Note: This function is only applicable for Hacknet Servers (the upgraded version of a Hacknet Node).

Returns the cost of upgrading the cache level of the specified Hacknet Server by *n*.

If an invalid value for *n* is provided, then this function returns 0. If the specified Hacknet Server is already at the max cache level, then Infinity is returned.

1.1.7.13 numHashes() Netscript Function

Warning: This page contains spoilers for the game

numHashes()

Note: This function is only applicable for Hacknet Servers (the upgraded version of a Hacknet Node).

Returns the number of hashes you have

1.1.7.14 hashCost() Netscript Function

Warning: This page contains spoilers for the game

hashCost(*upgName*)

Arguments

- **upgName** (*string*) – Name of upgrade to get the cost of. Must be an exact match

Note: This function is only applicable for Hacknet Servers (the upgraded version of a Hacknet Node).

Returns the number of hashes required for the specified upgrade. The name of the upgrade must be an exact match.

Example:

```
var upgradeName = "Sell for Corporation Funds";
if (hacknet.numHashes() > hacknet.hashCost(upgradeName)) {
    hacknet.spendHashes(upgName);
}
```

1.1.7.15 spendHashes() Netscript Function

Warning: This page contains spoilers for the game

spendHashes(*upgName*, *upgTarget*)

Arguments

- **upgName** (*string*) – Name of upgrade to spend hashes on. Must be an exact match
- **upgTarget** (*string*) – Object to which upgrade applies. Required for certain upgrades

---

Note: This function is only applicable for Hacknet Servers (the upgraded version of a Hacknet Node).

Spend the hashes generated by your Hacknet Servers on an upgrade. Returns a boolean value - true if the upgrade is successfully purchased, and false otherwise.

The name of the upgrade must be an exact match. The `upgTarget` argument is used for upgrades such as `Reduce Minimum Security`, which applies to a specific server. In this case, the `upgTarget` argument must be the hostname of the server.

Example:

```
hacknet.spendHashes("Sell for Corporation Funds");
hacknet.spendHashes("Increase Maximum Money", "foodnstuff");
```

1.1.7.16 Referencing a Hacknet Node

Most of the functions in the Hacknet Node API perform an operation on a single Node. Therefore, a numeric index is used to identify and specify which Hacknet Node a function should act on. This index number corresponds to the number at the end of the name of the Hacknet Node. For example, the first Hacknet Node you purchase will have the name "hacknet-node-0" and is referenced using index 0. The fifth Hacknet Node you purchase will have the name "hacknet-node-4" and is referenced using index 4.

1.1.7.17 RAM Cost

Accessing the *hacknet* namespace incurs a one time cost of 4 GB of RAM. In other words, using multiple Hacknet Node API functions in a script will not cost more than 4 GB of RAM.

1.1.7.18 Utilities

The following functions are not officially part of the Hacknet Node API, but they can be useful when writing Hacknet Node-related scripts. Since they are not part of the API, they do not need to be accessed using the *hacknet* namespace.

- *getHacknetMultipliers()*

1.1.7.19 Example(s)

The following is an example of one way a script can be used to automate the purchasing and upgrading of Hacknet Nodes.

This script attempts to purchase Hacknet Nodes until the player has a total of 8. Then it gradually upgrades those Node's to a minimum of level 140, 64 GB RAM, and 8 cores

```
function myMoney() {
    return getServerMoneyAvailable("home");
}

disableLog("getServerMoneyAvailable");
```

(continues on next page)

(continued from previous page)

```
disableLog("sleep");

var cnt = 8;

while(hacknet.numNodes() < cnt) {
    res = hacknet.purchaseNode();
    print("Purchased hacknet Node with index " + res);
};

for (var i = 0; i < cnt; i++) {
    while (hacknet.getNodeStats(i).level <= 80) {
        var cost = hacknet.getLevelUpgradeCost(i, 10);
        while (myMoney() < cost) {
            print("Need $" + cost + " . Have $" + myMoney());
            sleep(3000);
        }
        res = hacknet.upgradeLevel(i, 10);
    };
};

print("All nodes upgraded to level 80");

for (var i = 0; i < cnt; i++) {
    while (hacknet.getNodeStats(i).ram < 16) {
        var cost = hacknet.getRamUpgradeCost(i, 2);
        while (myMoney() < cost) {
            print("Need $" + cost + " . Have $" + myMoney());
            sleep(3000);
        }
        res = hacknet.upgradeRam(i, 2);
    };
};

print("All nodes upgraded to 16GB RAM");
```

1.1.8 Netscript Trade Information eXchange (TIX) API

The Trade Information eXchange (TIX) is the communications protocol supported by the World Stock Exchange (WSE). The WSE provides an API that allows you to automatically communicate with the *Stock Market*. This API lets you write code using Netscript to build automated trading systems and create your own algorithmic trading strategies. Access to this TIX API can be purchased by visiting the World Stock Exchange in-game.

Access to the TIX API currently costs $5 billion. After you purchase it, you will retain this access even after you 'reset' by installing Augmentations

1.1.8.1 getStockSymbols() Netscript Function()

getStockSymbols()

RAM cost 2 GB

Returns an array of the symbols of the tradable stocks

1.1.8.2 getStockPrice() Netscript Function

getStockPrice(*sym*)

Arguments

- **sym** (*string*) – Stock symbol

RAM cost 2 GB

Given a stock's symbol, returns the price of that stock (the symbol is a sequence of two to four capital letters, **not** the name of the company to which that stock belongs).

Note: The stock's price is the average of its bid and ask price. See *Spread (Bid Price & Ask Price)* for details on what this means.

Example:

```
getStockPrice("FSIG");
```

1.1.8.3 getStockAskPrice() Netscript Function

getStockAskPrice(*sym*)

Arguments

- **sym** (*string*) – Stock symbol

RAM cost 2 GB

Given a stock's symbol, returns the ask price of that stock (the symbol is a sequence of two to four capital letters, **not** the name of the company to which that stock belongs).

See *Spread (Bid Price & Ask Price)* for details on what the ask price is.

1.1.8.4 getStockBidPrice() Netscript Function

getStockBidPrice(*sym*)

Arguments

- **sym** (*string*) – Stock symbol

RAM cost 2 GB

Given a stock's symbol, returns the bid price of that stock (the symbol is a sequence of two to four capital letters, **not** the name of the company to which that stock belongs).

See *Spread (Bid Price & Ask Price)* for details on what the bid price is.

1.1.8.5 getStockPosition() Netscript Function

getStockPosition(*sym*)

Arguments

- **sym** (*string*) – Stock symbol

RAM cost 2 GB

Returns an array of four elements that represents the player's position in a stock.

The first element is the returned array is the number of shares the player owns of the stock in the Long position. The second element in the array is the average price of the player's shares in the Long position.

The third element in the array is the number of shares the player owns of the stock in the Short position. The fourth element in the array is the average price of the player's Short position.

All elements in the returned array are numeric.

Example:

```
pos = getStockPosition("ECP");
shares      = pos[0];
avgPx       = pos[1];
sharesShort = pos[2];
avgPxShort  = pos[3];
```

1.1.8.6 getStockMaxShares() Netscript Function

getStockMaxShares(*sym*)

Arguments

- **sym** (*string*) – Stock symbol

RAM cost 2 GB

Returns the maximum number of shares that the stock has. This is the maximum amount of the stock that can be purchased in both the Long and Short positions combined

1.1.8.7 getStockPurchaseCost() Netscript Function

getStockPurchaseCost(*sym*, *shares*, *posType*)

Arguments

- **sym** (*string*) – Stock symbol
- **shares** (*number*) – Number of shares to purchase
- **posType** (*string*) – Specifies whether the order is a "Long" or "Short" position. The values "L" or "S" can also be used.

RAM cost 2 GB

Calculates and returns how much it would cost to buy a given number of shares of a stock. This takes into account *spread* and commission fees.

1.1.8.8 getStockSaleGain() Netscript Function

getStockSaleGain(*sym*, *shares*, *posType*)

Arguments

- **sym** (*string*) – Stock symbol
- **shares** (*number*) – Number of shares to sell
- **posType** (*string*) – Specifies whether the order is a "Long" or "Short" position. The values "L" or "S" can also be used.

RAM cost 2 GB

Calculates and returns how much you would gain from selling a given number of shares of a stock. This takes into account *spread* and commission fees.

1.1.8.9 buyStock() Netscript Function

buyStock(*sym*, *shares*)

Arguments

- **sym** (*string*) – Symbol of stock to purchase
- **shares** (*number*) – Number of shares to purchased. Must be positive. Will be rounded to nearest integer

RAM cost 2.5 GB

Attempts to purchase shares of a stock using a Market Order.

If the player does not have enough money to purchase the specified number of shares, then no shares will be purchased. Remember that every transaction on the stock exchange costs a certain commission fee.

If this function successfully purchases the shares, it will return the stock price at which each share was purchased. Otherwise, it will return 0.

1.1.8.10 sellStock() Netscript Function

sellStock(*sym*, *shares*)

Arguments

- **sym** (*string*) – Symbol of stock to sell
- **shares** (*number*) – Number of shares to sell. Must be positive. Will be rounded to nearest integer

RAM cost 2.5 GB

Attempts to sell shares of a stock using a Market Order.

If the specified number of shares in the function exceeds the amount that the player actually owns, then this function will sell all owned shares. Remember that every transaction on the stock exchange costs a certain commission fee.

The net profit made from selling stocks with this function is reflected in the script's statistics. This net profit is calculated as:

```
shares * (sell price - average price of purchased shares)
```

If the sale is successful, this function will return the stock price at which each share was sold. Otherwise, it will return 0.

1.1.8.11 shortStock() Netscript Function

shortStock(*sym*, *shares*)

Arguments

- **sym** (*string*) – Symbol of stock to short

- **shares** (*number*) – Number of shares to short. Must be positive. Will be rounded to nearest integer

RAM cost 2.5 GB

Attempts to purchase a short position of a stock using a Market Order.

The ability to short a stock is **not** immediately available to the player and must be unlocked later on in the game.

If the player does not have enough money to purchase the specified number of shares, then no shares will be purchased. Remember that every transaction on the stock exchange costs a certain commission fee.

If the purchase is successful, this function will return the stock price at which each share was purchased. Otherwise, it will return 0.

1.1.8.12 sellShort() Netscript Function

sellShort(*sym*, *shares*)

Arguments

- **sym** (*string*) – Symbol of stock to sell
- **shares** (*number*) – Number of shares to sell. Must be positive. Will be rounded to nearest integer

RAM cost 2.5 GB

Attempts to sell a short position of a stock using a Market Order.

The ability to short a stock is **not** immediately available to the player and must be unlocked later on in the game.

If the specified number of shares exceeds the amount that the player actually owns, then this function will sell all owned shares. Remember that every transaction on the stock exchange costs a certain commission fee.

If the sale is successful, this function will return the stock price at which each share was sold. Otherwise it will return 0.

1.1.8.13 placeOrder() Netscript Function

placeOrder(*sym*, *shares*, *price*, *type*, *pos*)

Arguments

- **sym** (*string*) – Symbol of stock to player order for
- **shares** (*number*) – Number of shares for order. Must be positive. Will be rounded to nearest integer
- **price** (*number*) – Execution price for the order
- **type** (*string*) – Type of order. It must specify "limit" or "stop", and must also specify "buy" or "sell". This is NOT case-sensitive. Here are four examples that will work:
  - limitbuy
  - limitsell
  - stopbuy
  - stopsell
- **pos** (*string*) – Specifies whether the order is a "Long" or "Short" position. The Values "L" or "S" can also be used. This is NOT case-sensitive.

RAM cost 2.5 GB

Places an order on the stock market. This function only works for *Limit and Stop Orders*.

Returns true if the order is successfully placed, and false otherwise.

Note: The ability to place limit and stop orders is **not** immediately available to the player and must be unlocked later on in the game.

1.1.8.14 cancelOrder() Netscript Function

cancelOrder(*sym*, *shares*, *price*, *type*, *pos*)

Arguments

- **sym** (*string*) – Symbol of stock to player order for
- **shares** (*number*) – Number of shares for order. Must be positive. Will be rounded to nearest integer
- **price** (*number*) – Execution price for the order
- **type** (*string*) – Type of order. It must specify "limit" or "stop", and must also specify "buy" or "sell". This is NOT case-sensitive. Here are four examples that will work:
  - limitbuy
  - limitsell
  - stopbuy
  - stopsell
- **pos** (*string*) – Specifies whether the order is a "Long" or "Short" position. The Values "L" or "S" can also be used. This is NOT case-sensitive.

RAM cost 2.5 GB

Cancels an oustanding Limit or Stop order on the stock market.

The ability to use limit and stop orders is **not** immediately available to the player and must be unlocked later on in the game.

1.1.8.15 getOrders() Netscript Function

getOrders()

RAM cost 2.5 GB

Returns your order book for the stock market. This is an object containing information for all the *Limit and Stop Orders* you have in the stock market.

Note: This function isn't accessible until you have unlocked the ability to use Limit and Stop Orders.

The object has the following structure:

```
{
    StockSymbol1: [ // Array of orders for this stock
        {
            shares: Order quantity
            price: Order price
            type: Order type
            position: Either "L" or "S" for Long or Short position
        },
        {
            ...
        },
        ...
    ],
    StockSymbol2: [ // Array of orders for this stock
        ...
    ],
    ...
}
```

The "Order type" property can have one of the following four values:

- "Limit Buy Order"
- "Limit Sell Order"
- "Stop Buy Order"
- "Stop Sell Order"

Note that the order book will only contain information for stocks that you actually have orders in. For example, if you do not have orders in Nova Medical (NVMD), then the returned object will not have a "NVMD" property.

Example:

```
{
    ECP: [
        {
            shares: 5,
            price: 100,000
            type: "Stop Buy Order",
            position: "S",
        },
        {
            shares: 25,
            price: 125,000
            type: "Limit Sell Order",
            position: "L",
        },
    ],
    SYSC: [
        {
            shares: 100,
            price: 10,000
            type: "Limit Buy Order",
            position: "L",
        },
    ],
}
```

1.1.8.16 getStockVolatility() Netscript Function

getStockVolatility(*sym*)

Arguments

- **sym** (*string*) – Symbol of stock

RAM cost 2.5 GB

Returns the volatility of the specified stock.

Volatility represents the maximum percentage by which a stock's price can change every tick. The volatility is returned as a decimal value, NOT a percentage (e.g. if a stock has a volatility of 3%, then this function will return 0.03, NOT 3).

In order to use this function, you must first purchase access to the Four Sigma (4S) Market Data TIX API.

1.1.8.17 getStockForecast() Netscript Function

getStockForecast(*sym*)

Arguments

- **sym** (*string*) – Symbol of stock

RAM cost 2.5 GB

Returns the probability that the specified stock's price will increase (as opposed to decrease) during the next tick.

The probability is returned as a decimal value, NOT a percentage (e.g. if a stock has a 60% chance of increasing, then this function will return 0.6, NOT 60).

In other words, if this function returned 0.30 for a stock, then this means that the stock's price has a 30% chance of increasing and a 70% chance of decreasing during the next tick.

In order to use this function, you must first purchase access to the Four Sigma (4S) Market Data TIX API.

1.1.8.18 purchase4SMarketData() Netscript Function

purchase4SMarketData()

RAM cost 2.5 GB

Purchase 4S Market Data Access.

Returns true if you successfully purchased it or if you already have access. Returns false otherwise.

1.1.8.19 purchase4SMarketDataTixApi() Netscript Function

purchase4SMarketDataTixApi()

RAM cost 2.5 GB

Purchase 4S Market Data TIX API Access.

Returns true if you successfully purchased it or if you already have access. Returns false otherwise.

1.1.9 Netscript Singularity Functions

The Singularity Functions are a special set of Netscript functions. These functions allow you to control many additional aspects of the game through scripts, such as working for factions/companies, purchasing/installing Augmentations, and creating programs.

The Singularity Functions are **not** immediately available to the player and must be unlocked later in the game.

Warning: This page contains spoilers for the game

The Singularity Functions are unlocked in BitNode-4. If you are in BitNode-4, then you will automatically have access to all of these functions. You can use the Singularity Functions in other BitNodes if and only if you have the Source-File for BitNode-4 (aka Source-File 4). Each level of Source-File 4 will open up additional Singularity Functions that you can use in other BitNodes. If your Source-File 4 is upgraded all the way to level 3, then you will be able to access all of the Singularity Functions.

1.1.9.1 universityCourse() Netscript Function

universityCourse(*universityName*, *courseName*)

Arguments

- **universityName** (*string*) – Name of university. Not case-sensitive. You must be in the correct city for whatever university you specify.
  - Summit University
  - Rothman University
  - ZB Institute Of Technology
- **courseName** (*string*) – Name of course. Not case-sensitive.
  - Study Computer Science
  - Data Strucures
  - Networks
  - Algorithms
  - Management
  - Leadership

  If you are not in BitNode-4, then you must have Level 1 of Source-File 4 in order to use this function.

  This function will automatically set you to start taking a course at a university. If you are already in the middle of some "working" action (such as working at a company, for a faction, or on a program), then running this function will automatically cancel that action and give you your earnings.

  The cost and experience gains for all of these universities and classes are the same as if you were to manually visit and take these classes.

  This function will return true if you successfully start taking the course, and false otherwise.

1.1.9.2 gymWorkout() Netscript Function

gymWorkout(*gymName*, *stat*)

Arguments

- **gymName** (*string*) – Name of gym. Not case-sensitive. You must be in the correct city for whatever gym you specify.
  - Crush Fitness Gym
  - Snap Fitness Gym
  - Iron Gym
  - Powerhouse Gym
  - Millenium Fitness Gym
- **stat** (*string*) – The stat you want to train. Not case-sensitive.
  - strength OR str
  - defense OR def
  - dexterity OR dex
  - agility OR agi

  If you are not in BitNode-4, then you must have Level 1 of Source-File 4 in order to use this function.

  This function will automatically set you to start working out at a gym to train a particular stat. If you are already in the middle of some "working" action (such as working at a company, for a faction, or on a program), then running this function will automatically cancel that action and give you your earnings.

  The cost and experience gains for all of these gyms are the same as if you were to manually visit these gyms and train

  This function will return true if you successfully start working out at the gym, and false otherwise.

1.1.9.3 travelToCity() Netscript Function

travelToCity(*cityName*)

Arguments

- **cityName** (*string*) – City to travel to. CASE-SENSITIVE.
  - Aevum
  - Chongqing
  - Sector-12
  - New Tokyo
  - Ishima
  - Volhaven

If you are not in BitNode-4, then you must have Level 1 of Source-File 4 in order to use this function.

This function allows the player to travel to any city. The cost for using this function is the same as the cost for traveling through the Travel Agency.

This function will return true if you successfully travel to the specified city and false otherwise.

1.1.9.4 purchaseTor() Netscript Function

purchaseTor()

If you are not in BitNode-4, then you must have Level 1 of Source-File 4 in order to use this function.

This function allows you to automatically purchase a TOR router. The cost for purchasing a TOR router using this function is the same as if you were to manually purchase one.

This function will return true if it successfully purchase a TOR router and false otherwise.

1.1.9.5 purchaseProgram() Netscript Function

purchaseProgram(*programName*)

Arguments

- **programName** (*string*) – Name of program to purchase. Must include '.exe' extension. Not case-sensitive.

If you are not in BitNode-4, then you must have Level 1 of Source-File 4 in order to use this function.

This function allows you to automatically purchase programs. You MUST have a TOR router in order to use this function. The cost of purchasing programs using this function is the same as if you were purchasing them through the Dark Web using the Terminal *buy* command.

Example:

```
purchaseProgram("brutessh.exe");
```

This function will return true if the specified program is purchased, and false otherwise.

1.1.9.6 getStats() Netscript Function

getStats()

If you are not in BitNode-4, then you must have Level 1 of Source-File 4 in order to run this function.

Returns an object with the Player's stats. The object has the following properties:

```
{
    hacking
    strength
    defense
    dexterity
    agility
    charisma
    intelligence
}
```

Example:

```
res = getStats();
print('My charisma level is: ' + res.charisma);
```

1.1.9.7 getCharacterInformation() Netscript Function

getCharacterInformation()

If you are not in BitNode-4, then you must have Level 1 of Source-File 4 in order to run this function.

Returns an object with various information about your character. The object has the following properties:

```
{
    bitnode:                Current BitNode number
    city:                   Name of city you are currently in
    factions:               Array of factions you are currently a member of
    hp:                     Current health points
    jobs:                   Array of all companies at which you have jobs
    jobTitle:               Array of job positions for all companies you are␣
↪employed at. Same order as 'jobs'
    maxHp:                  Maximum health points
    tor:                    Boolean indicating whether or not you have a tor␣
↪router

    // The following is an object with many of the player's multipliers from␣
↪Augmentations/Source Files
    mult: {
        agility:            Agility stat
        agilityExp:         Agility exp
        companyRep:         Company reputation
        crimeMoney:         Money earned from crimes
        crimeSuccess:       Crime success chance
        defense:            Defense stat
        defenseExp:         Defense exp
        dexterity:          Dexterity stat
        dexterityExp:       Dexterity exp
        factionRep:         Faction reputation
        hacking:            Hacking stat
        hackingExp:         Hacking exp
        strength:           Strength stat
        strengthExp:        Strength exp
        workMoney:          Money earned from jobs
    },

    // The following apply only to when the character is performing
    // some type of working action, such as working for a company/faction
    timeWorked:             Timed worked in ms
    workHackExpGain:        Hacking experience earned so far from work
    workStrExpGain:         Str experience earned so far from work
    workDefExpGain:         Def experience earned so far from work
    workDexExpGain:         Dex experience earned so far from work
    workAgiExpGain:         Agi experience earned so far from work
    workChaExpGain:         Cha experience earned so far from work
    workRepGain:            Reputation earned so far from work, if applicable
    workMoneyGain:          Money earned so far from work, if applicable
}
```

1.1.9.8 isBusy() Netscript Function

isBusy()

If you are not in BitNode-4, then you must have Level 1 of Source-File 4 in order to run this function.

Returns a boolean indicating whether or not the player is currently performing an 'action'. These actions include:

- Working for a company/faction
- Studying at a univeristy
- Working out at a gym
- Creating a program
- Committing a crime
- Carrying out a Hacking Mission

1.1.9.9 stopAction() Netscript Function

stopAction()

If you are not in BitNode-4, then you must have Level 1 of Source-File 4 in order to run this function. This function is used to end whatever 'action' the player is currently performing. The player will receive whatever money/experience/etc. he has earned from that action.

The actions that can be stopped with this function are:

- Studying at a university
- Working for a company/faction
- Creating a program
- Committing a Crime

This function will return true if the player's action was ended. It will return false if the player was not performing an action when this function was called.

1.1.9.10 upgradeHomeRam() Netscript Function

upgradeHomeRam()

If you are not in BitNode-4, then you must have Level 2 of Source-File 4 in order to use this function.

This function will upgrade amount of RAM on the player's home computer. The cost is the same as if you were to do it manually.

This function will return true if the player's home computer RAM is successfully upgraded, and false otherwise.

1.1.9.11 getUpgradeHomeRamCost() Netscript Function

getUpgradeHomeRamCost()

If you are not in BitNode-4, then you must have Level 2 of Source-File 4 in order to use this function.

Returns the cost of upgrading the player's home computer RAM.

1.1.9.12 workForCompany() Netscript Function

workForCompany(*companyName=lastCompany*)

Arguments

- **companyName** (*string*) – Name of company to work for. Must be an exact match. Optional. If not specified, this argument defaults to the last job that you worked

If you are not in BitNode-4, then you must have Level 2 of Source-File 4 in order to use this function.

This function will automatically set you to start working at the company at which you are employed. If you are already in the middle of some "working" action (such as working for a faction, training at a gym, or creating a program), then running this function will automatically cancel that action and give you your earnings.

This function will return true if the player starts working, and false otherwise.

Note that when you are working for a company, you will not actually receive your earnings (reputation, money, experience) until you FINISH the action. This can be an issue if, for example, you only want to work until you get 100,000 company reputation. One small hack to get around this is to continuously restart the action to receive your earnings:

```
while (getCompanyRep(COMPANY HERE) < VALUE) {
    workForCompany();
    sleep(60000);
}
```

This way, your company reputation will be updated every minute.

1.1.9.13 applyToCompany() Netscript Function

applyToCompany(*companyName*, *field*)

Arguments

- **companyName** (*string*) – Name of company to apply to. CASE-SENSITIVE.
- **field** (*string*) – Field to which you want to apply. Not case-sensitive
  - software
  - software consultant
  - it
  - security engineer
  - network engineer
  - business
  - business consultant
  - security
  - agent
  - employee
  - part-time employee
  - waiter
  - part-time waiter

If you are not in BitNode-4, then you must have Level 2 of Source-File 4 in order to use this function.

This function will automatically try to apply to the specified company for a position in the specified field. This function can also be used to apply for promotions by specifying the company and field you are already employed at.

This function will return true if you successfully get a job/promotion, and false otherwise. Note that if you are trying to use this function to apply for a promotion and you don't get one, it will return false.

1.1.9.14 getCompanyRep() Netscript Function

getCompanyRep(*companyName*)

Arguments

- **companyName** (*string*) – Name of the company. CASE-SENSITIVE

If you are not in BitNode-4, then you must have Level 2 of Source-File 4 in order to use this function.

This function will return the amount of reputation you have at the specified company. If the company passed in as an argument is invalid, -1 will be returned.

1.1.9.15 getCompanyFavor() Netscript Function

getCompanyFavor(*companyName*)

Arguments

- **companyName** (*string*) – Name of the company. CASE-SENSITIVE

If you are not in BitNode-4, then you must have Level 2 of Source-File 4 in order to use this function.

This function will return the amount of favor you have at the specified company. If the company passed in as an argument is invalid, -1 will be returned.

1.1.9.16 getCompanyFavorGain() Netscript Function

getCompanyFavorGain(*companyName*)

Arguments

- **companyName** (*string*) – Name of the company. CASE-SENSITIVE

If you are not in BitNode-4, then you must have Level 2 of Source-File 4 in order to use this function.

This function will return the amount of favor you will gain for the specified company when you reset by installing Augmentations.

1.1.9.17 checkFactionInvitations() Netscript Function

checkFactionInvitations()

If you are not in BitNode-4, then you must have Level 2 of Source-File 4 in order to use this function.

Returns an array with the name of all Factions you currently have oustanding invitations from.

1.1.9.18 joinFaction() Netscript Function

joinFaction(*name*)

Arguments

- **name** (*string*) – Name of faction to join. CASE-SENSITIVE

If you are not in BitNode-4, then you must have Level 2 of Source-File 4 in order to use this function.

This function will automatically accept an invitation from a faction and join it.

1.1.9.19 workForFaction() Netscript Function

workForFaction(*factionName*, *workType*)

Arguments

- **factionName** (*string*) – Name of faction to work for. CASE-SENSITIVE
- **workType** (*string*) – Type of work to perform for the faction:
  - hacking/hacking contracts/hackingcontracts
  - field/fieldwork/field work
  - security/securitywork/security work

If you are not in BitNode-4, then you must have Level 2 of Source-File 4 in order to use this function.

This function will automatically set you to start working for the specified faction. Obviously, you must be a member of the faction or else this function will fail. If you are already in the middle of some "working" action (such as working for a company, training at a gym, or creating a program), then running this function will automatically cancel that action and give you your earnings.

This function will return true if you successfully start working for the specified faction, and false otherwise.

Note that when you are working for a faction, you will not actually receive your earnings (reputation, experience) until you FINISH the action. This can be an issue if, for example, you only want to work until you get 100,000 faction reputation. One small hack to get around this is to continuously restart the action to receive your earnings:

```
while (getFactionRep(FACTION NAME) < VALUE) {
    workForFaction(FACNAME, WORKTYPE);
    sleep(60000);
}
```

This way, your faction reputation will be updated every minute.

1.1.9.20 getFactionRep() Netscript Function

getFactionRep(*factionName*)

Arguments

- **factionName** (*string*) – Name of faction. CASE-SENSITIVE

If you are not in BitNode-4, then you must have Level 2 of Source-File 4 in order to use this function.

This function returns the amount of reputation you have for the specified faction.

1.1.9.21 getFactionFavor() Netscript Function

getFactionFavor (*factionName*)

Arguments

- **factionName** (*string*) – Name of faction. CASE-SENSITIVE

If you are not in BitNode-4, then you must have Level 2 of Source-File 4 in order to use this function.

This function returns the amount of favor you have for the specified faction.

1.1.9.22 getFactionFavorGain() Netscript Function

getFactionFavorGain (*factionName*)

Arguments

- **factionName** (*string*) – Name of faction. CASE-SENSITIVE

If you are not in BitNode-4, then you must have Level 2 of Source-File 4 in order to use this function.

This function returns the amount of favor you will gain for the specified faction when you reset by installing Augmentations.

1.1.9.23 donateToFaction() Netscript Function

donateToFaction (*factionName*, *donateAmt*)

Arguments

- **factionName** (*string*) – Name of faction to donate to. CASE-SENSITIVE
- **donateAmt** (*number*) – Amount of money to donate

If you are not in BitNode-4, then you must have Level 3 of Source-File 4 in order to use this function.

Attempts to donate money to the specified faction in exchange for reputation. Returns true if you successfully donate the money, and false otherwise.

1.1.9.24 createProgram() Netscript Function

createProgram (*programName*)

Arguments

- **programName** (*string*) – Name of program to create. Not case-sensitive

If you are not in BitNode-4, then you must have Level 3 of Source-File 4 in order to use this function.

This function will automatically set you to start working on creating the specified program. If you are already in the middle of some "working" action (such as working for a company, training at a gym, or taking a course), then running this function will automatically cancel that action and give you your earnings.

Example:

createProgram("relaysmtp.exe");

Note that creating a program using this function has the same hacking level requirements as it normally would. These level requirements are:

- BruteSSH.exe: 50

- FTPCrack.exe: 100
- relaySMTP.exe: 250
- HTTPWorm.exe: 500
- SQLInject.exe: 750
- DeepscanV1.exe: 75
- DeepscanV2.exe: 400
- ServerProfiler.exe: 75
- AutoLink.exe: 25

This function returns true if you successfully start working on the specified program, and false otherwise.

1.1.9.25 commitCrime() Netscript Function

commitCrime(*crime*)

Arguments

- **crime** (*string*) – Name of crime to attempt. Not case-sensitive. This argument is fairly lenient in terms of what inputs it accepts. Here is a list of valid inputs for all of the crimes:
  - shoplift
  - rob store
  - mug
  - larceny
  - deal drugs
  - bond forgery
  - traffick arms
  - homicide
  - grand theft auto
  - kidnap
  - assassinate
  - heist

If you are not in BitNode-4, then you must have Level 3 of Source-File 4 in order to use this function.

This function is used to automatically attempt to commit crimes. If you are already in the middle of some 'working' action (such as working for a company or training at a gym), then running this function will automatically cancel that action and give you your earnings.

This function returns the number of seconds it takes to attempt the specified crime (e.g It takes 60 seconds to attempt the 'Rob Store' crime, so running *commitCrime('rob store')* will return 60).

Warning: I do not recommend using the time returned from this function to try and schedule your crime attempts. Instead, I would use the isBusy() Singularity function to check whether you have finished attempting a crime. This is because although the game sets a certain crime to be X amount of seconds, there is no guarantee that your browser will follow that time limit.

1.1.9.26 getCrimeChance() Netscript Function

getCrimeChance(*crime*)

Arguments

- **crime** (*string*) – Name of crime. Not case-sensitive. This argument is fairlyn lenient in terms of what inputs it accepts. Check the documentation for the *commitCrime()* function for a list of example inputs.

If you are not in BitNode-4, then you must have Level 3 of Source-File 4 in order to use this function.

This function returns your chance of success at commiting the specified crime. The chance is returned as a decimal (i.e. 60% would be returned as 0.6).

1.1.9.27 getOwnedAugmentations() Netscript Function

getOwnedAugmentations(*purchased=false*)

Arguments

- **purchase** (*boolean*) – Specifies whether the returned array should include Augmentations you have purchased but not yet installed. By default, this argument is false which means that the return value will NOT have the purchased Augmentations.

If you are not in BitNode-4, then you must have Level 3 of Source-File 4 in order to use this function.

This function returns an array containing the names (as strings) of all Augmentations you have.

1.1.9.28 getOwnedSourceFiles() Netscript Function

getOwnedSourceFiles()

If you are not in BitNode-4, then you must have Level 3 of Source-File 4 in order to use this function.

Returns an array of source files [{n: 1, lvl: 3}, {n: 4, lvl: 3}]

1.1.9.29 getAugmentationsFromFaction() Netscript Function

getAugmentationsFromFaction(*facName*)

Arguments

- **facName** (*string*) – Name of faction. CASE-SENSITIVE

If you are not in BitNode-4, then you must have Level 3 of Source-File 4 in order to use this function.

Returns an array containing the names (as strings) of all Augmentations that are available from the specified faction.

1.1.9.30 getAugmentationPrereq() Netscript Function

getAugmentationPrereq(*augName*)

Arguments

- **augName** (*string*) – Name of Augmentation. CASE-SENSITIVE

If you are not in BitNode-4, then you must have Level 3 of Source-File 4 in order to use this function.

This function returns an array with the names of the prerequisite Augmentation(s) for the specified Augmentation. If there are no prerequisites, a blank array is returned.

If an invalid Augmentation name is passed in for the *augName* argument, this function will return a blank array.

1.1.9.31 getAugmentationCost() Netscript Function

getAugmentationCost(*augName*)

Arguments

- **augName** (*string*) – Name of Augmentation. CASE-SENSITIVE

If you are not in BitNode-4, then you must have Level 3 of Source-File 4 in order to use this function.

This function returns an array with two elements that gives the cost for the specified Augmentation. The first element in the returned array is the reputation requirement of the Augmentation, and the second element is the money cost.

If an invalid Augmentation name is passed in for the *augName* argument, this function will return the array [-1, -1].

1.1.9.32 purchaseAugmentation() Netscript Function

purchaseAugmentation(*factionName*, *augName*)

Arguments

- **factionName** (*string*) – Name of faction to purchase Augmentation from. CASE-SENSITIVE
- **augName** (*string*) – Name of Augmentation to purchase. CASE-SENSITIVE

If you are not in BitNode-4, then you must have Level 3 of Source-File 4 in order to use this function.

This function will try to purchase the specified Augmentation through the given Faction.

This function will return true if the Augmentation is successfully purchased, and false otherwise.

1.1.9.33 installAugmentations() Netscript Function

installAugmentations(*cbScript*)

Arguments

- **cbScript** (*string*) – Optional callback script. This is a script that will automatically be run after Augmentations are installed (after the reset). This script will be run with no arguments and 1 thread. It must be located on your home computer.

If you are not in BitNode-4, then you must have Level 3 of Source-File 4 in order to use this function.

This function will automatically install your Augmentations, resetting the game as usual.

This function will return false if it was not able to install Augmentations.

If this function successfully installs Augmentations, then it has no return value because all scripts are immediately terminated.

1.1.10 Netscript Bladeburner API

Netscript provides the following API for interacting with the game's Bladeburner mechanic.

The Bladeburner API is **not** immediately available to the player and must be unlocked later in the game

> **Warning:** This page contains spoilers for the game

The Bladeburner API is unlocked in BitNode-7. If you are in BitNode-7, you will automatically gain access to this API. Otherwise, you must have Source-File 7 in order to use this API in other BitNodes

Bladeburner API functions must be accessed through the 'bladeburner' namespace

In *Netscript 1.0*:

```
bladeburner.getContractNames();
bladeburner.startAction("general", "Training");
```

In *NetscriptJS (Netscript 2.0)*:

```
ns.bladeburner.getContractNames();
ns.bladeburner.startAction("general", "Training");
```

1.1.10.1 getContractNames() Netscript Function

getContractNames()
: Returns an array of strings containing the names of all Bladeburner contracts

1.1.10.2 getOperationNames() Netscript Function

getOperationNames()
: Returns an array of strings containing the names of all Bladeburner operations

1.1.10.3 getBlackOpNames() Netscript Function

getBlackOpNames()
: Returns an array of strings containing the names of all Bladeburner Black Ops

1.1.10.4 getGeneralActionNames() Netscript Function

getGeneralActionNames()
: Returns an array of strings containing the names of all general Bladeburner actions

1.1.10.5 getSkillNames() Netscript Function

getSkillNames()
: Returns an array of strings containing the names of all Bladeburner skills

1.1.10.6 startAction() Netscript Function

startAction(*type*, *name*)

Arguments

- **type** (*string*) – Type of action. See *Bladeburner Action Types*
- **name** (*string*) – Name of action. Must be an exact match

Attempts to start the specified Bladeburner action. Returns true if the action was started successfully, and false otherwise.

1.1.10.7 stopBladeburnerAction() Netscript Function

stopBladeburnerAction()
Stops the current Bladeburner action

1.1.10.8 getCurrentAction() Netscript Function

getCurrentAction()
Returns an object that represents the player's current Bladeburner action:

```
{
    type: Type of Action
    name: Name of Action
}
```

If the player is not performing an action, the function will return an object with the 'type' property set to "Idle".

1.1.10.9 getActionTime() Netscript Function

getActionTime(*type*, *name*)

Arguments

- **type** (*string*) – Type of action. See *Bladeburner Action Types*
- **name** (*string*) – Name of action. Must be an exact match

Returns the number of seconds it takes to complete the specified action

1.1.10.10 getActionEstimatedSuccessChance() Netscript Function

getActionEstimatedSuccessChance(*type*, *name*)

Arguments

- **type** (*string*) – Type of action. See *Bladeburner Action Types*
- **name** (*string*) – Name of action. Must be an exact match

Returns the estimated success chance for the specified action. This chance is returned as a decimal value, NOT a percentage (e.g. if you have an estimated success chance of 80%, then this function will return 0.80, NOT 80).

1.1.10.11 getActionRepGain() Netscript Function

getActionRepGain (*type*, *name*[, *level=current level*])

Arguments

- **type** (*string*) – Type of action. See *Bladeburner Action Types*
- **name** (*string*) – Name of action. Must be an exact match
- **level** (*number*) – Optional action level at which to calculate the gain

Returns the average Bladeburner reputation gain for successfully completing the specified action. Note that this value is an 'average' and the real reputation gain may vary slightly from this value.

1.1.10.12 getActionCountRemaining() Netscript Function

getActionCountRemaining (*type*, *name*)

Arguments

- **type** (*string*) – Type of action. See *Bladeburner Action Types*
- **name** (*string*) – Name of action. Must be an exact match

Returns the remaining count of the specified action.

Note that this is meant to be used for Contracts and Operations. This function will return 'Infinity' for actions such as Training and Field Analysis. This function will return 1 for BlackOps not yet completed regardless of wether the player has the required rank to attempt the mission or not.

1.1.10.13 getActionMaxLevel() Netscript Function

getActionMaxLevel (*type*, *name*)

Arguments

- **type** (*string*) – Type of action. See *Bladeburner Action Types*
- **name** (*string*) – Name of action. Must be an exact match

Returns the maximum level for this action.

Returns -1 if an invalid action is specified.

1.1.10.14 getActionCurrentLevel() Netscript Function

getActionCurrentLevel (*type*, *name*)

Arguments

- **type** (*string*) – Type of action. See *Bladeburner Action Types*
- **name** (*string*) – Name of action. Must be an exact match

Returns the current level of this action.

Returns -1 if an invalid action is specified.

1.1.10.15 getActionAutolevel() Netscript Function

getActionAutolevel(*type*, *name*)

Arguments

- **type** (*string*) – Type of action. See *Bladeburner Action Types*
- **name** (*string*) – Name of action. Must be an exact match

Return a boolean indicating whether or not this action is currently set to autolevel.

Returns false if an invalid action is specified.

1.1.10.16 setActionAutolevel() Netscript Function

setActionAutolevel(*type*, *name*, *autoLevel*)

Arguments

- **type** (*string*) – Type of action. See *Bladeburner Action Types*
- **name** (*string*) – Name of action. Must be an exact match
- **autoLevel** (*boolean*) – Whether or not to autolevel this action

Enable/disable autoleveling for the specified action.

1.1.10.17 setActionLevel() Netscript Function

setActionLevel(*type*, *name*, *level*)

Arguments

- **type** (*string*) – Type of action. See *Bladeburner Action Types*
- **name** (*string*) – Name of action. Must be an exact match
- **int** (*level*) – Level to set this action to

Set the level for the specified action.

1.1.10.18 getRank() Netscript Function

getRank()

Returns the player's Bladeburner Rank

1.1.10.19 getBlackOpRank() Netscript Function

getBlackOpRank(*name*)

Arguments

- **name** (*string*) – name of the BlackOp. Must be an exact match.

Returns the rank required to complete this BlackOp.

Returns -1 if an invalid action is specified.

1.1.10.20 getSkillPoints() Netscript Function

getSkillPoints()
Returns the number of Bladeburner skill points you have

1.1.10.21 getSkillLevel() Netscript Function

getSkillLevel(*skillName=""*)

Arguments

- **skillName** (*string*) – Name of skill. Case-sensitive and must be an exact match

This function returns your level in the specified skill.

The function returns -1 if an invalid skill name is passed in

1.1.10.22 getSkillUpgradeCost() Netscript Function

getSkillUpgradeCost(*skillName=""*)

Arguments

- **skillName** (*string*) – Name of skill. Case-sensitive and must be an exact match

This function returns the number of skill points needed to upgrade the specified skill.

The function returns -1 if an invalid skill name is passed in.

1.1.10.23 upgradeSkill() Netscript Function

upgradeSkill(*skillName*)

Arguments

- **skillName** (*string*) – Name of Skill to be upgraded. Case-sensitive and must be an exact match

Attempts to upgrade the specified Bladeburner skill. Returns true if the skill is successfully upgraded, and false otherwise

1.1.10.24 getTeamSize() Netscript Function

getTeamSize(*type*, *name*)

Arguments

- **type** (*string*) – Type of action. See *Bladeburner Action Types*
- **name** (*string*) – Name of action. Must be an exact match

Returns the number of Bladeburner team members you have assigned to the specified action.

Setting a team is only applicable for Operations and BlackOps. This function will return 0 for other action types.

1.1.10.25 setTeamSize() Netscript Function

setTeamSize(*type*, *name*, *size*)

Arguments

- **type** (*string*) – Type of action. See *Bladeburner Action Types*
- **name** (*string*) – Name of action. Must be an exact match
- **size** (*int*) – Number of team members to set. Will be converted using Math.round()

Set the team size for the specified Bladeburner action.

Returns the team size that was set, or -1 if the function failed.

1.1.10.26 getCityEstimatedPopulation() Netscript Function

getCityEstimatedPopulation(*cityName*)

Arguments

- **cityName** (*string*) – Name of city. Case-sensitive

Returns the estimated number of Synthoids in the specified city, or -1 if an invalid city was specified.

1.1.10.27 getCityEstimatedCommunities() Netscript Function

getCityEstimatedCommunities(*cityName*)

Arguments

- **cityName** (*string*) – Name of city. Case-sensitive

Returns the estimated number of Synthoid communities in the specified city, or -1 if an invalid city was specified.

1.1.10.28 getCityChaos() Netscript Function

getCityChaos(*cityName*)

Arguments

- **cityName** (*string*) – Name of city. Case-sensitive

Returns the chaos in the specified city, or -1 if an invalid city was specified

1.1.10.29 getCity() Netscript Function

getCity()

Returns the city that the player is currently in (for Bladeburner).

1.1.10.30 switchCity() Netscript Function

switchCity(*cityName*)

Arguments

- **cityName** (*string*) – Name of city

Attempts to switch to the specified city (for Bladeburner only).

Returns true if successful, and false otherwise

1.1.10.31 getStamina() Netscript Function

getStamina()

Returns an array with two elements:

[Current stamina, Max stamina]

Example usage:

```
function getStaminaPercentage() {
    let res = bladeburner.getStamina();
    return res[0] / res[1];
}
```

1.1.10.32 joinBladeburnerFaction() Netscript Function

joinBladeburnerFaction()

Attempts to join the Bladeburner faction.

Returns true if you successfully join the Bladeburner faction, or if you are already a member.

Returns false otherwise.

1.1.10.33 joinBladeburnerDivision() Netscript Function

joinBladeburnerDivision()

Attempts to join the Bladeburner division.

Returns true if you successfully join the Bladeburner division, or if you are already a member.

Returns false otherwise

1.1.10.34 getBonusTime() Netscript Function

getBonusTime()

Returns the amount of accumulated "bonus time" (seconds) for the Bladeburner mechanic.

"Bonus time" is accumulated when the game is offline or if the game is inactive in the browser.

"Bonus time" makes the game progress faster, up to 5x the normal speed. For example, if an action takes 30 seconds to complete but you've accumulated over 30 seconds in bonus time, then the action will only take 6 seconds in real life to complete.

1.1.10.35 Bladeburner Action Types

Several functions in the Bladeburner API require you to specify an action using its type and name. The following are valid values when specifying the action's type:

Contracts

- contract

- contracts
- contr

Operations

- operation
- operations
- op
- ops

Black Ops

- blackoperation
- black operation
- black operations
- black op
- black ops
- blackop
- blackops

General Actions (Training, Field Analysis, Recruitment)

- general
- general action
- gen

1.1.10.36 Examples

Basic example usage:

```
tprint(bladeburner.getContractNames());
tprint(bladeburner.getOperationNames());
tprint(bladeburner.getBlackOpNames());
tprint(bladeburner.getGeneralActionNames());
tprint(bladeburner.getSkillNames());
tprint(bladeburner.getActionTime("contract", "Tracking"));
tprint("Rank: " + bladeburner.getRank());
tprint("Skill Points: " + bladeburner.getSkillPoints());
tprint("Cloak Skill Level: " + bladeburner.getSkillLevel("Cloak"));
tprint("Trying to upgradeSkill: " + bladeburner.upgradeSkill("Cloak"));
tprint("Skill Points remaining: " + bladeburner.getSkillPoints());

tprint("Trying to switch to a nonexistent city: " + bladeburner.switchCity("lskgns"));

var chongqing = "Chongqing";
tprint("Trying to switch to Chongqing: " + bladeburner.switchCity(chongqing));
tprint("Chongqing chaos: " + bladeburner.getCityChaos(chongqing));
tprint("Chongqing estimated pop: " + bladeburner.
↪getCityEstimatedPopulation(chongqing));
tprint("Chongqing estimated communities: " + bladeburner.
↪getCityEstimatedCommunities(chongqing));
```

Bladeburner handler example. Note that this avoids the need of using the *bladeburner* namespace identifier by attaching the Bladeburner API functions to an object:

```
const FIELD_ANALYSIS_INTERVAL = 10; //Number of minutes between field analysis states
const FIELD_ANALYSIS_DURATION = 5;  //Duration in minutes

function BladeburnerHandler(ns, params) {
    //Netscript environment becomes part of the instance
    this.ns = ns;

    //Netscript bladeburner API becomes part of this instance
    for (var bladeburnerFn in ns.bladeburner) {
        this[bladeburnerFn] = ns.bladeburner[bladeburnerFn];
    }

    this.fieldAnalysis = {
        inProgress:         params.startFieldAnalysis ? true : false,
        cyclesRemaining:    params.startFieldAnalysis ? FIELD_ANALYSIS_DURATION : 0,
        cyclesSince:        params.startFieldAnalysis ? FIELD_ANALYSIS_INTERVAL : 0,
    }
}



BladeburnerHandler.prototype.getStaminaPercentage = function() {
    var res = this.getStamina();
    return 100 * (res[0] / res[1]);
}

BladeburnerHandler.prototype.hasSimulacrum = function() {
    var augs = this.ns.getOwnedAugmentations();
    return augs.includes("The Blade's Simulacrum");
}

BladeburnerHandler.prototype.handle = function() {
    //If we're doing something else manually (without Simlacrum),
    //it overrides Bladeburner stuff
    if (!this.hasSimulacrum() && this.ns.isBusy()) {
        this.ns.print("Idling bc player is busy with some other action");
        return;
    }

    if (this.fieldAnalysis.inProgress) {
        --(this.fieldAnalysis.cyclesRemaining);
        if (this.fieldAnalysis.cyclesRemaining < 0) {
            this.fieldAnalysis.inProgress = false;
            this.fieldAnalysis.cyclesSince = 0;
            return this.handle();
        } else {
            this.startAction("general", "Field Analysis");
            this.ns.print("handler is doing field analyis for " +
                          (this.fieldAnalysis.cyclesRemaining+1) + " more mins");
            return 31; //Field Analysis Time + 1
        }
    } else {
        ++(this.fieldAnalysis.cyclesSince);
        if (this.fieldAnalysis.cyclesSince > FIELD_ANALYSIS_INTERVAL) {
            this.fieldAnalysis.inProgress = true;
```

(continues on next page)

(continued from previous page)

```
            this.fieldAnalysis.cyclesRemaining = FIELD_ANALYSIS_DURATION;
            return this.handle();
        }
    }

    this.stopBladeburnerAction();

    var staminaPerc = this.getStaminaPercentage();
    if (staminaPerc < 55) {
        this.ns.print("handler is starting training due to low stamina percentage");
        this.startAction("general", "Training");
        return 31; //Training time + 1
    } else {
        var action = this.chooseAction();
        this.ns.print("handler chose " + action.name + " " + action.type + " through␣
↪chooseAction()");
        this.startAction(action.type, action.name);
        return (this.getActionTime(action.type, action.name) + 1);
    }
}

BladeburnerHandler.prototype.chooseAction = function() {
    //Array of all Operations
    var ops = this.getOperationNames();

    //Sort Operations in order of increasing success chance
    ops.sort((a, b)=>{
        return this.getActionEstimatedSuccessChance("operation", a) -
               this.getActionEstimatedSuccessChance("operation", b);
    });

    //Loop through until you find one with 99+% success chance
    for (let i = 0; i < ops.length; ++i) {
        let successChance   = this.getActionEstimatedSuccessChance("operation",␣
↪ops[i]);
        let count           = this.getActionCountRemaining("operation", ops[i]);
        if (successChance >= 0.99 && count > 10) {
            return {type: "operation", name: ops[i]};
        }
    }

    //Repeat for Contracts
    var contracts = this.getContractNames();
    contracts.sort((a, b)=>{
        return this.getActionEstimatedSuccessChance("contract", a) -
               this.getActionEstimatedSuccessChance("contract", b);
    });

    for (let i = 0; i < contracts.length; ++i) {
        let successChance   = this.getActionEstimatedSuccessChance("contract",␣
↪contracts[i]);
        let count           = this.getActionCountRemaining("contract", contracts[i]);
        if (successChance >= 0.80 && count > 10) {
            return {type: "contract", name: contracts[i]};
        }
    }
```

(continues on next page)

(continued from previous page)

```
    return {type:"general", name:"Training"};
}


BladeburnerHandler.prototype.process = async function() {
    await this.ns.sleep(this.handle() * 1000);
}

export async function main(ns) {
    //Check if Bladeburner is available. This'll throw a runtime error if it's not
    ns.bladeburner.getContractNames();

    var startFieldAnalysis = true;
    if (ns.args.length >= 1 && ns.args[0] == "false") {
        startFieldAnalysis = false;
    }

    var handler = new BladeburnerHandler(ns, {
        startFieldAnalysis: startFieldAnalysis
    });
    while(true) {
        await handler.process();
    }
}
```

1.1.11 Netscript Gang API

Netscript provides the following API for interacting with the game's Gang mechanic.

The Gang API is **not** immediately available to the player and must be unlocked later in the game

> **Warning:** This page contains spoilers for the game

The Gang mechanic and the Gang API are unlocked in BitNode-2.

Gang API functions must be accessed through the 'gang' namespace

In *Netscript 1.0*:

```
gang.getMemberNames();
gang.recruitMember("Fry");
```

In *NetscriptJS (Netscript 2.0)*:

```
ns.gang.getMemberNames();
ns.gang.recruitMember("Fry");
```

1.1.11.1 getMemberNames() Netscript Function

getMemberNames()

Get the names of all Gang members

Returns An array of the names of all Gang members as strings

1.1.11.2 getGangInformation() Netscript Function

getGangInformation()

Get general information about the gang

Returns An object with the gang information.

The object has the following structure:

```
{
    faction:                Name of faction that the gang belongs to ("Slum Snakes
↪", etc.)
    isHacking:              Boolean indicating whether or not its a hacking gang
    moneyGainRate:          Money earned per second
    power:                  Gang's power for territory warfare
    respect:                Gang's respect
    respectGainRate:        Respect earned per second
    territory:              Amount of territory held. Returned in decimal form,␣
↪not percentage
    territoryClashChance:   Clash chance. Returned in decimal form, not percentage
    wantedLevel:            Gang's wanted level
    wantedLevelGainRate:    Wanted level gained/lost per second (negative for␣
↪losses)
}
```

1.1.11.3 getOtherGangInformation() Netscript Function

getOtherGangInformation()

Get territory and power information about all gangs

Returns An object with information about all gangs

The object has the following structure:

```
{
    "Slum Snakes" : {
        power: Slum Snakes' power
        territory: Slum Snakes' territory, in decimal form
    },
    "Tetrads" : {
        power: ...
        territory: ...
    },
    "The Syndicate" : {
        power: ...
        territory: ...
    },
    ... (for all six gangs)
}
```

1.1.11.4 getMemberInformation() Netscript Function

getMemberInformation(*name*)

Arguments

- **name** (*string*) – Name of member

Get stat and equipment-related information about a Gang Member

Returns An object with the gang member information.

The object has the following structure:

```
{
    agility:                Agility stat
    agilityEquipMult:       Agility multiplier from equipment. Decimal form
    agilityAscensionMult:   Agility multiplier from ascension. Decimal form
    augmentations:          Array of names of all owned Augmentations
    charisma:               Charisma stat
    charismaEquipMult:      Charisma multiplier from equipment. Decimal form
    charismaAscensionMult:  Charisma multiplier from ascension. Decimal form
    defense:                Defense stat
    defenseEquipMult:       Defense multiplier from equipment. Decimal form
    defenseAscensionMult:   Defense multiplier from ascension. Decimal form
    dexterity:              Dexterity stat
    dexterityEquipMult:     Dexterity multiplier from equipment. Decimal form
    dexterityAscensionMult: Dexterity multiplier from ascension. Decimal form
    equipment:              Array of names of all owned Non-Augmentation Equipment
    hacking:                Hacking stat
    hackingEquipMult:       Hacking multiplier from equipment. Decimal form
    hackingAscensionMult:   Hacking multiplier from ascension. Decimal form
    strength:               Strength stat
    strengthEquipMult:      Strength multiplier from equipment. Decimal form
    strengthAscensionMult:  Strength multiplier from ascension. Decimal form
    task:                   Name of currently assigned task
}
```

1.1.11.5 canRecruitMember() Netscript Function

canRecruitMember()

Returns Boolean indicating whether a member can currently be recruited

1.1.11.6 recruitMember() Netscript Function

recruitMember(*name*)

Arguments

- **name** (*string*) – Name of member to recruit

Attempt to recruit a new gang member.

Possible reasons for failure:

- Cannot currently recruit a new member
- There already exists a member with the specified name

Returns True if the member was successfully recruited. False otherwise

1.1.11.7 getTaskNames() Netscript Function

getTaskNames()

Get the name of all valid tasks that Gang members can be assigned to

Returns Array of strings of all task names

1.1.11.8 setMemberTask() Netscript Function

setMemberTask(*memberName*, *taskName*)

Arguments

- **memberName** (*string*) – Name of Gang member to assign
- **taskName** (*string*) – Task to assign

Attempts to assign the specified Gang Member to the specified task. If an invalid task is specified, the Gang member will be set to idle ("Unassigned")

Returns True if the Gang Member was successfully assigned to the task. False otherwise

1.1.11.9 getEquipmentNames() Netscript Function

getEquipmentNames()

Get the name of all possible equipment/upgrades you can purchase for your Gang Members. This includes Augmentations.

Returns Array of strings of the names of all Equpiment/Augmentations

1.1.11.10 getEquipmentCost() Netscript Function

getEquipmentCost(*equipName*)

Arguments

- **equipName** (*string*) – Name of equipment

Get the amount of money it takes to purchase a piece of Equipment or an Augmentation. If an invalid Equipment/Augmentation is specified, this function will return Infinity.

Returns Cost to purchase the specified Equipment/Augmentation (number). Infinity for invalid arguments

1.1.11.11 getEquipmentType() Netscript Function

getEquipmentType(*equipName*)

Arguments

- **equipName** (*string*) – Name of equipment

Get the specified equipment type, which can be one of the following:

- Weapon
- Armor
- Vehicle
- Rootkit
- Augmentation

Returns A string stating the type of the equipment

1.1.11.12 purchaseEquipment() Netscript Function

purchaseEquipment(*memberName*, *equipName*)

Arguments

- **memberName** (*string*) – Name of Gang member to purchase the equipment for
- **equipName** (*string*) – Name of Equipment/Augmentation to purchase

Attempt to purchase the specified Equipment/Augmentation for the specified Gang member.

Returns True if the equipment was successfully purchased. False otherwise

1.1.11.13 ascendMember() Netscript Function

ascendMember(*name*)

Arguments

- **name** (*string*) – Name of member to ascend

Ascend the specified Gang Member.

Returns An object with info about the ascension results.

The object has the following structure:

```
{
    respect:    Amount of respect lost from ascending
    hack:       Hacking multiplier gained from ascending. Decimal form
    str:        Strength multiplier gained from ascending. Decimal form
    def:        Defense multiplier gained from ascending. Decimal form
    dex:        Dexterity multiplier gained from ascending. Decimal form
    agi:        Agility multiplier gained from ascending. Decimal form
    cha:        Charisma multiplier gained from ascending. Decimal form
}
```

1.1.11.14 setTerritoryWarfare() Netscript Function

setTerritoryWarfare(*engage*)

Arguments

- **engage** (*bool*) – Whether or not to engage in territory warfare

Set whether or not the gang should engage in territory warfare

1.1.11.15 getChanceToWinClash() Netscript Function

getChanceToWinClash(*gangName*)

Arguments

- **gangName** (*string*) – Target gang

Returns the chance you have to win a clash with the specified gang. The chance is returned in decimal form, not percentage

1.1.11.16 getBonusTime() Netscript Function

getBonusTime()

Returns the amount of accumulated "bonus time" (seconds) for the Gang mechanic.

"Bonus time" is accumulated when the game is offline or if the game is inactive in the browser.

"Bonus time" makes the game progress faster, up to 10x the normal speed.

Returns Bonus time for the Gang mechanic in seconds

1.1.12 Netscript Coding Contract API

Netscript provides the following API for interacting with *Coding Contracts*.

The Coding Contract API must be accessed through the 'codingcontract' namespace

In *Netscript 1.0*:

```
codingcontract.getDescription("foo.cct", "home");
codingcontract.attempt(1, "foo.cct", "foodnstuff");
```

In *NetscriptJS (Netscript 2.0)*:

```
ns.codingcontract.getDescription("foo.cct", "home");
ns.codingcontract.attempt(1, "foo.cct", "foodnstuff");
```

1.1.12.1 attempt() Netscript Function

attempt(*answer*, *fn*[, *hostname/ip=current ip*, *opts={}*])

Arguments

- **answer** – Solution for the contract
- **fn** (*string*) – Filename of the contract
- **hostname/ip** (*string*) – Hostname or IP of the server containing the contract. Optional. Defaults to current server if not provided
- **opts** (*object*) – Optional parameters for configuring function behavior. Properties:
  - returnReward (*boolean*) If truthy, then the function will return a string that states the contract's reward when it is successfully solved.

Attempts to solve the Coding Contract with the provided solution.

Returns Boolean indicating whether the solution was correct. If the `returnReward` option is configured, then the function will instead return a string. If the contract is successfully solved, the string will contain a description of the contract's reward. Otherwise, it will be an empty string.

1.1.12.2 getContractType() Netscript Function

getContractType(*fn*[, *hostname/ip=current ip*])

Arguments

- **fn** (*string*) – Filename of the contract

- **hostname/ip** (*string*) – Hostname or IP of the server containing the contract. Optional. Defaults to current server if not provided

Returns a name describing the type of problem posed by the Coding Contract. (e.g. Find Largest Prime Factor, Total Ways to Sum, etc.)

Returns A string with the contract's problem type

1.1.12.3 getDescription() Netscript Function

getDescription(*fn*[, *hostname/ip=current ip*])

Arguments

- **fn** (*string*) – Filename of the contract
- **hostname/ip** (*string*) – Hostname or IP of the server containing the contract. Optional. Defaults to current server if not provided

Get the full text description for the problem posed by the Coding Contract

Returns A string with the contract's text description

1.1.12.4 getData() Netscript Function

getData(*fn*[, *hostname/ip=current ip*])

Arguments

- **fn** (*string*) – Filename of the contract
- **hostname/ip** (*string*) – Hostname or IP of the server containing the contract. Optional. Defaults to current server if not provided

Get the data associated with the specific Coding Contract. Note that this is not the same as the contract's description. This is just the data that the contract wants you to act on in order to solve

Returns The specified contract's data

1.1.12.5 getNumTriesRemaining() Netscript Function

getNumTriesRemaining(*fn*[, *hostname/ip=current ip*])

Arguments

- **fn** (*string*) – Filename of the contract
- **hostname/ip** (*string*) – Hostname or IP of the server containing the contract. Optional. Defaults to current server if not provided

Get the number of tries remaining on the contract before it self-destructs.

Returns Number indicating how many attempts are remaining

1.1.13 Netscript Sleeve API

Netscript provides the following API for interacting with the game's *Duplicate Sleeve* mechanic.

The Sleeve API is **not** immediately available to the player and must be unlocked later in the game.

> **Warning:** This page contains spoilers for the game

The Sleeve API is unlocked in BitNode-10. If you are in BitNode-10, you will automatically gain access to this API. Otherwise, you must have Source-File 10 in order to use this API in other BitNodes

Sleeve API functions must be accessed through the 'sleeve' namespace

In *Netscript 1.0*:

```
sleeve.synchronize(0);
sleeve.commitCrime(0, "shoplift");
```

In *NetscriptJS (Netscript 2.0)*:

```
ns.sleeve.synchronize(0);
ns.sleeve.commitCrime(0, "shoplift");
```

1.1.13.1 getNumSleeves() Netscript Function

getNumSleeves()
Return the number of duplicate sleeves the player has.

1.1.13.2 getSleeveStats() Netscript Function

getSleeveStats(*sleeveNumber*)

Arguments

- **sleeveNumber** (*int*) – Index of the sleeve to get stats of. See *here*

Return a structure containing the stats of the sleeve

```
{
    shock: current shock of the sleeve [0-100],
    sync: current sync of the sleeve [0-100],
    hacking_skill: current hacking skill of the sleeve,
    strength: current strength of the sleeve,
    defense: current defense of the sleeve,
    dexterity: current dexterity of the sleeve,
    agility: current agility of the sleeve,
    charisma: current charisma of the sleeve,
}
```

1.1.13.3 getInformation() Netscript Function

getInformation(*sleeveNumber*)

Arguments

- **sleeveNumber** (*int*) – Index of the sleeve to retrieve information. See *here*

Return a struct containing tons of information about this sleeve

```
{
    city:     location of the sleeve,
    hp:       current hp of the sleeve,
    maxHp:    max hp of the sleeve,
    jobs:     jobs available to the sleeve,
    jobTitle: job titles available to the sleeve,
    tor:      does this sleeve have access to the tor router,
    mult: {
        agility:      agility multiplier,
        agilityExp:   agility exp multiplier,
        companyRep:   company reputation multiplier,
        crimeMoney:   crime money multiplier,
        crimeSuccess: crime success chance multiplier,
        defense:      defense multiplier,
        defenseExp:   defense exp multiplier,
        dexterity:    dexterity multiplier,
        dexterityExp: dexterity exp multiplier,
        factionRep:   faction reputation multiplier,
        hacking:      hacking skill multiplier,
        hackingExp:   hacking exp multiplier,
        strength:     strength multiplier,
        strengthExp:  strength exp multiplier,
        workMoney:    work money multiplier,
    },
    timeWorked: time spent on the current task in milliseconds,
    earningsForSleeves : { earnings synchronized to other sleeves
            workHackExpGain: hacking exp gained from work,
            workStrExpGain:  strength exp gained from work,
            workDefExpGain:  defense exp gained from work,
            workDexExpGain:  dexterity exp gained from work,
            workAgiExpGain:  agility exp gained from work,
            workChaExpGain:  charisma exp gained from work,
            workMoneyGain:   money gained from work,
    },
    earningsForPlayer : { earnings synchronized to the player
            workHackExpGain: hacking exp gained from work,
            workStrExpGain:  strength exp gained from work,
            workDefExpGain:  defense exp gained from work,
            workDexExpGain:  dexterity exp gained from work,
            workAgiExpGain:  agility exp gained from work,
            workChaExpGain:  charisma exp gained from work,
            workMoneyGain:   money gained from work,
    },
    earningsForTask : { earnings for this sleeve
            workHackExpGain: hacking exp gained from work,
            workStrExpGain:  strength exp gained from work,
            workDefExpGain:  defense exp gained from work,
            workDexExpGain:  dexterity exp gained from work,
            workAgiExpGain:  agility exp gained from work,
            workChaExpGain:  charisma exp gained from work,
            workMoneyGain:   money gained from work,
    },
    workRepGain: Reputation gain rate when working for factions or companies
}
```

1.1.13.4 getTask() Netscript Function

getTask(*sleeveNumber*)

Arguments

- **sleeveNumber** (*int*) – Index of the sleeve to retrieve task from. See *here*

Return the current task that the sleeve is performing. type is set to "Idle" if the sleeve isn't doing anything

```
{
    task:            string, // task type
    crime:           string, // crime currently attempting, if any
    location:        string, // location of the task, if any
    gymStatType:     string, // stat being trained at the gym, if any
    factionWorkType: string, // faction work type being performed, if any
}
```

1.1.13.5 setToShockRecovery() Netscript Function

setToShockRecovery(*sleeveNumber*)

Arguments

- **sleeveNumber** (*int*) – Index of the sleeve to start recovery. See *here*

Return a boolean indicating whether or not this action was set successfully.

1.1.13.6 setToSynchronize() Netscript Function

setToSynchronize(*sleeveNumber*)

Arguments

- **sleeveNumber** (*int*) – Index of the sleeve to start synchronizing. See *here*

Return a boolean indicating whether or not this action was set successfully.

1.1.13.7 setToCommitCrime() Netscript Function

setToCommitCrime(*sleeveNumber*, *name*)

Arguments

- **sleeveNumber** (*int*) – Index of the sleeve to start commiting crime. See *here*
- **name** (*string*) – Name of the crime. Must be an exact match.

Return a boolean indicating whether or not this action was set successfully.

Returns false if an invalid action is specified.

1.1.13.8 setToFactionWork() Netscript Function

setToFactionWork(*sleeveNumber*, *factionName*, *factionWorkType*)

Arguments

- **sleeveNumber** (*int*) – Index of the sleeve to work for the faction. See *here*

- **factionName** (*string*) – Name of the faction to work for.
- **factionWorkType** (*string*) – Name of the action to perform for this faction.

Return a boolean indicating whether or not the sleeve started working or this faction.

1.1.13.9 setToCompanyWork() Netscript Function

setToCompanyWork(*sleeveNumber*, *companyName*)

Arguments

- **sleeveNumber** (*int*) – Index of the sleeve to work for the company. See *here*
- **companyName** (*string*) – Name of the company to work for.

Return a boolean indicating whether or not the sleeve started working or this company.

1.1.13.10 setToUniversityCourse() Netscript Function

setToUniversityCourse(*sleeveNumber*, *university*, *className*)

Arguments

- **sleeveNumber** (*int*) – Index of the sleeve to start taking class. See *here*
- **university** (*string*) – Name of the university to attend.
- **className** (*string*) – Name of the class to follow.

Return a boolean indicating whether or not this action was set successfully.

1.1.13.11 setToGymWorkout() Netscript Function

setToGymWorkout(*sleeveNumber*, *gymName*, *stat*)

Arguments

- **sleeveNumber** (*int*) – Index of the sleeve to workout at the gym. See *here*
- **gymName** (*string*) – Name of the gym.
- **stat** (*string*) – Name of the stat to train.

Return a boolean indicating whether or not the sleeve started working out.

1.1.13.12 travel() Netscript Function

travel(*sleeveNumber*, *cityName*)

Arguments

- **sleeveNumber** (*int*) – Index of the sleeve to travel. See *here*
- **cityName** (*string*) – Name of the destination city.

Return a boolean indicating whether or not the sleeve reached destination.

1.1.13.13 getSleeveAugmentations() Netscript Function

getSleeveAugmentations(*sleeveNumber*)

Arguments

- **sleeveNumber** (*int*) – Index of the sleeve to retrieve augmentations from. See *here*

Return a list of augmentation names that this sleeve has installed.

1.1.13.14 getSleevePurchasableAugs() Netscript Function

getSleevePurchasableAugs(*sleeveNumber*)

Arguments

- **sleeveNumber** (*int*) – Index of the sleeve to retrieve purchasable augmentations from. See *here*

Return a list of augmentations that the player can buy for this sleeve.

```
[
    {
        name: string, // augmentation name
        cost: number, // augmentation cost
    }
]
```

1.1.13.15 purchaseSleeveAug() Netscript Function

purchaseSleeveAug(*sleeveNumber*, *augName*)

Arguments

- **sleeveNumber** (*int*) – Index of the sleeve to buy an aug for. See *here*
- **augName** (*string*) – Name of the aug to buy. Must be an exact match

Return true if the aug was purchased and installed on the sleeve.

1.1.13.16 Referencing a Duplicate Sleeve

Most of the functions in the Sleeve API perform an operation on a single Duplicate Sleeve. In order to specify which Sleeve the operation should be performed on, a numeric index is used as an identifier. The index should follow array-notation, such that the first Duplicate Sleeve has an index of 0, the second Duplicate Sleeve has an index of 1, and so on.

The order of the Duplicate Sleeves matches the order on the UI page.

1.1.13.17 Examples

Basic example usage:

```
for (var i = 0; i < sleeve.getNumSleeves(); i++) {
    sleeve.setToShockRecovery(i);
}

sleep(10 * 60 * 60); // wait 10h

for (var i = 0; i < sleeve.getNumSleeves(); i++) {
    sleeve.setToSynchronize(i);
}

sleep(10*60*60); // wait 10h

for (var i = 0; i < sleeve.getNumSleeves(); i++) {
    sleeve.setToCommitCrime(i, 'shoplift');
}
```

1.1.14 Netscript Miscellaneous

1.1.14.1 Netscript Ports

Netscript Ports are endpoints that can be used to communicate between scripts. A port is implemented as a sort of serialized queue, where you can only write and read one element at a time from the port. When you read data from a port, the element that is read is removed from the port.

The *read()*, *write()*, *tryWrite()*, *clear()*, and *peek()* Netscript functions can be used to interact with ports.

Right now, there are only 20 ports for Netscript, denoted by the number 1 through 20. When using the functions above, the ports are specified by passing the number as the first argument.

IMPORTANT: The data inside ports are not saved! This means if you close and re-open the game, or reload the page then you will lose all of the data in the ports!

Example Usage

Here's a brief example of how ports work. For the sake of simplicity we'll only deal with port 1.

Let's assume Port 1 starts out empty (no data inside). We'll represent the port as such:

```
[]
```

Now assume we ran the following simple script:

```
for (i = 0; i < 10; ++i) {
    write(1, i); //Writes the value of i to port 1
}
```

After this script executes, our script will contain every number from 0 through 9, as so:

```
[0, 1, 2, 3, 4, 5, 6, 7 , 8, 9]
```

Then, assume we run the following script:

```
for (i = 0; i < 3; ++i) {
    print(read(1)); //Reads a value from port 1 and then prints it
}
```

This script above will read the first three values from port 1 and then print them to the script's log. The log will end up looking like:

```
0
1
2
```

And the data in port 1 will look like:

```
[3, 4, 5, 6, 7, 8, 9]
```

Warning: In *NetscriptJS (Netscript 2.0)*, do not trying writing base Promises to a port.

Port Handles

WARNING: Port Handles only work in *NetscriptJS (Netscript 2.0)*. They do not work in *Netscript 1.0*

The *getPortHandle()* Netscript function can be used to get a handle to a Netscript Port. This handle allows you to access several new port-related functions and the port's underlying data structure, which is just a JavaScript array. The functions are:

`NetscriptPort.`**`write`**`(`*data*`)`

Arguments

- **data** – Data to write to the port

Returns If the port is full, the item that is removed from the port is returned. Otherwise, null is returned.

Writes *data* to the port. Works the same as the Netscript function *write*.

`NetscriptPort.`**`tryWrite`**`(`*data*`)`

Arguments

- **data** – Data to try to write to the port

Returns True if the data is successfully written to the port, and false otherwise.

Attempts to write *data* to the Netscript port. If the port is full, the data will not be written. Otherwise, the data will be written normally.

`NetscriptPort.`**`full`**`()`

Returns True if the Netscript Port is full, and false otherwise

`NetscriptPort.`**`empty`**`()`

Returns True if the Netscript Port is empty, and false otherwise

`NetscriptPort.`**`clear`**`()`
Clears all data from the port. Works the same as the Netscript function *clear*

`NetscriptPort.`**`data`**
The Netscript port underlying data structure, which is just a Javascript array. All valid Javascript Array methods can be called on this.

Port Handle Example:

```
port = getPortHandle(5);
back = port.data.pop(); //Get and remove last element in port

//Remove an element from the port
i = port.data.findIndex("foo");
if (i != -1) {
    port.data.slice(i, 1);
}

//Wait for port data before reading
while(port.empty()) {
    sleep(10000);
}
res = port.read();

//Wait for there to be room in a port before writing
while (!port.tryWrite(5)) {
    sleep(5000);
}

//Successfully wrote to port!
```

1.1.14.2 Comments

Netscript supports comments using the same syntax as Javascript comments. Comments are not evaluated as code, and can be used to document and/or explain code:

```
//This is a comment and will not get executed even though its in the code
/* Multi
 * line
 * comment */
print("This code will actually get executed");
```

1.1.14.3 Importing Functions

In Netscript you can import functions that are declared in other scripts. The script will incur the RAM usage of all imported functions. There are two ways of doing this:

```
import * as namespace from "script filename"; //Import all functions from script
import {fn1, fn2, ...} from "script filename"; //Import specific functions from script
```

Suppose you have a library script called *testlibrary.script*:

```
function foo1(args) {
    //function definition...
}

function foo2(args) {
    //function definition...
}

function foo3(args) {
    //function definition...
}
```

(continues on next page)

(continued from previous page)

```
function foo4(args) {
    //function definition...
}
```

Then, if you wanted to use these functions in another script, you can import them like so:

```
import * as testlib from "testlibrary.script";

values = [1,2,3];

//The imported functions must be specified using the namespace
someVal1 = testlib.foo3(values);
someVal2 = testlib.foo1(values);
if (someVal1 > someVal2) {
    //...
} else {
    //...
}
```

If you only wanted to import certain functions, you can do so without needing to specify a namespace for the import:

```
import {foo1, foo3} from "testlibrary.script"; //Saves RAM since not all functions␣
↪are imported!

values = [1,2,3];

//No namespace needed
someVal1 = foo3(values);
someVal2 = foo1(values);
if (someVal1 > someVal2) {
    //...
} else {
    //...
}
```

Warning: For those who are experienced with JavaScript, note that the *export* keyword should **NOT** be used in *Netscript 1.0*, as this will break the script. It can, however, be used in *NetscriptJS (Netscript 2.0)* (but it's not required).

1.1.14.4 Standard, Built-In JavaScript Objects

Standard built-in JavaScript objects such as Math, Date, Number, and others are supported as expected based on which version of Netscript you use (i.e. *Netscript 1.0* will support built-in objects that are defined in ES5, and *NetscriptJS (Netscript 2.0)* will support whatever your browser supports).

1.2 Basic Gameplay

This section documents Bitburner gameplay elements that are immediately available and/or accessible to the player.

1.2.1 Stats

The player has several stats that can be increased in order to progress in the game.

1.2.1.1 Hacking

Represents the player's ability to code and hack.

Affects:

- Time it takes to hack a server
- Time it takes to execute the grow() and weaken() Netscript function
- Chance to successfully hack a server
- Percent money stolen when hacking a server
- Success rate of certain crimes
- Success rate of Hacking option during Infiltration
- Time it takes to create a program
- Faction reputation gain when carrying out Hacking Contracts or Field Work
- Company reputation gain for certain jobs

Gain experience by:

- Manually hacking servers through Terminal
- Executing hack(), grow(), or weaken() through a script
- Committing certain crimes
- Infiltration
- Carrying out Hacking Contracts or doing Field work for Factions
- Working certain jobs at a company
- Studying at a university

1.2.1.2 Strength

Represents the player's physical offensive power

Affects:

- Success rate of certain crimes
- Success rate of Combat options during Infiltration
- Faction reputation gain for Security and Field Work
- Company reputation gain for certain jobs

Gain experience by:

- Committing certain crimes
- Infiltration
- Working out at a gym

- Doing Security/Field Work for a faction
- Working certain jobs at a company

1.2.1.3 Defense

Represents the player's ability to withstand damage

Affects:

- Success rate of certain crimes
- The player's HP
- Success rate of Combat options during Infiltration
- How much damage the player takes during Infiltration
- Faction reputation gain for Security and Field Work
- Company reputation gain for certain jobs

Gain experience by:

- Committing certain crimes
- Infiltration
- Working out at a gym
- Doing Security/Field Work for a faction
- Working certain jobs at a company

1.2.1.4 Dexterity

Represents the player's skill and adeptness in performing certain tasks

Affects:

- Success rate of certain crimes
- Success rate of Combat, Lockpick, and Escape options during Infiltration
- Faction reputation gain for Security and Field Work
- Company reputation gain for certain jobs

Gain experience by:

- Committing certain crimes
- Infiltration
- Working out at a gym
- Doing Security/Field Work for a faction
- Working certain jobs at a company

1.2.1.5 Agility

Represents the player's speed and ability to move

Affects:

- Success rate of certain crimes
- Success rate of Combat, Sneak, and Escape options during Infiltration
- Faction reputation gain for Security and Field Work
- Company reputation gain for certain jobs

Gain experience by:

- Committing certain crimes
- Infiltration
- Working out at a gym
- Doing Security/Field Work for a faction
- Working certain jobs at a company

1.2.1.6 Charisma

Represents the player's social abilities

Affects:

- Success rate of certain crimes
- Success rate of Bribe option during Infiltration
- Faction reputation gain for Field Work
- Company reputation gain for most jobs

Gain experience by:

- Committing certain crimes
- Infiltration
- Working out at a gym
- Working a relevant job at a company
- Doing Field work for a Faction

1.2.2 Terminal

The Terminal is a console emulator program that lets you interface with all of the Servers in the game. The Terminal can be accessed by clicking the 'Terminal' tab on the navigation menu on the left-hand side of the game (you may need to expand the 'Hacking' header in order to see the 'Terminal' tab). Alternatively, the *keyboard shortcut* Alt + t can be used to open the Terminal.

1.2.2.1 Configuration

The terminal has a configuration file called .fconf. To edit this file, go to the terminal and enter:

```
nano .fconf
```

1.2.2.2 Filesystem (Directories)

The Terminal contains a **very** basic filesystem that allows you to store and organize your files into different directories. Note that this is **not** a true filesystem implementation. Instead, it is done almost entirely using string manipulation. For this reason, many of the nice & useful features you'd find in a real filesystem do not exist.

Here are the Terminal commands you'll commonly use when dealing with the filesystem.

- *ls*
- *cd*
- *mv*

1.2.2.2.1 Directories

In order to create a directory, simply name a file using a full absolute Linux-style path:

```
/scripts/myScript.js
```

This will automatically create a "directory" called `scripts`. This will also work for subdirectories:

```
/scripts/hacking/helpers/myHelperScripts.script
```

Files in the root directory do not need to begin with a forward slash:

```
thisIsAFileInTheRootDirectory.txt
```

Note that there is no way to manually create or remove directories. The creation and deletion of directories is automatically handled as you name/rename/delete files.

1.2.2.2.2 Absolute vs Relative Paths

Many Terminal commands accept absolute both absolute and relative paths for specifying a file.

An absolute path specifies the location of the file from the root directory (/). Any path that begins with the forward slash is an absolute path:

```
$ nano /scripts/myScript.js
$ cat /serverList.txt
```

A relative path specifies the location of the file relative to the current working directory. Any path that does **not** begin with a forward slash is a relative path. Note that the Linux-style dot symbols will work for relative paths:

```
. (a single dot) - represents the current directory
.. (two dots) - represents the parent directory

$ cd ..
```

(continues on next page)

(continued from previous page)

```
$ nano ../scripts/myScript.js
$ nano ../../helper.js
```

1.2.2.2.3 Netscript

Note that in order to reference a file, *Netscript* functions require the **full** absolute file path. For example

```
run("/scripts/hacking/helpers.myHelperScripts.script");
rm("/logs/myHackingLogs.txt");
rm("thisIsAFileInTheRootDirectory.txt");
```

Note: A full file path **must** begin with a forward slash (/) if that file is not in the root directory.

1.2.2.2.4 Missing Features

These features that are typically in Linux filesystems have not yet been added to the game:

- Tab autocompletion does not work with relative paths
- `mv` only accepts full filepaths for the destination argument. It does not accept directories

1.2.2.3 Commands

1.2.2.3.1 alias

$ alias [-g] [name=”value”]

Create or display aliases. An alias enables a replacement of a word with another string. It can be used to abbreviate a commonly used command, or commonly used parts of a command. The NAME of an alias defines the word that will be replaced, while the VALUE defines what it will be replaced by. For example, you could create the alias ‘nuke’ for the Terminal command ‘run NUKE.exe’ using the following:

```
$ alias nuke="run NUKE.exe"
```

Then, to run the NUKE.exe program you would just have to enter ‘nuke’ in Terminal rather than the full command. It is important to note that ‘default’ aliases will only be substituted for the first word of a Terminal command. For example, if the following alias was set:

```
$ alias worm="HTTPWorm.exe"
```

and then you tried to run the following terminal command:

```
$ run worm
```

This would fail because the worm alias is not the first word of a Terminal command. To allow an alias to be substituted anywhere in a Terminal command, rather than just the first word, you must set it to be a global alias using the -g flag:

```
$ alias -g worm="HTTPWorm.exe"
```

Now, the 'worm' alias will be substituted anytime it shows up as an individual word in a Terminal command.

Entering just the command 'alias' without any arguments prints the list of all defined aliases in the reusable form 'alias NAME=VALUE' on the Terminal.

The *unalias* Terminal command can be used to remove aliases.

1.2.2.3.2 analyze

Prints details and statistics about the current server. The information that is printed includes basic server details such as the hostname, whether the player has root access, what ports are opened/closed, and also hacking-related information such as an estimated chance to successfully hack, an estimate of how much money is available on the server, etc.

1.2.2.3.3 buy

$ buy [-l/program]

Purchase a program through the Dark Web. Requires a TOR Router to use.

If this command is ran with the '-l' flag, it will display a list of all programs that can be purchased through the Dark Web, as well as their costs.

Otherwise, the name of the program must be passed in as a parameter. This name is NOT case-sensitive:

```
$ buy brutessh.exe
```

Note that you do not need to be connected to the actual dark web server in order to run this command. You can use this command at any time on the Terminal.

1.2.2.3.4 cat

$ cat [filename]

Display a message (.msg), literature (.lit), or text (.txt) file:

```
$ cat j1.msg
$ cat foo.lit
$ cat servers.txt
```

1.2.2.3.5 cd

$ cd [dir]

Change to the specified directory.

See *Filesystem (Directories)* for details on directories.

Note that this command works even for directories that don't exist. If you change to a directory that doesn't exist, it will not be created. A directory is only created once there is a file in it:

```
$ cd scripts/hacking
$ cd /logs
$ cd ..
```

1.2.2.3.6 check

$ check [script name] [args...]

Print the logs of the script specified by the script name and arguments to the Terminal. Each argument must be separated by a space. **Remember that a running script is uniquely identified both by its name and the arguments that are used to start it**. So, if a script was ran with the following arguments:

```
$ run foo.script 1 2 foodnstuff
```

Then to run the 'check' command on this script you would have to pass the same arguments in:

```
$ check foo.script 1 2 foodnstuff
```

1.2.2.3.7 clear/cls

Clear the Terminal screen, deleting all of the text. Note that this does not delete the user's command history, so using the up and down arrow keys is still valid. Also note that this is permanent and there is no way to undo this. Both 'clear' and 'cls' do the same thing:

```
$ clear
$ cls
```

1.2.2.3.8 connect

$ connect [hostname/ip]

Connect to a remote server. The hostname or IP address of the remote server must be given as the argument to this command. Note that only servers that are immediately adjacent to the current server in the network can be connected to. To see which servers can be connected to, use the 'scan' command.

1.2.2.3.9 download

Downloads a script or text file to your computer (your real-life computer):

```
$ download masterScript.script
$ download importantInfo.txt
```

You can also download all of your scripts/text files as a zip file using the following Terminal commands:

```
$ download *
$ download *.script
$ download *.txt
```

1.2.2.3.10 expr

$ expr [math expression]

Evaluate a mathematical expression. The expression is evaluated in JavaScript, and therefore all JavaScript operators should be supported.

Examples:

```
$ expr 5.6 * 10 - 123
$ expr 3 ** 3
```

1.2.2.3.11 free

Display's the memory usage on the current machine. Print the amount of RAM that is available on the current server as well as how much of it is being used.

1.2.2.3.12 hack

Attempt to hack the current server. Requires root access in order to be run.

Related: Hacking Mechanics (TODO Add link here when page gets made)

1.2.2.3.13 help

$ help [command]

Display Terminal help information. Without arguments, 'help' prints a list of all valid Terminal commands and a brief description of their functionality. You can also pass the name of a Terminal command as an argument to 'help' to print more detailed information about the Terminal command. Examples:

```
$ help alias
$ help scan-analyze
```

1.2.2.3.14 home

Connect to your home computer. This will work no matter what server you are currently connected to.

1.2.2.3.15 hostname

Prints the hostname of the server you are currently connected to.

1.2.2.3.16 ifconfig

Prints the IP address of the server you are currently connected to.

1.2.2.3.17 kill

$ kill [script name] [args...] $ kill [pid]

Kill the script specified by the script filename and arguments OR by its PID.

If you are killing the script using its filename and arguments, then each argument must be separated by a space. Remember that a running script is uniquely identified by both its name and the arguments that are used to start it. So, if a script was ran with the following arguments:

```
$ run foo.script 50e3 sigma-cosmetics
```

Then to kill this script the same arguments would have to be used:

```
$ kill foo.script 50e3 sigma-cosmetics
```

If you are killing the script using its PID, then the PID argument must be numeric.

1.2.2.3.18 killall

Kills all scripts on the current server.

1.2.2.3.19 ls

$ ls [dir] [| grep pattern]

Prints files and directories on the current server to the Terminal screen.

If this command is run with no arguments, then it prints all files and directories on the current server to the Terminal screen. Directories will be printed first in alphabetical order, followed by the files (also in alphabetical order).

The `dir` optional parameter allows you to specify the directory for which to display files.

The `| grep pattern` optional parameter allows you to only display files and directories with a certain pattern in their names.

Examples:

```
// List files/directories with the '.script' extension in the current directory
$ ls | grep .script

// List files/directories with the '.js' extension in the root directory
$ ls / | grep .js

// List files/directories with the word 'purchase' in the name, in the␣
↪:code:`scripts` directory
$ ls scripts | grep purchase
```

1.2.2.3.20 lscpu

Prints the number of CPU cores the current server has.

1.2.2.3.21 mem

$ mem [script name] [-t] [num threads]

Displays the amount of RAM needed to run the specified script with a single thread. The command can also be used to print the amount of RAM needed to run a script with multiple threads using the '-t' flag. If the '-t' flag is specified, then an argument for the number of threads must be passed in afterwards. Examples:

```
$ mem foo.script
$ mem foo.script -t 50
```

The first example above will print the amount of RAM needed to run 'foo.script' with a single thread. The second example above will print the amount of RAM needed to run 'foo.script' with 50 threads.

1.2.2.3.22 mv

$ mv [source] [destination]

Move the source file to the specified destination in the filesystem. See *Filesystem (Directories)* for more details about the Terminal's filesystem. This command only works for scripts and text files (.txt). It cannot, however, be used to convert from script to text file, or vice versa.

This function can also be used to rename files.

Note: Unlike the Linux `mv` command, the *destination* argument must be the full filepath. It cannot be a directory.

Examples:

```
$ mv hacking.script scripts/hacking.script
$ mv myScript.js myOldScript.js
```

1.2.2.3.23 nano

$ nano [filename]

Opens up the specified file in the Text Editor. Only scripts (.script, .ns, .js) and text files (.txt) can be edited. If the file does not already exist, then a new empty file will be created.

1.2.2.3.24 ps

Prints all scripts that are currently running on the current server.

1.2.2.3.25 rm

$ rm [filename]

Removes the specified file from the current server. This works for every file type except literature files (.lit).

WARNING: This is permanent and cannot be undone

1.2.2.3.26 run

$ run [file name] [-t] [num threads] [args...]

Execute a program, script, or *Coding Contracts*.

The '[-t]', '[num threads]', and '[args...]' arguments are only valid when running a script. The '-t' flag is used to indicate that the script should be run with the specified number of threads. If the flag is omitted, then the script will be run with a single thread by default. If the '-t' flag is used, then it MUST come immediately after the script name, and the [num threads] argument MUST come immediately afterwards.

[args...] represents a variable number of arguments that will be passed into the script. See the documentation about script arguments. Each specified argument must be separated by a space.

Examples

Run a program:

```
$ run BruteSSH.exe
```

Run *foo.script* with 50 threads and the arguments [1e3, 0.5, foodnstuff]:

```
$ run foo.script -t 50 1e3 0.5 foodnstuff
```

Run a Coding Contract:

```
$ run foo-contract.cct
```

1.2.2.3.27 scan

Prints all immediately-available network connections. This will print a list of all servers that you can currently connect to using the 'connect' Terminal command.

1.2.2.3.28 scan-analyze

$ scan-analyze [depth]

Prints detailed information about all servers up to *[depth]* nodes away on the network. Calling 'scan-analyze 1' will display information for the same servers that are shown by the 'scan' Terminal command. This command also shows the relative paths to reach each server.

By default, the maximum depth that can be specified for 'scan-analyze' is 3. However, once you have the *DeepscanV1.exe* and *DeepscanV2.exe* programs, you can execute 'scan-analyze' with a depth up to 5 and 10, respectively.

The information 'scan-analyze' displays about each server includes whether or not you have root access to it, its required hacking level, the number of open ports required to run NUKE.exe on it, and how much RAM it has.

1.2.2.3.29 scp

$ scp [script name] [target server]

Copies the specified script from the current server to the target server. The second argument passed in must be the hostname or IP of the target server.

1.2.2.3.30 sudov

Prints whether or not you have root access to the current server.

1.2.2.3.31 tail

$ tail [script name] [args...]

Displays dynamic logs for the script specified by the script name and arguments. Each argument must be separated by a space. Remember that a running script is uniquely identified by both its name and the arguments that were used to run it. So, if a script was ran with the following arguments:

```
$ run foo.script 10 50000
```

Then in order to check its logs with 'tail' the same arguments must be used:

```
$ tail foo.script 10 50000
```

1.2.2.3.32 theme

$ theme [preset] | [#background #text #highlight]

Change the color of the game's user interface

This command can be called with a preset theme. Currently, the supported presets are:

- default
- muted
- solarized

However, you can also specify your own color scheme using hex values. To do so, you must specify three hex color values for the background color, the text color, and the highlight color. These hex values must be preceded by a pound sign (#) and must be either 3 or 6 digits. Example:

```
$ theme #ffffff #385 #235012
```

A color picker such as Google's can be used to get your desired hex color values

1.2.2.3.33 top

Prints a list of all scripts running on the current server as well as their thread count and how much RAM they are using in total.

1.2.2.3.34 unalias

$ unalias "[alias name]"

Deletes the specified alias. Note that the double quotation marks are required.

As an example, if an alias was declared using:

```
$ alias r="run"
```

Then it could be removed using:

```
$ unalias "r"
```

It is not necessary to differentiate between global and non-global aliases when using 'unalias'

1.2.2.3.35 wget

$ wget [url] [target file]

Retrieves data from a url and downloads it to a file on the current server. The data can only be downloaded to a script (.script, .ns, .js) or a text file (.txt). If the target file already exists, it will be overwritten by this command.

Note that will not be possible to download data from many websites because they do not allow cross-origin origin sharing (CORS). This includes websites such as gist and pastebin. One notable site it will work on is rawgithub. Example:

```
$ wget https://raw.githubusercontent.com/danielyxie/bitburner/master/README.md game_
↪readme.txt
```

1.2.2.4 Argument Parsing

When evaluating a terminal command, arguments are initially parsed based on whitespace (usually spaces). Each whitespace character signifies the end of an argument, and potentially the start of new one. For most terminal commands, this is all you need to know.

When running scripts, however, it is important to know in more detail how arguments are parsed. There are two main points:

1. Quotation marks can be used to wrap a single argument and force it to be parsed as a string. Any whitespace inside the quotation marks will not cause a new argument to be parsed.
2. Anything that can represent a number is automatically cast to a number, unless its surrounded by quotation marks.

Here's an example to show how these rules work. Consider the following script *argType.script*:

```
tprint("Number of args: " + args.length);
for (var i = 0; i < args.length; ++i) {
    tprint(typeof args[i]);
}
```

Then if we run the following terminal command:

```
$ run argType.script 123 1e3 "5" "this is a single argument"
```

We'll see the following in the Terminal:

```
Running script with 1 thread(s) and args: [123, 1000, "5", "this is a single argument
↪"].
May take a few seconds to start up the process...
argType.script: Number of args: 4
argType.script: number
argType.script: number
argType.script: string
argType.script: string
```

1.2.2.5 Chaining Commands

You can run multiple Terminal commands at once by separating each command with a semicolon (;).

Example:

```
$ run foo.script; tail foo.script
```

1.2.3 Servers

In this game, a server refers to a computer that can be connected to, accessed, and manipulated through the Terminal. All servers in the game are connected to each other to form a large, global network. To learn about how to navigate this network and connect to other servers, see the *Terminal* page.

1.2.3.1 Server RAM

Perhaps the most important property of a server to make note of is its RAM, which refers to how much memory is available on that machine. RAM is important because it is required to run Scripts. More RAM allows the user to run more powerful and complicated scripts.

The *free*, *scan-analyze*, and *analyze* Terminal commands can be used to check how much RAM a server has.

1.2.3.2 Identifying Servers

A server is identified by two properties: its IP address and its hostname. An IP address is a 32-bit number represented in dot-decimal notation. For example, "56.1.5.0" and "86.5.1.0" might be two IP addresses you see in the game. A hostname is a label assigned to a server. A hostname will usually give you a general idea of what the server is. For example, the company Nova Medical might have a server with the hostname "nova-med".

Hostnames and IP addresses are unique. This means that if one server has the IP address "1.1.1.1" and the hostname "some-server", then no other server in the game can have that IP address or that hostname.

There are many *Netscript Functions* and *Terminal* commands in the game that will require you to target a specific server. This is done using either the IP address or the hostname of the server.

1.2.3.3 Player-owned Servers

The player starts with a single server: his/her home computer. This server will have the hostname "home." The player's home computer is special for a variety of reasons:

1. The home computer's RAM can be upgraded. This can be done by visiting certain locations in the World.

2. The home computer persists through Augmentation Installations. This means that you will not lose any RAM upgrades or Scripts on your home computer when you install *Augmentations* (you will however, lose programs and messages on your home computer).

The player can also purchase additional servers. This can be done by visiting certain locations in the World, or it can be done automatically through a script using the `purchaseServer()` Netscript Function. The advantage of purchased servers is that, in terms of RAM, they are cheaper than upgrading your home computer. The disadvantage is that your purchased servers are lost when you install Augmentations.

1.2.3.4 Hackable Servers

Most servers that are not owned by the player can be hacked for money and exp. See the *Hacking* page for more details.

Different servers have different levels of security, but also offer different rewards when being hacked.

1.2.4 Hacking

In the year 2077, currency has become digital and decentralized. People and corporations store their money on servers. By hacking these servers, you can steal their money and gain experience.

1.2.4.1 Gaining Root Access

The first step to hacking a server is to gain root access to that server. This can be done using the NUKE virus (NUKE.exe). You start the game with a copy of the NUKE virus on your home computer. The NUKE virus attacks the target server's open ports using buffer overflow exploits. When successful, you are granted root administrative access to the machine.

In order for the NUKE virus to succeed, the target server needs to have enough open ports. Some servers have no security and will not need any ports opened. Some will have very high security and will need many ports opened. In order to open ports on another server, you will need to run programs that attack the server to open specific ports. These programs can be coded once your hacking skill gets high enough, or they can be purchased if you can find a seller.

There are two ways to execute port-opening programs and the NUKE virus:

1. Connect to the target server through the *Terminal* and use the *run* Terminal command:

```
$ run [programName]
```

2. Use a *Netscript Function*:

- `nuke()`
- `brutessh()`
- `ftpcrack()`
- `relaysmtp()`
- `httpworm()`
- `sqlinject()`

There are two ways to determine how many ports need to be opened on a server in order to successfully NUKE it:

1. Connect to that server through the *Terminal* and use the *analyze* command
2. Use the `getServerNumPortsRequired()` Netscript function

Once you have enough ports opened on a server and have ran the NUKE virus to gain root access, you will be able to hack it.

1.2.4.2 General Hacking Mechanics

When you execute the hack command, either manually through the terminal or automatically through a script, you attempt to hack the server. This action takes time. The more advanced a server's security is, the more time it will take. Your hacking skill level also affects the hacking time, with a higher hacking skill leading to shorter hacking times. Also, running the hack command manually through terminal is faster than hacking from a script.

Your attempt to hack a server will not always succeed. The chance you have to successfully hack a server is also determined by the server's security and your hacking skill level. Even if your hacking attempt is unsuccessful, you will still gain experience points.

When you successfully hack a server. You steal a certain percentage of that server's total money. This percentage is, once again, determined by the server's security and your hacking skill level. The amount of money on a server is not limitless. So, if you constantly hack a server and deplete its money, then you will encounter diminishing returns in your hacking (since you are only hacking a certain percentage). You can increase the amount of money on a server using a script and the `grow()` function in Netscript.

1.2.4.3 Server Security

Each server has a security level, typically between 1 and 100. A higher number means the server has stronger security. It is possible for a server to have a security of level 100 or higher, in which case hacking that server will become impossible (0% chance for hack to succeed).

As mentioned above, a server's security level is an important factor to consider when hacking. You can check a server's security level using the *analyze* Terminal command. You can also check a server's security in a script, using the `getServerSecurityLevel()` Netscript Function. See the Netscript documentation for more details.

Whenever a server is hacked manually or through a script, its security level increases by a small amount. Calling the `grow()` function in a script will also increase security level of the target server. These actions will make it harder for you to hack the server, and decrease the amount of money you can steal. You can lower a server's security level in a script using the `weaken()` function in Netscript. See the Netscript documentation for more details

A server has a minimum security level that is equal to one third of its starting security, rounded to the nearest integer. To be more precise:

```
server.minSecurityLevel = Math.max(1, Math.round(server.startingSecurityLevel / 3))
```

This means that a server's security level will not fall below this value if you are trying to weaken() it.

1.2.5 Scripts

Scripts are programs that can be used to automate the hacking process and almost every other part of the game. Scripts must be written in the *Netscript* language.

It is highly recommended that you have a basic background in programming to start writing scripts. You by no means need to be an expert. All you need is some familiarity with basic programming constructs like for/while loops, conditionals (if/else), functions, variables, etc. If you'd like to learn a little bit about programming, see *Learn to Program in Netscript*.

1.2.5.1 Script Arguments

When running a script, you can choose to pass arguments to that script. The script's logic can access and act on these arguments. This allows for flexibility in your scripts. For more details, see *Netscript Script Arguments*.

For information on how to run scripts with arguments, see *Working with Scripts in Terminal* and *Working with Scripts in Netscript* below.

1.2.5.2 Identifying a Script

Many commands and functions act on an executing script (i.e. a script that is running). Therefore, there must be a way to specify which script you want those commands & functions to act on.

A script that is being executed is uniquely identified by both its name and the arguments that it was run with.

The arguments must be an **exact** match. This means that both the order and type of the arguments matter.

1.2.5.3 Multithreading scripts

A script can be run with multiple threads. This is also called multithreading. The effect of multithreading is that every call to the `hack()`, `grow()`, and `weaken()` Netscript functions will have their results multiplied by the number of threads. For example, if a normal single-threaded script is able to hack $10,000, then running the same script with 5 threads would yield $50,000.

(This is the **only** affect of running a script with multiple threads. Scripts will not actually become multithreaded in the real-world sense.)

When multithreading a script, the total RAM cost can be calculated by simply multiplying the base RAM cost of the script with the number of threads, where the base cost refers to the amount of RAM required to run the script single-threaded. In the terminal, you can run the *mem* Terminal command to see how much RAM a script requires with *n* threads:

```
$ mem [scriptname] -t n
```

1.2.5.4 Working with Scripts in Terminal

Running a script requires RAM. The more complex a script is, the more RAM it requires to run. Scripts can be run on any server you have root access to.

Here are some *Terminal* commands that are useful when working with scripts:

check [script] [args...]

Prints the logs of the script specified by the name and arguments to Terminal. Arguments should be separated by a space. Remember that scripts are uniquely identified by their arguments as well as their name. For example, if you ran a script *foo.script* with the argument *foodnstuff* then in order to 'check' it you must also add the *foodnstuff* argument to the check command:

```
$ check foo.script foodnstuff
```

free

Shows the current server's RAM usage and availability

kill [script] [args...]

Stops a script that is running with the specified script name and arguments. Arguments should be separated by a space. Remember that scripts are uniquely identified by their arguments as well as their name. For example, if you ran a script *foo.script* with the argument 1 and 2, then just typing "*kill foo.script*" will not work. You have to use:

```
$ kill foo.script 1 2
```

mem [script] [-t] [n]

Check how much RAM a script requires to run with n threads

nano [script]

Create/Edit a script. The name of the script must end with a valid extension: .script, .js, or .ns

ps

Displays all scripts that are actively running on the current server

rm [script]

Delete a script from the server. This is permanent

run [script] [-t] [n] [args...]

Run a script with n threads and the specified arguments. Each argument should be separated by a space. Both the arguments and thread specification are optional. If neither are specified, then the script will be run single-threaded with no arguments.

Examples:

Run 'foo.script' single-threaded with no arguments:

```
$ run foo.script
```

Run 'foo.script' with 10 threads and no arguments:

```
$ run foo.script -t 10
```

Run 'foo.script' single-threaded with three arguments: [foodnstuff, sigma-cosmetics, 10]:

```
$ run foo.script foodnstuff sigma-cosmetics 10
```

Run 'foo.script' with 50 threads and a single argument: [foodnstuff]:

```
$ run foo.script -t 50 foodnstuff
```

tail [script] [args...]

Displays the logs of the script specified by the name and arguments. Note that scripts are uniquely identified by their arguments as well as their name. For example, if you ran a script 'foo.script' with the argument 'foodnstuff' then in order to 'tail' it you must also add the 'foodnstuff' argument to the tail command as so: tail foo.script foodnstuff

top

Displays all active scripts and their RAM usage

1.2.5.5 Working with Scripts in Netscript

TODO/Coming Soon...

1.2.5.6 Notes about how Scripts Work Offline

The scripts that you write and execute are interpreted in Javascript. For this reason, it is not possible for these scripts to run while offline (when the game is closed). It is important to note that for this reason, conditionals such as if/else statements and certain commands such as purchaseHacknetNode() or nuke() will not work while the game is offline.

However, Scripts WILL continue to generate money and hacking exp for you while the game is offline. This offline production is based off of the scripts' production while the game is online.

grow() and weaken() are two Netscript commands that will also be applied when the game is offline, although at a slower rate compared to if the game was open. This is done by having each script keep track of the rate at which the grow() and weaken() commands are called when the game is online. These calculated rates are used to determine how many times these function calls would be made while the game is offline.

Also, note that because of the way the Netscript interpreter is implemented, whenever you reload or re-open the game all of the scripts that you are running will start running from the BEGINNING of the code. The game does not keep track of where exactly the execution of a script is when it saves/loads.

1.2.6 World

In Bitburner, the world consists of six different cities:

- Sector-12 (this is where you start out)
- Aevum
- Ishima
- New Tokyo

- Chongqing
- Volhaven

1.2.7 Factions

Throughout the game you may receive invitations from factions. There are many different factions, and each faction has different criteria for determining its potential members. Joining a faction and furthering its cause is crucial to progressing in the game and unlocking endgame content.

It is possible to join multiple factions if you receive invitations from them. However, note that joining a faction may prevent you from joining other rival factions. (Don't worry, this usually isn't the case. Also, it would only be temporary since resetting the game by installing *Augmentations* will clear all your factions)

The 'Factions' link on the menu brings up a list of all factions that you have joined. You can select a Faction on this list to go to that Faction page. This page displays general information about the Faction and also lets you perform work for the faction. Working for a Faction is similar to working for a company except that you don't get paid a salary. You will only earn reputation in your Faction and train your stats. Also, cancelling work early when working for a Faction does **not** result in reduced reputation earnings.

Earning reputation for a Faction unlocks powerful Augmentations. Purchasing and installing these Augmentations will upgrade your abilities. The Augmentations that are available to unlock vary from faction to faction.

1.2.7.1 List of Factions and their Requirements

Early Game Factions	Faction Name	Requirements	Joining this Faction prevents you from joining:
	CyberSec	• Hack CSEC Manually	
	Tian Di Hui	• $1m • Hacking Level 50 • Be in Chongqing, New Tokyo, or Ishima	
	Netburners	• Hacking Level 80 • Total Hacknet Levels of 100 • Total Hacknet RAM of 8 • Total Hacknet Cores of 4	
City Factions	Sector-12	• Be in Sector-12 • $15m	• Chongqing • New Tokyo • Ishima • Volhaven

Continued on next page

Table 1 – continued from previous page

	Chongqing	• Be in Chongqing • $20m	• Sector-12 • Aevum • Volhaven
	New Tokyo	• Be in New Tokyo • $20m	• Sector-12 • Aevum • Volhaven
	Ishima	• Be in Ishima • $30m	• Sector-12 • Aevum • Volhaven
	Aevum	• Be in Aevum • $40m	• Chongqing • New Tokyo • Ishima • Volhaven
	Volhaven	• Be in Volhaven • $50m	• Sector-12 • Aevum • Chongqing • New Tokyo • Ishima
Hacking Groups	NiteSec	• Hack avmnite-02h manually • Home Computer RAM of at least 32GB	
	The Black Hand	• Hack I.I.I.I manually • Home Computer RAM of at least 64GB	
	Bitrunners	• Hack run4theh111z manually • Home Computer RAM of at least 128GB	

Continued on next page

Table 1 – continued from previous page

Megacorporations	ECorp	• Have 200k reputation with the Corporation	
	MegaCorp	• Have 200k reputation with the Corporation	
	KuaiGong International	• Have 200k reputation with the Corporation	
	Four Sigma	• Have 200k reputation with the Corporation	
	NWO	• Have 200k reputation with the Corporation	
	Blade Industries	• Have 200k reputation with the Corporation	
	OmniTek Incorporated	• Have 200k reputation with the Corporation	
	Bachman & Associates	• Have 200k reputation with the Corporation	
	Clarke Incorporated	• Have 200k reputation with the Corporation	
	Fulcrum Secret Technologies	• Have 250k reputation with the Corporation • Hack fulcrumassets manually	

Continued on next page

Table 1 – continued from previous page

Criminal Organizations	Slum Snakes	• All Combat Stats of 30 • -9 Karma • $1m	
	Tetrads	• Be in Chongqing, New Tokyo, or Ishima • All Combat Stats of 75 • -18 Karma	
	Silhouette	• CTO, CFO, or CEO of a company • $15m • -22 Karma	
	Speakers for the Dead	• Hacking Level 100 • All Combat Stats of 300 • 30 People Killed • -45 Karma • Not working for CIA or NSA	
	The Dark Army	• Hacking Level 300 • All Combat Stats of 300 • Be in Chongqing • 5 People Killed • -45 Karma • Not working for CIA or NSA	
	The Syndicate	• Hacking Level 200 • All Combat Stats of 200 • Be in Aevum or Sector-12 • $10m • -90 Karma • Not working for CIA or NSA	

Continued on next page

Table 1 – continued from previous page

Endgame Factions	The Covenant	• 20 Augmentations • $75b • Hacking Level of 850 • All Combat Stats of 850	
	Daedalus	• 30 Augmentations • $100b • Hacking Level of 2500 OR All Combat Stats of 1500	
	Illuminati	• 30 Augmentations • $150b • Hacking Level of 1500 • All Combat Stats of 1200	

1.2.8 Augmentations

Advances in science and medicine have lead to powerful new technologies that allow people to augment themselves beyond normal human capabilities. There are many different types of Augmentations, ranging from cybernetic to genetic to biological. Acquiring these Augmentations enhances the user's physical and mental faculties.

Augmentations provide persistent upgrades in the form of multipliers. These multipliers apply to a wide variety of things such as stats, experience gain, and hacking, just to name a few. Your multipliers can be viewed in the 'Character' page (*keyboard shortcut* Alt + c)

1.2.8.1 How to acquire Augmentations

Because of how powerful Augmentations are, the technology behind them is kept private and secret by the corporations and organizations that create them. Therefore, the only way for the player to obtain Augmentations is through Factions. After joining a Faction and earning enough reputation in it, you will be able to purchase its Augmentations. Different Factions offer different Augmentations. Augmentations must be purchased in order to be installed, and they are fairly expensive.

1.2.8.2 Installing Augmentations

You will not gain the benefits of your purchased Augmentations until you actually install them. You can choose to install Augmentations through the 'Augmentations' menu tab (Found under 'Character'. Alternatively, use the keyboard shortcut Alt + a).

Unfortunately, installing Augmentations has side effects. You will lose most of the progress you've made, including your skills, stats, and money. You will have to start over, but you will have all of the Augmentations you have installed to help you progress. This is the game's "soft reset" or "prestige" mechanic.

To summarize, here is a list of everything you will LOSE when you install an Augmentation:

- Stats/Skills
- Money
- Scripts on all servers EXCEPT your home computer
- Purchased servers
- Hacknet Nodes
- Company/faction reputation
- Jobs and Faction memberships
- Programs
- Stocks
- TOR router

Here is everything you will KEEP when you install an Augmentation:

- Every Augmentation you have installed
- Scripts on your home computer
- RAM Upgrades on your home computer
- World Stock Exchange account and TIX API Access

1.2.8.3 Purchasing Multiple Augmentations

You do not have to install an Augmentation right after you purchase it. You can purchase as many Augmentations as you'd like before you choose to install them. When you install your purchased Augmentations they will ALL get installed.

There are a few drawbacks to this, however. First, obviously, you won't gain the benefits of your purchased Augmentations until after you install them. Second, purchasing multiple Augmentations before installing them will cause the Augmentations to get progressively more expensive. When you purchase an Augmentation, the price of purchasing another Augmentation doubles. This multiplier stacks for each Augmentation you purchase. Once you install your purchased Augmentations, their costs are reset back to the original prices.

1.2.9 Companies

When exploring the *world*, you can visit various companies. At these companies, you can apply for jobs.

Working a job lets you earn money, experience, and reputation with that company.

1.2.9.1 Information about all Companies

TODO

1.2.10 Crimes

Commiting crimes is an active gameplay mechanic that allows the player to train their stats and potentially earn money. The player can attempt to commit crimes by visiting 'The Slums' through the 'City' tab (*Keyboard shortcut* Alt + w). 'The Slums' is available in every city.

1.2.10.1 Basic Mechanics

When you visit the 'Slums' you will see a list of buttons that show all of the available crimes. Simply select one of the options to begin attempting that crime. Attempting to commit a crime takes a certain amount of time. This time varies between crimes. During this time, you cannot do anything else.

Crimes are not always successful. Your rate of success is determined by your stats (and Augmentation multipliers) and can be seen on the crime-selection page. If you are unsuccessful at committing a crime you will gain EXP, but you will not earn money. If you are successful at committing the crime you will gain extra EXP (double of what an unsuccessful attempt would give) and earn money.

Harder crimes are typically more profitable, and also give more EXP.

1.2.10.2 Crime details

TODO

1.2.11 Infiltration

Infiltration is a gameplay mechanic that allows you to infiltrate a company's facility to try and steal the company's classified secrets. These secrets can be sold for money or for reputation with a faction.

1.2.11.1 Overview

Many companies have facilities that you can attempt to infiltrate. By infiltrating, you can steal classified company secrets and then sell these for money or for faction reputation. To try and infiltrate a company, visit a company through the 'World' menu. There will be an option that says 'Infiltrate Company'.

When infiltrating a company, you must progress through clearance levels in the facility. Every clearance level has some form of security that you must get past. There are several forms of security, ranging from high-tech security systems to armed guards. For each form of security, there are a variety of options that you can choose to try and bypass the security. Examples include hacking the security, engaging in combat, assassination, or sneaking past the security. The chance to succeed for each option is determined in part by your stats. So, for example, trying to hack the security system relies on your hacking skill, whereas trying to sneak past the security relies on your agility level.

The facility has a 'security level' that affects your chance of success when trying to get past a clearance level. Every time you advance to the next clearance level, the facility's security level will increase by a fixed percentage. Furthermore the options you choose and whether you succeed or fail will affect the security level as well. For example, if you try to kill a security guard and fail, the security level will increase by a lot. If you choose to sneak past security and succeed, the security level will not increase at all.

Every 5 clearance levels, you will steal classified company secrets that can be sold for money or faction reputation. However, in order to sell these secrets you must successfully escape the facility using the 'Escape' option. Furthermore, companies have a max clearance level. If you reach the max clearance level you will automatically escape the facility with all of your stolen secrets.

1.2.12 Stock Market

The Stock Market refers to the World Stock Exchange (WSE), through which you can buy and sell stocks in order to make money.

The WSE can be found in the 'City' tab, and is accessible in every city.

1.2.12.1 Fundamentals

The Stock Market is not as simple as "buy at price X and sell at price Y". The following are several fundamental concepts you need to understand about the stock market.

Note: For those that have experience with finance/trading/investing, please be aware that the game's stock market does not function exactly like it does in the real world. So these concepts below should seem similar, but won't be exactly the same.

1.2.12.1.1 Positions: Long vs Short

When making a transaction on the stock market, there are two types of positions: Long and Short. A Long position is the typical scenario where you buy a stock and earn a profit if the price of that stock increases. Meanwhile, a Short position is the exact opposite. In a Short position you purchase shares of a stock and earn a profit if the price of that stock decreases. This is also called 'shorting' a stock.

Note: Shorting stocks is not available immediately, and must be unlocked later in the game.

1.2.12.1.2 Forecast & Second-Order Forecast

A stock's forecast is its likelihood of increasing or decreasing in value. The forecast is typically represented by its probability of increasing in either a decimal or percentage form. For example, a forecast of 70% means the stock has a 70% chance of increasing and a 30% chance of decreasing.

A stock's second-order forecast is the target value that its forecast trends towards. For example, if a stock has a forecast of 60% and a second-order forecast of 70%, then the stock's forecast should slowly trend towards 70% over time. However, this is determined by RNG so there is a chance that it may never reach 70%.

Both the forecast and the second-order forecast change over time.

A stock's forecast can be viewed after purchasing Four Sigma (4S) Market Data access. This lets you see the forecast info on the Stock Market UI. If you also purchase access to the 4S Market Data TIX API, then you can view a stock's forecast using the `getStockForecast()` function.

A stock's second-order forecast is always hidden.

1.2.12.1.3 Spread (Bid Price & Ask Price)

The **bid price** is the maximum price at which someone will buy a stock on the stock market.

The **ask price** is the minimum price that a seller is willing to receive for a stock on the stock market

The ask price will always be higher than the bid price (This is because if a seller is willing to receive less than the bid price, that transaction is guaranteed to happen). The difference between the bid and ask price is known as the **spread**. A stock's "price" will be the average of the bid and ask price.

The bid and ask price are important because these are the prices at which a transaction actually occurs. If you purchase a stock in the long position, the cost of your purchase depends on that stock's ask price. If you then try to sell that stock (still in the long position), the price at which you sell is the stock's bid price. Note that this is reversed for a short position. Purchasing a stock in the short position will occur at the stock's bid price, and selling a stock in the short position will occur at the stock's ask price.

1.2.12.1.4 Transactions Influencing Stock Forecast

Buying or selling a large number of shares of a stock will influence that stock's forecast & second-order forecast. The forecast is the likelihood that the stock will increase or decrease in price. The magnitude of this effect depends on the number of shares being transacted. More shares will have a bigger effect.

The effect that transactions have on a stock's second-order forecast is significantly smaller than the effect on its forecast.

1.2.12.1.5 Order Types

There are three different types of orders you can make to buy or sell stocks on the exchange: Market Order, Limit Order, and Stop Order.

Note: Limit Orders and Stop Orders are not available immediately, and must be unlocked later in the game.

When you place a Market Order to buy or sell a stock, the order executes immediately at whatever the current price of the stock is. For example if you choose to short a stock with 5000 shares using a Market Order, you immediately purchase those 5000 shares in a Short position at whatever the current market price is for that stock.

A Limit Order is an order that only executes under certain conditions. A Limit Order is used to buy or sell a stock at a specified price or better. For example, lets say you purchased a Long position of 100 shares of some stock at a price of $10 per share. You can place a Limit Order to sell those 100 shares at $50 or better. The Limit Order will execute when the price of the stock reaches a value of $50 or higher.

A Stop Order is the opposite of a Limit Order. It is used to buy or sell a stock at a specified price (before the price gets 'worse'). For example, lets say you purchased a Short position of 100 shares of some stock at a price of $100 per share. The current price of the stock is $80 (a profit of $20 per share). You can place a Stop Order to sell the Short position if the stock's price reaches $90 or higher. This can be used to lock in your profits and limit any losses.

Here is a summary of how each order works and when they execute:

In a LONG Position:

A Limit Order to buy will execute if the stock's price <= order's price

A Limit Order to sell will execute if the stock's price >= order's price

A Stop Order to buy will execute if the stock's price >= order's price

A Stop Order to sell will execute if the stock's price <= order's price

In a SHORT Position:

A Limit Order to buy will execute if the stock's price >= order's price

A Limit Order to sell will execute if the stock's price <= order's price

A Stop Order to buy will execute if the stock's price <= order's price

A Stop Order to sell will execute if the stock's price >= order's price.

1.2.12.1.6 Player Actions Influencing Stocks

It is possible for your actions elsewhere in the game to influence the stock market.

Hacking If a server has a corresponding stock (e.g. *foodnstuff* server -> FoodNStuff stock), then hacking that server can decrease the stock's second-order forecast. This causes the corresponding stock's forecast to trend downwards in value over time.

This effect only occurs if you set the *stock* option to true when calling the `hack()` function. The chance that hacking a server will cause this effect is based on what percentage of the server's total money you steal.

A single hack will have a minor effect, but continuously hacking a server for lots of money over time will have a noticeable effect in making the stock's forecast trend downwards.

Growing If a server has a corresponding stock (e.g. *foodnstuff* server -> FoodNStuff stock), then growing that server's money can increase the stock's second-order forecast. This causes the corresponding stock's forecast to trend upwards in value over time.

This effect only occurs if you set the *stock* option to true when calling the `grow()` function. The chance that growing a server will cause this effect is based on what percentage of the server's total money to add to it.

A single grow operation will have a minor effect, but continuously growing a server for lots of money over time will have a noticeable effect in making the stock's forecast trend upwards.

Working for a Company If a company has a corresponding stock, then working for that company will increase the corresponding stock's second-order forecast. This will cause the stock's forecast to (slowly) trend upwards in value over time.

The potency of this effect is based on how "effective" you are when you work (i.e. its based on your stats and multipliers).

1.2.12.2 Automating the Stock Market

You can write scripts to perform automatic and algorithmic trading on the Stock Market. See *Netscript Trade Information eXchange (TIX) API* for more details.

1.2.12.3 Under the Hood

Stock prices are updated every ~6 seconds.

Whether a stock's price moves up or down is determined by RNG. However, stocks have properties that can influence the way their price moves. These properties are hidden, although some of them can be made visible by purchasing the Four Sigma (4S) Market Data upgrade. Some examples of these properties are:

- Volatility
- Likelihood of increasing or decreasing (i.e. the stock's forecast)
- Likelihood of forecast increasing or decreasing (i.e. the stock's second-order forecast)
- How easily a stock's price/forecast is influenced by transactions
- Spread percentage
- Maximum price (not a real maximum, more of a "soft cap")

Each stock has its own unique values for these properties.

1.2.12.4 Offline Progression

The Stock Market does not change or process anything while the game has closed. However, it does accumulate time when offline. This accumulated time allows the stock market to run 50% faster when the game is opened again. This means that stock prices will update every ~4 seconds instead of 6.

1.2.13 Coding Contracts

Coding Contracts are a mechanic that lets players earn rewards in exchange for solving programming problems.

Coding Contracts are files with the ".cct" extensions. They can be accessed through the *Terminal* or through scripts using the *Netscript Coding Contract API*

Each contract has a limited number of attempts. If you provide the wrong answer too many times and exceed the number of attempts, the contract will self destruct (delete itself)

Currently, Coding Contracts are randomly generated and spawned over time. They can appear on any server (including your home computer), except for your purchased servers.

1.2.13.1 Running in Terminal

To run a Coding Contract in the Terminal, simply use the *run* command:

```
$ run some-contract.cct
```

Doing this will bring up a popup. The popup will display the contract's problem, the number of attempts remaining, and an area to provide an answer.

1.2.13.2 Interacting through Scripts

See *Netscript Coding Contract API*.

1.2.13.3 Submitting Solutions

Different contract problem types will require different types of solutions. Some may be numbers, others may be strings or arrays. If a contract asks for a specific solution format, then use that. Otherwise, follow these rules when submitting solutions:

- String-type solutions should **not** have quotation marks surrounding the string (unless specifically asked for). Only quotation marks that are part of the actual string solution should be included.
- Array-type solutions should be submitted with each element in the array separated by commas. Brackets are optional. For example, both of the following are valid solution formats:

  ```
  1,2,3
  [1,2,3]
  ```

  However, if the solution is a multidimensional array, then all arrays that are not the outer-most array DO require the brackets. For example, an array of arrays can be submitted as one of the following:

  ```
  [1,2],[3,4]
  [[1,2],[3,4]]
  ```

- Numeric solutions should be submitted normally, as expected

1.2.13.4 Rewards

There are currently four possible rewards for solving a Coding Contract:

- Faction Reputation for a specific Faction
- Faction Reputation for all Factions that you are a member of

- Company reputation for a specific Company
- Money

The 'amount' of reward varies based on the difficulty of the problem posed by the Coding Contract. There is no way to know what a Coding Contract's exact reward will be until it is solved.

1.2.13.5 Notes

- The *scp* Terminal command does not work on Coding Contracts

1.2.13.6 List of all Problem Types

The following is a list of all of the problem types that a Coding Contract can contain. The list contains the name of (i.e. the value returned by `getContractType()`) and a brief summary of the problem it poses.

Name	Problem Summary
Find Largest Prime Factor	Given a number, find its largest prime factor. A prime factor is a factor that is a prime number.
Subarray with Maximum Sum	Given an array of integers, find the contiguous subarray (containing at least one number) which has the largest sum and return that sum.
Total Ways to Sum	Given a number, how many different ways can that number be written as a sum of at least two positive integers?
Spiralize Matrix	Given an array of array of numbers representing a 2D matrix, return the elements of that matrix in clockwise spiral order. Example: The spiral order of [1, 2, 3, 4] [5, 6, 7, 8] [9, 10, 11, 12] is [1, 2, 3, 4, 8, 12, 11, 10, 9, 5, 6, 7]
Array Jumping Game	You are given an array of integers where each element represents the maximum possible jump distance from that position. For example, if you are at position i and your maximum jump length is n, then you can jump to any position from i to i+n. Assuming you are initially positioned at the start of the array, determine whether you are able to reach the last index of the array EXACTLY.
Merge Overlapping Intervals	Given an array of intervals, merge all overlapping intervals. An interval is an array with two numbers, where the first number is always less than the second (e.g. [1, 5]). The intervals must be returned in ASCENDING order.

1.3 Advanced Gameplay

This section documents Bitburner gameplay elements that are **not** immediately available and/or accessible to the player. These gameplay mechanics must be unlocked.

Warning: This page contains spoilers regarding the game's story/plot-line.

1.3.1 BitNodes

A BitNode is an important part of the game's storyline. In the game, you discover what BitNodes are by following the trail of clues left by the mysterious jump3r (essentially a minimal questline).

1.3.1.1 What is a BitNode

A BitNode is the complex simulated reality in which you reside. By following the messages from jump3r, you discover that humanity was enslaved by an advanced alien race, called the Enders, using virtual simulations that trapped the minds of humans.

However, the Enders didn't just create a single virtual reality to enslave humans, but many different simulations. In other words, there are many different BitNodes that exist. These BitNode are very different from each other.

jump3r tells you that the only hope for humanity is to destroy all of these BitNodes. Therefore, the end goal for the player is to enter and then destroy each BitNode at least once.

Destroying a BitNode resets most of the player's progress but grants the player a powerful second-tier persistent upgrade called a *Source-File*. Different BitNodes grant different Source-Files.

Each BitNode has unique characteristics that are related to varying backstories. For example, in one BitNode the world is in the middle of a financial catastrophe with a collapsing market. In this BitNode, most forms of income such as working at a company or Hacknet Nodes are significantly less profitable. Servers have less money on them and lowered growth rates, but it is easier to lower their security level using the weaken() Netscript function.

Furthermore, some BitNodes introduce new content and mechanics. For example there is one BitNode that grants access to the *Netscript Singularity Functions*. There is another BitNode in which you can manage a gang to earn money and reputation.

1.3.1.2 How to destroy a BitNode

Initially, the only way to destroy a BitNode is to join the Daedalus *Daedalus*. From Daedalus, the player can obtain an Augmentation called 'The Red Pill', which doesn't cost any money but does require a good amount of faction reputation.

After installing 'The Red Pill', the player must search for and then manually hack a server called 'w0r1d_d43m0n'. This server requires a hacking level of 3000 in order to successfully hack it. This will destroy the player's current BitNode.

There is a second method of destroying a BitNode, but it must be unlocked by first destroying BitNode-6 or BitNode-7 (Bladeburners).

Todo: Link to Bladeburner documentation page here

When the player destroys a BitNode, most of his/her progress will be reset. This includes things such as Augmentations and RAM upgrades on the home computer. The only things that will persist through destroying BitNodes is:

- Source-Files
- Scripts on the home computer

1.3.1.3 BitNode Details

TODO

> **Warning:** This page contains spoilers regarding the game's story/plot-line.

1.3.2 Source-Files

Source-Files are a type of persistent upgrade that are more powerful than Augmentations. Source-Files are received by destroying a BitNode. There are many different BitNodes in the game and each BitNode will grant a different Source-File when it is destroyed.

A Source-File can be upgraded by destroying its corresponding BitNode a second or third time (AKA playing through that BitNode again). It can be upgraded to a maximum of level 3.

1.3.2.1 List of all Source-Files

BitNode-1: Source Genesis	• Lets the player start with 32 GB of RAM on home computer • Increases all of the player's multipliers by 16%/24%/28%
BitNode-2: Rise of the Underworld	• Increases the player's crime success rate, crime money, and charisma multipliers by 24%/36%/42%
BitNode-3: Corporatocracy	• Lets the player create Corporations in other BitNodes (although some BitNodes will disable this mechanic) • Increases the player's charisma and company salary multipliers by 8%/12%/14%
BitNode-4: The Singularity	• Lets the player access and use Netscript Singularity Functions in other BitNodes. • Each level of this Source-File opens up more of the Singularity Functions to use
BitNode-5: Artificial Intelligence	• Unlocks *Intelligence* • Unlocks `getBitNodeMultipliers()` Netscript function • Increases all of the player's hacking-related multipliers by 8%/12%/14%
BitNode-6: Bladeburners	• Unlocks the Bladeburner feature in other BitNodes • Increases all of the player's level and experience gain rate multipliers for combat stats by 8%/12%/14%
BitNode-7: Bladeburners 2079	• Allows the player to access the *Netscript Bladeburner API* in other BitNodes • Increases all of the player's Bladeburner multipliers by 8%/12%/14%
BitNode-8: Ghost of Wall Street	• Increases the player's hacking growth multiplier by 12%/18%/21% • Level 1 grants permanent access to *WSE* and *TIX API* • Level 2 grants permanent access to shorting stocks • Level 3 grants permanent access to use limit/stop orders
BitNode-9: Coming Soon	
BitNode-10: Digital Carbon	• Each level of this grants a Duplicate Sleeve • Allows the player to access the *Netscript Sleeve API* in other BitNodes
BitNode-11: The Big Crash	• Company wages increases both the player's salary

1.3.3 Intelligence

Intelligence is a *stat* that is unlocked by having *Source-File 5* (i.e. Destroying BitNode-5).

Intelligence is unique because it is permanent and persistent. It never gets reset back to 1. However, gaining Intelligence experience is extremely slow. The methods of gaining Intelligence exp is also hidden. You won't know when you gain experience and how much. It is a stat that gradually builds up as you continue to play the game.

Intelligence will boost your production for many actions in the game, including:

- Hacking
- Infiltration
- Hacking Missions
- Crime success rate
- Bladeburner
- Reputation gain for companies & factions

1.3.4 Sleeves

When VitaLife unveiled their Persona Core technology that allowed people to digitize and transfer their consciousness into other vessels, human bodies became nothing more than 'sleeves' for the human consciousness. This technology thus became known as "Sleeve technology".

Sleeve technology unlocks two different gameplay features:

- Duplicate Sleeves
- Re-sleeving

Sleeve technology is unlocked in *BitNode-10*.

1.3.4.1 Duplicate Sleeves

Duplicate Sleeves are MK-V Synthoids (synthetic androids) into which your consciuosness has been copied. In other words, these Synthoids contain a perfect duplicate of your mind.

Duplicate Sleeves are essentially clones which you can use to perform work-type actions, such as working for a company/faction or committing a crime. When sleeves perform these tasks, they will earn money, experience, and reputation.

Sleeves are their own individuals, which means they each have their own experience and stats.

When a sleeve earns experience, it earns experience for itself, the player's original consciousness, as well as all of the player's other sleeves.

Duplicate Sleeves are **not** reset when installing Augmentations, but they are reset when switching BitNodes.

1.3.4.1.1 Obtaining Duplicate Sleeves

There are two methods of obtaining Duplicate Sleeves:

1. Destroy BitNode-10. Each completion give you one additional Duplicate Sleeve

2. Purchase Duplicate Sleeves from *the faction The Covenant*. This is only available in BitNodes-10 and above, and is only available after defeating BitNode-10 at least once. Sleeves purchased this way are **permanent** (they persist through BitNodes). You can purchase up to 5 Duplicate Sleeves from The Covenant.

1.3.4.1.2 Synchronization

Synchronization is a measure of how aligned your consciousness is with that of your Duplicate Sleeves. It is a numeral value between 1 and 100, and it affects how much experience is earned when the sleeve is performing a task.

Let N be the sleeve's synchronization. When the sleeve earns experience by performing a task, both the sleeve and the player's original host consciousness of N% of the amount of experience normally earned by the task. All of the player's other sleeves earn ((N/100)^2 * 100)% of the experience.

Synchronization can be increased by assigning sleeves to the 'Synchronize' task.

1.3.4.1.3 Sleeve Shock

Sleeve shock is a measure of how much trauma the sleeve has due to being placed in a new body. It is a numeral value between 0 and 99, where 99 indicates full shock and 0 indicates no shock. Shock affects the amount of experience earned by the sleeve.

Sleeve shock slowly decreases over time. You can further increase the rate at which it decreases by assigning sleeves to the 'Shock Recovery' task.

1.3.4.1.4 Augmentations

You can purchase *Augmentations* for your Duplicate Sleeves. In order to do this, the Sleeve's Shock must be at 0. Any Augmentation that is currently available to you through a faction is also available for your Duplicate Sleeves. There are a few Augmentations, such as NeuroFlux Governor and Bladeburner-specific ones, that cannot be purchased for a Duplicate Sleeve.

When you purchase an Augmentation for a Duplicate Sleeve, it is instantly installed. When this happens, the Sleeve's stats are instantly reset back to 0, similar to when you normally install Augmentations.

The cost of purchasing an Augmentation for a Duplicate Sleeve is **not** affected by how many Augmentations you have purchased for yourself, and vice versa.

1.3.4.1.5 Memory

Sleeve memory dictates what a sleeve's synchronization will be when its reset by switching BitNodes. For example, if a sleeve has a memory of 10, then when you switch BitNodes its synchronization will initially be set to 10, rather than 1.

Memory can only be increased by purchasing upgrades from The Covenant. Just like the ability to purchase additional sleeves, this is only available in BitNodes-10 and above, and is only available after defeating BitNode-10 at least once.

Memory is a persistent stat, meaning it never gets reset back to 1. The maximum possible value for a sleeve's memory is 100.

1.3.4.2 Re-sleeving

Re-sleeving is the process of digitizing and transferring your consciousness into a new human body, or "sleeve". When you re-sleeve into a new body, your stat experience and Augmentations get replaced with those of the new body.

In order to re-sleeve, you must purchase new bodies. This can be done at VitaLife in New Tokyo. Once you purchase a body to re-sleeve into, the effects will take place immediately.

Note that resleeving **REMOVES** all of your currently-installed Augmentations, and replaces them with the ones provided by the purchased sleeve. However, Augmentations that are purchased but not installed will **not** be removed. If you have purchased an Augmentation and then re-sleeve into a body which already has that Augmentation, it will be removed since you cannot have duplicate Augmentations.

1.4 Keyboard Shortcuts

This page documents the various keyboard shortcuts that can be used in the game.

1.4.1 Game Navigation

These are used to switch between the different menus/tabs in the game. These shortcuts are almost always available. Exceptions include:

- Working at a company or for a faction
- Creating a program
- Taking a university class
- Training at a gym
- Active Mission (aka Hacking Mission)

Shortcut	Action
Alt + t	Switch to *Terminal*
Alt + c	Switch to 'Stats' page
Alt + e	Switch to Script Editor. Will open up the last-edited file or a new file
Alt + s	Switch to 'Active Scripts' page
Alt + h	Switch to 'Hacknet Nodes' page
Alt + w	Switch to 'City' page
Alt + j	Go to the company where you are employed ('Job' page on navigation menu)
Alt + r	Go to Travel Agency in current City ('Travel' page on navigation menu)
Alt + p	Switch to 'Create Program' page
Alt + f	Switch to 'Factions' page
Alt + a	Switch to 'Augmentations' page
Alt + u	Switch to 'Tutorial' page
Alt + o	Switch to 'Options' page

1.4.2 Script Editor

See the *Script Editor* documentation for more details.

1.4.3 Terminal Shortcuts

These shortcuts are available only in the *Terminal*

Shortcut	Action
Up/Down arrow	Cycle through previous commands
Ctrl + c	Cancel a hack/analyze action
Ctrl + l	Clear screen
Tab	Autocomplete command

1.4.4 Terminal Bash Shortcuts

These shortcuts were implemented to better emulate a bash shell. They must be enabled in your *Terminal*'s *.fconf* file. This can be done be entering the Terminal command:

```
nano .fconf
```

and then setting the *ENABLE_BASH_HOTKEYS* option to 1.

Note that these Bash shortcuts override any other shortcuts defined in the game (unless otherwise noted), as well as your browser's shortcuts

Also note that more Bash-like shortcuts will be implemented in the future

Shortcut	Action
Ctrl + c	Clears current Terminal input (does NOT override default Ctrl + c command)
Ctrl + p	Same as Up Arrow
Ctrl + m	Same as Down Arrow
Ctrl + a	Move cursor to beginning of line (same as 'Home' key)
Ctrl + e	Move cursor to end of line (same as 'End' key)
Ctrl + b	Move cursor to previous character
Alt + b	Move cursor to previous word
Ctrl + f	Move cursor to next character
Alt + f	Move cursor to next word
Ctrl + h/d	Delete previous character ('Backspace')

1.4.5 Popup/Dialog Box Shortcuts

The following shortcuts work if there are any popup or dialog boxes on the screen.

Shortcut	Action
Esc	Close the current popup, cancelling any prompts on a dialog box
Enter	Clicks the "Yes/Confirm" option for every dialog box

1.5 Script Editors

Third-party libraries are used to implement the game's Script Editor(s). There are currently two options for the Script Editor:

- Ace
- CodeMirror

You can select which of the two editors you want to use on the Script Editor page ('Create Script' on the main menu).

Ace was the game's original Script Editor, while CodeMirror was added later in v0.43.0. The two editors share many of the same features, so there is not a significant difference between the two. Currently, CodeMirror is slightly more modern, more customizable, and has a few quality-of-life improvements compared to Ace.

1.5.1 Universal Key Bindings

These keyboard shortcuts are available in both the Ace and CodeMirror editors, regardless of what key binding option you are using:

Shortcut	Action
Ctrl + b	Save script and return to *Terminal*
Ctrl + space	Show Autocomplete Hints

1.5.2 Linter

Both script editors contain a linter, which is a tool that analyzes your code and flags anything it thinks might be an error. You can see warnings and errors from the linter on the left-hand side of the script editor. There will be an icon on whatever lines the linter thinks might be problematic. Hovering over the icon will display information on what the issue is.

Note that **just because the linter shows an error/warning, this does NOT automatically mean that your script is broken and will fail to run.** This is especially true if you are using *NetscriptJS (Netscript 2.0)*. The linter used by the script editors isn't necessarily perfect or up-to-date. Furthermore, the linter does not affect anything when you actually run scripts.

1.5.3 Ace

The following documents what the different settings/options do for the Ace editor, as well as the different key binding settings. Note that the information for the key bindings may not be completely comprehensive. You'll have to dig into the editor source code if you want to learn more.

1.5.3.1 Settings

Setting	Effect
Theme	Switch between different color schemes
Key Binding	Switch between different key binding options. This changes what keyboard shortcuts are available
Highlight Active Line	When enabled, the line on which the cursor currently resides will be highlighted.
Show Invisibles	When enabled, you will be able to view hidden whitespace characters such as spaces, tabs, and newlines.
Use Soft Tab	When enabled, tabs will be replaced with spaces
Max Error Count	Specifies the (approximate) number of lines that will be linted

1.5.3.2 Ace Key Bindings

For Ace, the "Ace" Key Binding setting uses the default configuration. A list of these can be found here.

1.5.3.3 Vim Key Bindings

For Ace, the "Vim" Key Binding setting configures the editor to use Vim key mappings. Note that while this tries to emulate Vim features as faithfully as possible, it is not a complete Vim implementation.

Since I'm not familiar with Vim, I'll leave Ace's Vim Mode implementation here, which I believe shows most of the implemented features.

Note that the following Vim Ex commands will properly save the script and/or quit the editor in game:

Command	Effect
:w	Save the script and return to *Terminal*
:q	Return to *Terminal* **WITHOUT** saving
:x	Save the script and return to *Terminal*
:wq	Save the script and return to *Terminal*

1.5.3.4 Emacs Key Bindings

For Ace, the "Emacs" Key Binding setting configures the editor to use Emacs key mappings. Note that while this tries to emulate the Emacs key mappings as faithfully as possible, it won't necessarily be a complete implementation.

Since I'm not familiar with Emacs, I'll leave Ace's Emacs Mode implementation here, which I believe shows most of the implemented features.

1.5.4 CodeMirror

The following documents what the different settings/options do for the CodeMirror editor, as well as the shortcuts for the different key binding settings. Note that the information for the key bindings may not be completely comprehensive. You'll have to dig into the editor source code if you want to learn everything.

1.5.4.1 Settings

Setting	Effect
Theme	Switch between different color schemes
Key Binding	Switch between different key binding options. This changes what keyboard shortcuts are available
Highlight Active Line	When enabled, the line on which the cursor currently resides will be highlighted.
Show Invisibles	When enabled, you will be able to view hidden whitespace characters such as spaces, tabs, and newlines.
Use Soft Tab	When enabled, tabs will be replaced with spaces
Auto-Close Brackets/Quotes	When enabled, any opening brackets or quotes that are typed will be closed
Enable Linting	Enable/Disable the *Linter*
Continue Comments	When enabled, pressing 'Enter' inside a comment block will continue the comment on the next line

1.5.4.2 Default Key Bindings

Todo: Fill out

1.5.4.3 Sublime Key Bindings

Todo: Fill out

1.5.4.4 Vim Key Bindings

Todo: Fill out

1.5.4.5 Emacs Key Bindings

Todo: Fill out

1.6 Game Frozen or Stuck?

1.6.1 Infinite Loop in NetscriptJS

If your game is frozen or stuck in any way, then the most likely culprit is an infinitely running loop in *NetscriptJS (Netscript 2.0)*. To get past the freezing, run the game with *?noScripts* in the URL:

https://danielyxie.github.io/bitburner/?noScripts

Then, to fix your script, make sure you have a sleep or any other timed function like *hack()* or *grow()* in any infinite loops:

```
while(true) {
    // This is an infinite loop that does something
    ...
    await ns.sleep(1000); // Add a 1s sleep to prevent freezing
}
```

1.6.2 Bug

Otherwise, the game is probably frozen/stuck due to a bug. To report a bug, follow the guidelines here.

1.7 Guides & Tips

Getting Started Guide for Intermediate Programmers

Beginners FAQ

1.7.1 Getting Started Guide for Beginner Programmers

Note: Note that the scripts and strategies given in this guide aren't necessarily optimal. They're just meant to introduce you to the game and help you get started.

This is an introductory guide to getting started with Bitburner. It is not meant to be a comprehensive guide for the entire game, only the early stages. If you are confused or overwhelmed by the game, especially the programming and scripting aspects, this guide is perfect for you!

Note that this guide is tailored towards those with minimal programming experience.

1.7.1.1 Introduction

Bitburner is a cyberpunk-themed incremental RPG. The player progresses by raising their *Stats*, earning money, and *climbing the corporate ladder*. Eventually, after reaching certain criteria, the player will begin receiving invitations from *Factions*. Joining these factions and working for them will unlock *Augmentations*. Purchasing and installing Augmentations provide persistent upgrades and are necessary for progressing in the game.

The game has a minimal story/quest-line that can be followed to reach the end of the game. Since this guide is only about getting started with Bitburner, it will not cover the entire "quest-line".

1.7.1.2 First Steps

I'm going to assume you followed the introductory tutorial when you first began the game. In this introductory tutorial you created a script called `foodnstuff.script` and ran it on the `foodnstuff` server. Right now, we'll kill this script. There are two ways to do this:

1. You can go to the Terminal and enter:

```
$ kill foodnstuff.script
```

2. You can go to the `Active Scripts` page (*Keyboard shortcut* Alt + s) and press the "Kill Script" button for `foodnstuff.script`.

If you skipped the introductory tutorial, then ignore the part above. Instead, go to the `Hacknet Nodes` page (*Keyboard shortcut* Alt + h) and purchase a Hacknet Node to start generating some passive income.

1.7.1.3 Creating our First Script

Now, we'll create a generic hacking script that can be used early on in the game (or throughout the entire game, if you want).

Before we write the script, here are some things you'll want to familiarize yourself with:

- *General Hacking Mechanics*
- *Server Security*
- *`hack()`*
- *`grow()`*
- *`weaken()`*
- *`brutessh()`*
- *`nuke()`*

To briefly summarize the information from the links above: Each server has a security level that affects how difficult it is to hack. Each server also has a certain amount of money, as well as a maximum amount of money it can hold. Hacking a server steals a percentage of that server's money. The *hack()* Netscript function is used to hack server. The *grow()* Netscript function is used to increase the amount of money available on a server. The *weaken()* Netscript function is used to decrease a server's security level.

Now let's move on to actually creating the script. Go to your home computer and then create a script called `early-hack-template.script` by going to Terminal and entering the following two commands:

```
$ home
$ nano early-hack-template.script
```

This will take you to the script editor, which you can use to code and create *Scripts*. It will be helpful to consult the *Netscript* documentation. Specifically, you'll want to take a look at *Netscript Basic Functions*.

Enter the following code in the script editor:

```
// Defines the "target server", which is the server
// that we're going to hack. In this case, it's "foodnstuff"
var target = "foodnstuff";

// Defines how much money a server should have before we hack it
// In this case, it is set to 75% of the server's max money
var moneyThresh = getServerMaxMoney(target) * 0.75;

// Defines the maximum security level the target server can
// have. If the target's security level is higher than this,
// we'll weaken it before doing anything else
var securityThresh = getServerMinSecurityLevel(target) + 5;

// If we have the BruteSSH.exe program, use it to open the SSH Port
// on the target server
if (fileExists("BruteSSH.exe", "home")) {
    brutessh(target);
}

// Get root access to target server
nuke(target);

// Infinite loop that continously hacks/grows/weakens the target server
while(true) {
    if (getServerSecurityLevel(target) > securityThresh) {
        // If the server's security level is above our threshold, weaken it
        weaken(target);
    } else if (getServerMoneyAvailable(target) < moneyThresh) {
        // If the server's money is less than our threshold, grow it
        grow(target);
    } else {
        // Otherwise, hack it
        hack(target);
    }
}
```

The script above contains comments that document what it does, but let's go through it step-by-step anyways.

```
var target = "foodnstuff";
```

This first command defines a string which contains our target server. That's the server that we're going to hack. For now, it's set to *foodnstuff* because that's the only server with a required hacking level of 1. If you want to hack a

different server, simply change this variable to be the hostname of another server.

```
var moneyThresh = getServerMaxMoney(target) * 0.75;
```

This second command defines a numerical value representing the minimum amount of money that must be available on the target server in order for our script to hack it. If the money available on the target server is less than this value, then our script will *grow()* the server rather than hacking it. It is set to 75% of the maximum amount of money that can be available on the server. The *getServerMaxMoney()* Netscript function is used to find this value

```
var securityThresh = getServerMinSecurityLevel(target) + 5;
```

This third command defines a numerical value representing the maximum security level the target server can have. If the target server's security level is higher than this value, then our script will *weaken()* the script before doing anything else.

```
if (fileExists("BruteSSH.exe", "home")) {
    brutessh(target);
}

nuke(target);
```

This section of code is used to gain root access on the target server. This is necessary for hacking. See *here for more details*.

```
while (true) {
    if (getServerSecurityLevel(target) > securityThresh) {
        // If the server's security level is above our threshold, weaken it
        weaken(target);
    } else if (getServerMoneyAvailable(target) < moneyThresh) {
        // Otherwise, if the server's money is less than our threshold, grow it
        grow(target);
    } else {
        // Otherwise, hack it
        hack(target);
    }
}
```

This is the main section that drives our script. It dictates the script's logic and carries out the hacking operations. The *while (true)* creates an infinite loop that will continuously run the hacking logic until the the script is killed.

1.7.1.4 Running our Scripts

Now we want to start running our hacking script so that it can start earning us money and experience. Our home computer only has 8GB of RAM and we'll be using it for something else later. So instead, we'll take advantage of the RAM on other machines.

Go to `Terminal` and enter the following command:

```
$ scan-analyze 2
```

This will show detailed information about some servers on the network. The **network is randomized so it will be different for every person**. Here's what mine showed at the time I made this:

```
[home ~]> scan-analyze 2
~~~~~~~~~~ Beginning scan-analyze ~~~~~~~~~~
```

(continues on next page)

(continued from previous page)

```
>foodnstuff
--Root Access: NO, Required hacking skill: 1
--Number of open ports required to NUKE: 0
--RAM: 16

>sigma-cosmetics
--Root Access: NO, Required hacking skill: 5
--Number of open ports required to NUKE: 0
--RAM: 16

>joesguns
--Root Access: NO, Required hacking skill: 10
--Number of open ports required to NUKE: 0
--RAM: 16

---->max-hardware
------Root Access: NO, Required hacking skill: 80
------Number of open ports required to NUKE: 1
------RAM: 32

>hong-fang-tea
--Root Access: NO, Required hacking skill: 30
--Number of open ports required to NUKE: 0
--RAM: 16

---->nectar-net
------Root Access: NO, Required hacking skill: 20
------Number of open ports required to NUKE: 0
------RAM: 16

>harakiri-sushi
--Root Access: NO, Required hacking skill: 40
--Number of open ports required to NUKE: 0
--RAM: 16

>iron-gym
--Root Access: NO, Required hacking skill: 100
--Number of open ports required to NUKE: 1
--RAM: 32

---->zer0
------Root Access: NO, Required hacking skill: 75
------Number of open ports required to NUKE: 1
------RAM: 32

---->CSEC
------Root Access: NO, Required hacking skill: 54
------Number of open ports required to NUKE: 1
------RAM: 8
```

Take note of the following servers:

- `foodnstuff`
- `sigma-cosmetics`
- `joesguns`
- `nectar-net`

- `hong-fang-tea`
- `harakiri-sushi`

All of these servers have 16GB of RAM. Furthermore, all of these servers do not require any open ports in order to NUKE. In other words, we can gain root access to all of these servers and then run scripts on them.

First, let's determine how many threads of our hacking script we can run. *Read more about multithreading scripts here* The script we wrote uses 2.6GB of RAM. You can check this using the following `Terminal` command:

```
$ mem early-hack-template.script
```

This means we can run 6 threads on a 16GB server. Now, to run our scripts on all of these servers, we have to do the following:

1. Use the *scp* `Terminal` command to copy our script to each server.
2. Use the *connect* `Terminal` command to connect to a server.
3. Use the *run* `Terminal` command to run the *NUKE.exe* program and gain root access.
4. Use the *run* `Terminal` command again to run our script.
5. Repeat steps 2-4 for each server.

Here's the sequence of `Terminal` commands I used in order to achieve this:

```
$ home
$ scp early-hack-template.script foodnstuff
$ scp early-hack-template.script sigma-cosmetics
$ scp early-hack-template.script joesguns
$ scp early-hack-template.script nectar-net
$ scp early-hack-template.script hong-fang-tea
$ scp early-hack-template.script harakiri-sushi
$ connect foodnstuff
$ run NUKE.exe
$ run early-hack-template.script -t 6
$ home
$ connect sigma-cosmetics
$ run NUKE.exe
$ run early-hack-template.script -t 6
$ home
$ connect joesguns
$ run NUKE.exe
$ run early-hack-template.script -t 6
$ home
$ connect hong-fang-tea
$ run NUKE.exe
$ run early-hack-template.script -t 6
$ home
$ connect harakiri-sushi
$ run NUKE.exe
$ run early-hack-template.script -t 6
$ home
$ connect hong-fang-tea
$ connect nectar-net
$ run NUKE.exe
$ run early-hack-template.script -t 6
```

Note: Pressing the `Tab` key in the middle of a Terminal command will attempt to auto-complete the command. For

example, if you type in `scp ea` and then hit `Tab`, the rest of the script's name should automatically be filled in. This works for most commands in the game!

The *home* `Terminal` command is used to connect to the home computer. When running our scripts with the `run early-hack-template.script -t 6` command, the `-t 6` specifies that the script should be run with 6 threads.

Note that the `nectar-net` server isn't in the home computer's immediate network. This means you can't directly connect to it from home. You will have to search for it inside the network. The results of the *scan-analyze 2* command we ran before will show where it is. In my case, I could connect to it by going from *hong-fang-tea -> nectar-net*. However, this will probably be different for you.

After running all of these `Terminal` commands, our scripts are now up and running. These will earn money and hacking experience over time. These gains will be really slow right now, but they will increase once our hacking skill rises and we start running more scripts.

1.7.1.5 Increasing Hacking Level

There are many servers besides `foodnstuff` that can be hacked, but they have higher required hacking levels. Therefore, we should raise our hacking level. Not only will this let us hack more servers, but it will also increase the effectiveness of our hacking against `foodnstuff`.

The easiest way to train your hacking level is to visit Rothman University. You can do this by clicking the *City* tab on the left-hand navigation menu, or you can use the *keyboard shortcut* Alt + w. Rothman University should be one of the buttons near the top. Click the button to go to the location.

Once you go to Rothman University, you should see a screen with several options. These options describe different courses you can take. You should click the first button, which says: "Study Computer Science (free)".

After you click the button, you will start studying and earning hacking experience. While you are doing this, you cannot interact with any other part of the game until you click the button that says "Stop taking course".

Right now, we want a hacking level of 10. You need approximately 174 hacking experience to reach level 10. You can check how much hacking experience you have by clicking the *Stats* tab on the left-hand navigation menu, or by using *Keyboard shortcut* Alt + c. Since studying at Rothman University earns you 1 experience per second, this will take 174 seconds, or approximately 3 minutes. Feel free to do something in the meantime!

1.7.1.6 Editing our Hacking Script

Now that we have a hacking level of 10, we can hack the `joesguns` server. This server will be slightly more profitable than `foodnstuff`. Therefore, we want to change our hacking script to target `joesguns` instead of `foodnstuff`.

Go to `Terminal` and edit the hacking script by entering:

```
$ home
$ nano early-hack-template.script
```

At the top of the script, change the *target* variable to be *joesguns*:

```
var target = "joesguns";
```

Note that this will **NOT** affect any instances of the script that are already running. This will only affect instances of the script that are ran from this point forward.

1.7.1.7 Creating a New Script to Purchase New Servers

Next, we're going to create a script that automatically purchases additional servers. These servers will be used to run many scripts. Running this script will initially be very expensive since purchasing a server costs money, but it will pay off in the long run.

In order to create this script, you should familiarize yourself with the following Netscript functions:

- *purchaseServer()*
- *getPurchasedServerCost()*
- *getPurchasedServerLimit()*
- *getServerMoneyAvailable()*
- *scp()*
- *exec()*

Create the script by going to `Terminal` and typing:

```
$ home
$ nano purchase-server-8gb.script
```

Paste the following code into the script editor:

```
// How much RAM each purchased server will have. In this case, it'll
// be 8GB.
var ram = 8;

// Iterator we'll use for our loop
var i = 0;

// Continuously try to purchase servers until we've reached the maximum
// amount of servers
while (i < getPurchasedServerLimit()) {
    // Check if we have enough money to purchase a server
    if (getServerMoneyAvailable("home") > getPurchasedServerCost(ram)) {
        // If we have enough money, then:
        //  1. Purchase the server
        //  2. Copy our hacking script onto the newly-purchased server
        //  3. Run our hacking script on the newly-purchased server with 3 threads
        //  4. Increment our iterator to indicate that we've bought a new server
        var hostname = purchaseServer("pserv-" + i, ram);
        scp("early-hack-template.script", hostname);
        exec("early-hack-template.script", hostname, 3);
        ++i;
    }
}
```

This code uses a while loop to purchase the maximum amount of servers using the *purchaseServer()* Netscript function. Each of these servers will have 8GB of RAM, as defined in the `ram` variable. Note that the script uses the command `getServerMoneyAvailable("home")` to get the amount of money you currently have. This is then used to check if you can afford to purchase a server.

Whenever the script purchases a new server, it uses the *scp()* function to copy our script onto that new server, and then it uses the *exec()* function to execute it on that server.

To run this script, go to `Terminal` and type:

```
$ run purchase-server-8gb.script
```

This purchase will continuously run until it has purchased the maximum number of servers. When this happens, it'll mean that you have a bunch of new servers that are all running hacking scripts against the `joesguns` server!

Note: The reason we're using so many scripts to hack `joesguns` instead of targeting other servers is because it's more effective. This early in the game, we don't have enough RAM to efficiently hack multiple targets, and trying to do so would be slow as we'd be spread too thin. You should definitely do this later on, though!

Note that purchasing a server is fairly expensive, and purchasing the maximum amount of servers even more so. At the time of writing this guide, the script above requires $11 million in order to finish purchasing all of the 8GB servers. Therefore, we need to find additional ways to make money to speed up the process! These are covered in the next section.

1.7.1.8 Additional Sources of Income

There are other ways to gain money in this game besides scripts & hacking.

1.7.1.8.1 Hacknet Nodes

If you completed the introductory tutorial, you were already introduced to this method: Hacknet Nodes. Once you have enough money, you can start upgrading your Hacknet Nodes in order to increase your passive income stream. This is completely optional. Since each Hacknet Node upgrade takes a certain amount of time to "pay itself off", it may not necessarily be in your best interest to use these.

Nonetheless, Hacknet Nodes are a good source of income early in the game, although their effectiveness tapers off later on. If you do wind up purchasing and upgrading Hacknet Nodes, I would suggest only upgrading their levels for now. I wouldn't bother with RAM and Core upgrades until later on.

1.7.1.8.2 Crime

The best source of income right now is from *committing crimes*. This is because it not only gives you a large amount of money, but it also raises your hacking level. To commit crimes, click on the `City` tab on the left-hand navigation menu or use the *Keyboard shortcut* Alt + w. Then, click on the link that says `The Slums`.

In the Slums, you can attempt to commit a variety of crimes, each of which gives certain types of experience and money if successful. See *Crimes* for more details.

Note: You are not always successful when you attempt to commit a crime. Nothing bad happens if you fail a crime, but you won't earn any money and the experience gained will be reduced. Raising your stats improves your chance of successfully committing a crime.

Right now, the best option is the `Rob Store` crime. This takes 60 seconds to attempt and gives $400k if successful. I suggest this crime because you don't have to click or check in too often since it takes a whole minute to attempt. Furthermore, it gives hacking experience, which is very important right now.

Alternatively, you can also use the `Shoplift` crime. This takes 2 seconds to attempt and gives $15k if successful. This crime is slightly easier and is more profitable than `Rob Store`, but it requires constant clicking and it doesn't give hacking experience.

1.7.1.8.3 Work for a Company

If you don't want to constantly check in on the game to commit crimes, there's another option that's much more passive: working for a *company*. This will not be nearly as profitable as crimes, but it's completely passive.

Go to the `City` tab on the left-hand navigation menu and then go to `Joe's Guns`. At `Joe's Guns`, there will be an option that says `Apply to be an Employee`. Click this to get the job. Then, a new option will appear that simply says `Work`. Click this to start working. Working at `Joe's Guns` earns $110 per second and also grants some experience for every stat except hacking.

Working for a company is completely passive. However, you will not be able to do anything else in the game while you work. You can cancel working at any time. You'll notice that cancelling your work early causes you to lose out on some reputation gains, but you shouldn't worry about this. Company reputation isn't important right now.

Once your hacking hits level 75, you can visit `Carmichael Security` in the city and get a software job there. This job offers higher pay and also earns you hacking experience.

There are many more companies in the `City` tab that offer more pay and also more gameplay features. Feel free to explore!

1.7.1.9 After you Purchase your New Servers

After you've made a total of $11 million, your automatic server-purchasing script should finish running. This will free up some RAM on your home computer. We don't want this RAM to go to waste, so we'll make use of it. Go to `Terminal` and enter the following commands:

```
$ home
$ run early-hack-template.script -t 3
```

1.7.1.10 Reaching a Hacking Level of 50

Once you reach a hacking level of 50, two new important parts of the game open up.

1.7.1.10.1 Creating your first program: BruteSSH.exe

On the left-hand navigation menu you will notice a `Create Programs` tab with a red notification icon. This indicates that there are programs available to be created. Click on that tab (or use *Keyboard shortcut* Alt + p) and you'll see a list of all the programs you can currently create. Hovering over a program will give a brief description of its function. Simply click on a program to start creating it.

Right now, the program we want to create is `BruteSSH.exe`. This program is used to open up SSH ports on servers. This will allow you to hack more servers, as many servers in the game require a certain number of opened ports in order for `NUKE.exe` to gain root access.

When you are creating a program, you cannot interact with any other part of the game. Feel free to cancel your work on creating a program at any time, as your progress will be saved and can be picked back up later. `BruteSSH.exe` takes about 10 minutes to complete.

1.7.1.10.2 Optional: Create AutoLink.exe

On the `Create Programs` page, you will notice another program you can create called `AutoLink.exe`. If you don't mind waiting another 10-15 minutes, you should go ahead and create this program. It makes it much less tedious to connect to other servers, but it's not necessary for progressing.

1.7.1.10.3 Joining your first faction: CyberSec

Shortly after you reached level 50 hacking, you should have received a message that said this:

```
Message received from unknown sender:

We've been watching you. Your skills are very impressive. But you're wasting
your talents. If you join us, you can put your skills to good use and change
the world for the better. If you join us, we can unlock your full potential.
But first, you must pass our test. Find and hack our server using the Terminal.

-CyberSec

This message was saved as csec-test.msg onto your home computer.
```

If you didn't, or if you accidentally closed it, that's okay! Messages get saved onto your home computer. Enter the following `Terminal` commands to view the message:

```
$ home
$ cat csec-test.msg
```

This message is part of the game's main "quest-line". It is a message from the `CyberSec` *faction* that is asking you to pass their test. Passing their test is simple, you just have to find their server and hack it through the `Terminal`. Their server is called `CSEC`. To do this, we'll use the *scan-analyze* Terminal command, just like we did before:

```
$ home
$ scan-analyze 2
```

This will show you the network for all servers that are up to 2 "nodes" away from your home computer. Remember that the network is randomly generated so it'll look different for everyone. Here's the relevant part of my `scan-analyze` results:

```
>iron-gym
--Root Access: NO, Required hacking skill: 100
--Number of open ports required to NUKE: 1
--RAM: 32

---->zer0
------Root Access: NO, Required hacking skill: 75
------Number of open ports required to NUKE: 1
------RAM: 32

---->CSEC
------Root Access: NO, Required hacking skill: 54
------Number of open ports required to NUKE: 1
------RAM: 8
```

This tells me that I can reach `CSEC` by going through `iron-gym`:

```
$ connect iron-gym
$ connect CSEC
```

Note: If you created the `AutoLink.exe` program earlier, then there is an easier method of connecting to `CSEC`. You'll notice that in the `scan-analyze` results, all of the server hostnames are white and underlined. You can simply click one of the server hostnames in order to connect to it. So, simply click `CSEC`!

Note: Make sure you notice the required hacking skill for the `CSEC` server. This is a random value between 51 and 60. Although you receive the message from CSEC once you hit 50 hacking, you cannot actually pass their test until your hacking is high enough to hack their server.

After you are connected to the `CSEC` server, you can hack it. Note that this server requires one open port in order to gain root access. We can open the SSH port using the `BruteSSH.exe` program we created earlier. In `Terminal`:

```
$ run BruteSSH.exe
$ run NUKE.exe
$ hack
```

Keep hacking the server until you are successful. After you successfully hack it, you should receive a faction invitation from `CyberSec` shortly afterwards. Accept it. If you accidentally reject the invitation, that's okay. Just go to the `Factions` tab (*Keyboard shortcut* Alt + f) and you should see an option that lets you accept the invitation.

Congrats! You just joined your first faction. Don't worry about doing anything with this faction yet, we can come back to it later.

1.7.1.11 Using Additional Servers to Hack Joesguns

Once you have the `BruteSSH.exe` program, you will be able to gain root access to several additional servers. These servers have more RAM that you can use to run scripts. We'll use the RAM on these servers to run more scripts that target `joesguns`.

1.7.1.11.1 Copying our Scripts

The server's we'll be using to run our scripts are:

- `neo-net`
- `zer0`
- `max-hardware`
- `iron-gym`

All of these servers have 32GB of RAM. You can use the `Terminal` command `scan-analyze 3` to see for yourself. To copy our hacking scripts onto these servers, go to `Terminal` and run:

```
$ home
$ scp early-hack-template.script neo-net
$ scp early-hack-template.script zer0
$ scp early-hack-template.script max-hardware
$ scp early-hack-template.script iron-gym
```

Since each of these servers has 32GB of RAM, we can run our hacking script with 12 threads on each server. By now, you should know how to connect to servers. So find and connect to each of the servers above using the `scan-analyze 3` `Terminal` command. Then, use following `Terminal` command to run our hacking script with 12 threads:

```
$ run early-hack-template.script -t 12
```

Remember that if you have the `NUKE.exe` program, you can simply click on the hostname of a server after running *scan-analyze* to connect to it.

1.7.1.12 Profiting from Scripts & Gaining Reputation with CyberSec

Now it's time to play the waiting game. It will take some time for your scripts to start earning money. Remember that most of your scripts are targeting `joesguns`. It will take a bit for them to *`grow()`* and *`weaken()`* the server to the appropriate values before they start hacking it. Once they do, however, the scripts will be very profitable.

Note: For reference, in about two hours after starting my first script, my scripts had a production rate of $20k per second and had earned a total of $70 million. (You can see these stats on the `Active Scripts` tab).

After another 15 minutes, the production rate had increased to $25k per second and the scripts had made an additional $55 million.

Your results will vary based on how fast you earned money from crime/working/hacknet nodes, but this will hopefully give you a good indication of how much the scripts can earn.

In the meantime, we are going to be gaining reputation with the `CyberSec` *faction*. Go to the `Factions` tab on the left-hand navigation menu, and from there select `CyberSec`. In the middle of the page there should be a button for `Hacking Contracts`. Click it to start earning reputation for the `CyberSec` faction (as well as some hacking experience). The higher your hacking level, the more reputation you will gain. Note that while you are working for a faction, you cannot interact with the rest of the game in any way. You can cancel your faction work at any time with no penalty.

1.7.1.13 Purchasing Upgrades and Augmentations

As I mentioned before, within 1-2 hours I had earned over $200 million. Now, it's time to spend all of this money on some persistent upgrades to help progress!

1.7.1.13.1 Upgrading RAM on Home computer

The most important thing to upgrade right now is the RAM on your home computer. This will allow you to run more scripts.

To upgrade your RAM, go to the `City` tab and visit the company `Alpha Enterprises`. There will be an option that says `Purchase additional RAM for Home Computer`. Click it and follow the dialog box to upgrade your RAM.

I recommend getting your home computer's RAM to *at least* 128GB. Getting it even higher would be better.

1.7.1.13.2 Purchasing your First Augmentations

Once you get ~1000 reputation with the `CyberSec` *faction*, you can purchase your first *Augmentation* from them.

To do this, go to the `Factions` tab on the left-hand navigation menu (*Keyboard shortcut* Alt + f) and select `CyberSec`. There is an button near the bottom that says `Purchase Augmentations`. This will bring up a page that displays all of the Augmentations available from `CyberSec`. Some of them may be locked right now. To unlock these, you will need to earn more reputation with `CyberSec`.

Augmentations give persistent upgrades in the form of multipliers. These aren't very powerful early in the game because the multipliers are small. However, the effects of Augmentations stack multiplicatively **with each other**, so as you continue to install many Augmentations their effects will increase significantly.

Because of this, I would recommend investing more in RAM upgrades for your home computer rather than Augmentations early on. Having enough RAM to run many scripts will allow you to make much more money, and then you can come back later on and get all these Augmentations.

Right now, I suggest purchasing at the very least the `Neurotrainer I` Augmentation from `CyberSec`. If you have the money to spare, I would also suggest getting `BitWire` and several levels of the `NeuroFlux Governor` Augmentations. Note that each time you purchase an Augmentation, *the price of purchasing another increases by 90%*, so make sure you buy the most expensive Augmentation first. Don't worry, once you choose to install Augmentations, their prices will reset back to their original values.

1.7.1.14 Next Steps

That's the end of the walkthrough portion of this guide! You should continue to explore what the game has to offer. There's quite a few features that aren't covered or mentioned in this guide, and even more that get unlocked as you continue to play!

Also, check out the *Netscript* documentation to see what it has to offer. Writing scripts to perform and automate various tasks is where most of the fun in the game comes from (in my opinion)!

The following are a few things you may want to consider doing in the near future.

1.7.1.14.1 Installing Augmentations (and Resetting)

If you've purchased any *Augmentations*, you'll need to install them before you actually gain their effects. Installing Augmentations is the game's "soft-reset" or "prestige" mechanic. You can *read more details about it here*.

To install your Augmentations, click the `Augmentations` tab on the left-hand navigation menu (*Keyboard shortcut* Alt + a). You will see a list of all of the Augmentations you have purchased. Below that, you will see a button that says `Install Augmentations`. Be warned, after clicking this there is no way to undo it (unless you load an earlier save).

1.7.1.14.2 Automating the Script Startup Process

Whenever you install Augmentations, all of your scripts are killed and you'll have to re-run them. Doing this every time you install Augmentations would be very tedious and annoying, so you should write a script to automate the process. Here's a simple example for a startup script. Feel free to adjust it to your liking.

```
// Array of all servers that don't need any ports opened
// to gain root access. These have 16 GB of RAM
var servers0Port = ["foodnstuff",
                    "sigma-cosmetics",
                    "joesguns",
                    "nectar-net",
                    "hong-fang-tea",
                    "harakiri-sushi"];

// Array of all servers that only need 1 port opened
// to gain root access. These have 32 GB of RAM
var servers1Port = ["neo-net",
                    "zer0",
                    "max-hardware",
                    "iron-gym"];

// Copy our scripts onto each server that requires 0 ports
// to gain root access. Then use nuke() to gain admin access and
// run the scripts.
for (var i = 0; i < servers0Port.length; ++i) {
    var serv = servers0Port[i];
```

(continues on next page)

(continued from previous page)

```
    scp("early-hack-template.script", serv);
    nuke(serv);
    exec("early-hack-template.script", serv, 6);
}

// Wait until we acquire the "BruteSSH.exe" program
while (!fileExists("BruteSSH.exe")) {
    sleep(60000);
}

// Copy our scripts onto each server that requires 1 port
// to gain root access. Then use brutessh() and nuke()
// to gain admin access and run the scripts.
for (var i = 0; i < servers1Port.length; ++i) {
    var serv = servers1Port[i];

    scp("early-hack-template.script", serv);
    brutessh(serv);
    nuke(serv);
    exec("early-hack-template.script", serv, 12);
}
```

1.7.1.15 Random Tips

- Early on in the game, it's better to spend your money on upgrading RAM and purchasing new servers rather than spending it on Augmentations
- The more money available on a server, the more effective the `hack()` and `grow()` Netscript functions will be. This is because both of these functions use percentages rather than flat values. `hack()` steals a percentage of a server's total available money, and `grow()` increases a server's money by X%.
- There is a limit to how much money can exist on a server. This value is different for each server. The `getServerMaxMoney()` function will tell you this maximum value.
- At this stage in the game, your combat stats (strength, defense, etc.) are not nearly as useful as your hacking stat. Do not invest too much time or money into gaining combat stat exp.

1.7.2 What BitNode should I do?

Warning: This page contains spoilers regarding the game's story/plot-line.

After destroying their first *BitNode*, many players wonder which BitNode they should tackle next. This guide hopefully helps answer that question.

1.7.2.1 Overview of each BitNode

1.7.2.1.1 BitNode-1: Source Genesis

Description The first BitNode created by the Enders to imprison the minds of humans. It became the prototype and testing-grounds for all of the BitNodes that followed. This is the first BitNode that you play through. It has no

special modifications or mechanics.

Source-File

Max Level 3

This Source-File lets the player start with 32GB of RAM on his/her home computer when entering a new BitNode, and also increases all of the player's multipliers by:

- Level 1: 16%
- Level 2: 24%
- Level 3: 28%

Difficulty The easiest BitNode

1.7.2.1.2 BitNode-2: Rise of the Underworld

Description Organized crime groups quickly filled the void of power left behind from the collapse of Western government in the 2050s. As society and civlization broke down, people quickly succumbed to the innate human impulse of evil and savagery. The organized crime factions quickly rose to the top of the modern world.

In this BitNode:

- Your hacking level is reduced by 20%
- The growth rate and maximum amount of money available on servers are significantly decreased
- The amount of money gained from crimes and Infiltration is tripled
- Certain Factions (Slum Snakes, Tetrads, The Syndicate, The Dark Army, Speakers for the Dead, NiteSec, The Black Hand) give the player the ability to form and manage their own gangs. These gangs will earn the player money and reputation with the corresponding Faction
- Every Augmentation in the game will be available through the Factions listed above
- For every Faction NOT listed above, reputation gains are halved
- You will no longer gain passive reputation with Factions

Source-File

Max Level 3

This Source-File allows you to form gangs in other BitNodes once your karma decreases to a certain value. It also increases the player's crime success rate, crime money, and charisma multipliers by:

- Level 1: 24%
- Level 2: 36%
- Level 3: 42%

Difficulty Fairly easy, as hacking is still very profitable and the costs of various purchases/upgrades is not increased. The gang mechanic may seem strange as its very different from anything else, but it can be very powerful once you get the hang of it.

1.7.2.1.3 BitNode-3: Corporatocracy

Description Our greatest illusion is that a healthy society can revolve around a single-minded pursuit of wealth. Sometime in the early 21st century economic and political globalization turned the world into a corporatocracy, and it never looked back. Now, the privileged elite will happily bankrupt their own countrymen, decimate their

own community, and evict their neighbors from houses in their desperate bid to increase their wealth. In this BitNode you can create and manage your own corporation. Running a successful corporation has the potential of generating massive profits. All other forms of income are reduced by 75%. Furthermore:

- The price and reputation cost of all Augmentations is tripled
- The starting and maximum amount of money on servers is reduced by 75%
- Server growth rate is reduced by 80%
- You now only need 75 favour with a faction in order to donate to it, rather than 150

Source-File

Max Level 3

This Source-File lets you create corporations on other BitNodes (although some BitNodes will disable this mechanic). This Source-File also increases your charisma and company salary multipliers by:

- Level 1: 8%
- Level 2: 12%
- Level 3: 14%

Difficulty Somewhat-steep learning curve as you learn how to use and manage Corporations. Afterwards, however, the BitNode is easy as Corporations can be very profitable.

1.7.2.1.4 BitNode-4: The Singularity

Description The Singularity has arrived. The human race is gone, replaced by artificially superintelligent beings that are more machine than man.

In this BitNode, progressing is significantly harder:

- Experience gain rates for all stats are reduced.
- Most methods of earning money will now give significantly less.

In this BitNode you will gain access to a new set of Netscript Functions known as Singularity Functions. These functions allow you to control most aspects of the game through scripts, including working for factions/companies, purchasing/installing Augmentations, and creating programs.

Source-File

Max Level 3

This Source-File lets you access and use the Singularity Functions in other BitNodes. Each level of this Source-File will open up more Singularity Functions that you can use.

Difficulty: Depending on what Source-Files you have unlocked before attempting this BitNode, it can range from easy to moderate.

1.7.2.1.5 BitNode-5: Artificial Intelligence

Description They said it couldn't be done. They said the human brain, along with its consciousness and intelligence, couldn't be replicated. They said the complexity of the brain results from unpredictable, nonlinear interactions that couldn't be modeled by 1's and 0's. They were wrong.

In this BitNode:

- The base security level of servers is doubled

- The starting money on servers is halved, but the maximum money remains the same
- Most methods of earning money now give significantly less
- Infiltration gives 50% more reputation and money
- Corporations have 50% lower valuations and are therefore less profitable
- Augmentations are more expensive
- Hacking experience gain rates are reduced

Source-File

Max Level 3

This Source-File grants you a special new stat called Intelligence.

Intelligence is unique because it is permanent and persistent (it never gets reset back to 1). However gaining Intelligence experience is much slower than other stats, and it is also hidden (you won't know when you gain experience and how much). Higher Intelligence levels will boost your production for many actions in the game.

In addition, this Source-File will unlock the `getBitNodeMultipliers()` Netscript function, and will also raise all of your hacking-related multipliers by:

- Level 1: 8%
- Level 2: 12%
- Level 3: 14%

Difficulty Depending on what Source-Files you have unlocked before attempting this BitNode, it can range from easy to moderate.

1.7.2.1.6 BitNode-6: Bladeburners

Description In the middle of the 21st century, OmniTek Incorporated began designing and manufacturing advanced synthetic androids, or Synthoids for short. They achieved a major technological breakthrough in the sixth generation of their Synthoid design, called MK-VI, by developing a hyperintelligent AI. Many argue that this was the first sentient AI ever created. This resulted in Synthoid models that were stronger, faster, and more intelligent than the humans that had created them.

In this BitNode you will be able to access the Bladeburner Division at the NSA, which provides a new mechanic for progression. Furthermore:

- Hacking and Hacknet Nodes will be less profitable
- Your hacking level is reduced by 65%
- Hacking experience gain from scripts is reduced by 75%
- Corporations have 80% lower valuations and are therefore less profitable
- Working for companies is 50% less profitable
- Crimes and Infiltration are 25% less profitable

Source-File

Max Level 3

This Source-File allows you to access the NSA's Bladeburner Division in other BitNodes. In addition, this Source-File will raise both the level and experience gain rate of all your combat stats by:

- Level 1: 8%

- Level 2: 12%
- Level 3: 14%

Difficulty Initially difficult due to the fact that hacking is no longer profitable and you have to learn a new mechanic. After you get the hang of the Bladeburner mechanic, however, it becomes moderately easy.

1.7.2.1.7 BitNode-7: Bladeburners 2079

Description In the middle of the 21st century, you were doing cutting-edge work at OmniTek Incorporated as part of the AI design team for advanced synthetic androids, or Synthoids for short. You helped achieve a major technological breakthrough in the sixth generation of the company's Synthoid design, called MK-VI, by developing a hyperintelligent AI. Many argue that this was the first sentient AI ever created. This resulted in Synthoid models that were stronger, faster, and more intelligent than the humans that had created them.

In this BitNode you will be able to access the Bladeburner API, which allows you to access Bladeburner functionality through Netscript. Furthermore:

- The rank you gain from Bladeburner contracts/operations is reduced by 40%
- Bladeburner skills cost twice as many skill points
- Augmentations are 3x more expensive
- Hacking and Hacknet Nodes will be significantly less profitable
- Your hacking level is reduced by 65%
- Hacking experience gain from scripts is reduced by 75%
- Corporations have 80% lower valuations and are therefore less profitable
- Working for companies is 50% less profitable
- Crimes and Infiltration are 25% less profitable

Source-File

Max Level 3

This Source-File allows you to access the Bladeburner Netscript API in other BitNodes. In addition, this Source-File will increase all of your Bladeburner multipliers by:

- Level 1: 8%
- Level 2: 12%
- Level 3: 14%

Difficulty Slightly more difficult than BitNode-6. However, you will be able to automate more aspects of the Bladeburner feature, which means it will be more passive.

1.7.2.1.8 BitNode-8: Ghost of Wall Street

Description You are trying to make a name for yourself as an up-and-coming hedge fund manager on Wall Street.

In this BitNode:

- You start with $250 million
- The only way to earn money is by trading on the stock market
- You start with a WSE membership and access to the TIX API

- You are able to short stocks and place different types of orders (limit/stop)
- You can immediately donate to factions to gain reputation

Source-File

Max Level 3

This Source-File grants the following benefits:

- Level 1: Permanent access to WSE and TIX API
- Level 2: Ability to short stocks in other BitNodes
- Level 3: Ability to use limit/stop orders in other BitNodes

This Source-File also increases your hacking growth multipliers by:

- Level 1: 12%
- Level 2: 18%
- Level 3: 21%

Difficulty Very difficult until you unlock the Four Sigma (4S) Market Data API. After you unlock the API however, it becomes moderately easy.

1.7.2.1.9 BitNode-9: Hacktocracy

Description When Fulcrum Technologies released their open-source Linux distro Chapeau, it quickly became the OS of choice for the underground hacking community. Chapeau became especially notorious for powering the Hacknet, a global, decentralized network used for nefarious purposes. Fulcrum quickly abandoned the project and dissociated themselves from it.

This BitNode unlocks the Hacknet Server, an upgraded version of the Hacknet Node. Hacknet Servers generate hashes, which can be spent on a variety of different upgrades.

In this BitNode: * Your stats are significantly decreased * You cannnot purchase additional servers * Hacking is significantly less profitable

Source-File

Max Level 3

This Source-File grants the following benefits:

- Level 1: Permanently unlocks the Hacknet Server in other BitNodes
- Level 2: You start with 128GB of RAM on your home computer when entering a new BitNode
- Level 3: Grants a highly-upgraded Hacknet Server when entering a new BitNode

(Note that the Level 3 effect of this Source-File only applies when entering a new BitNode, NOT when installing Augmentation

Difficulty Hard

1.7.2.1.10 BitNode-10: Digital Carbon

Description In 2084, VitaLife unveiled to the world the Persona Core, a technology that allowed people to digitize their consciousness. Their consciousness could then be transferred into Synthoids or other bodies by trasmitting the digitized data. Human bodies became nothing more than 'sleeves' for the human consciousness. Mankind had finally achieved immortality - at least for those that could afford it.

This BitNode unlocks Sleeve technology. Sleeve technology allows you to:

1. Re-sleeve: Purchase and transfer your consciousness into a new body
2. Duplicate Sleeves: Duplicate your consciousness into Synthoids, allowing you to perform different tasks synchronously

In this BitNode: * Your stats are significantly decreased * All methods of gaining money are half as profitable (except Stock Market) * Purchased servers are more expensive, have less max RAM, and a lower maximum limit * Augmentations are 5x as expensive and require twice as much reputation

Source-File

Max Level 3

This Source-File unlocks Sleeve technology in other BitNodes. Each level of this Source-File also grants you a Duplicate Sleeve

Difficulty Hard

1.7.2.1.11 BitNode-11: The Big Crash

Description The 2050s was defined by the massive amounts of violent civil unrest and anarchic rebellion that rose all around the world. It was this period of disorder that eventually lead to the governmental reformation of many global superpowers, most notably the USA and China. But just as the world was slowly beginning to recover from these dark times, financial catastrophe hit. In many countries, the high cost of trying to deal with the civil disorder bankrupted the governments. In all of this chaos and confusion, hackers were able to steal billions of dollars from the world's largest electronic banks, prompting an international banking crisis as governments were unable to bail out insolvent banks. Now, the world is slowly crumbling in the middle of the biggest economic crisis of all time.

In this BitNode:

- Your hacking stat and experience gain are halved
- The starting and maximum amount of money available on servers is significantly decreased
- The growth rate of servers is significantly reduced
- Weakening a server is twice as effective
- Company wages are decreased by 50%
- Corporation valuations are 99% lower and are therefore significantly less profitable
- Hacknet Node production is significantly decreased
- Crime and Infiltration are more lucrative
- Augmentations are twice as expensive

Source-File

Max Level 3

Destroying this BitNode will give you Source-File 11, or if you already have this Source-File it will upgrade its level up to a maximum of 3. This Source-File makes it so that company favor increases BOTH the player's salary and reputation gain rate at that company by 1% per favor (rather than just the reputation gain). This Source-File also increases the player's company salary and reputation gain multipliers by:

- Level 1: 32%
- Level 2: 48%

- Level 3: 56%

Difficulty Hard

1.7.2.1.12 BitNode-12: The Recursion

Description Every time this BitNode is destroyed, it becomes slightly harder.

Source-File

Max Level Infinity

Each level of Source-File 12 will increase all of your multipliers by 1%. This effect is multiplicative with itself. In other words, level N of this Source-File will result in a multiplier of 1.01^N (or 0.99^N for multipliers that decrease)

Difficulty Initially very easy, but then it (obviously) becomes harder as you continue to do it.

1.7.2.2 Recommended BitNodes

As a player, you are not forced to tackle the BitNodes in any particular order. You are free to choose whichever ones you want. The "best" order can vary between players, depending on what you like to do any what kind of player you are. In general, here are the recommended BitNodes for different things:

1.7.2.2.1 For fast progression

Note: This does not recommend the absolute fastest path, as I don't know what exactly the fastest path is. But it does recommend the BitNodes that are commonly considered to be optimal by players.

---

1. Repeat **BitNode-1: Source Genesis** until you max out its Source-File. Its Source-File is extremely powerful, as it raises all multipliers by a significant amount.
2. Repeat **BitNode-12: The Recursion** several times. This BitNode will be extremely easy the first few times you tackle it, and its Source-File raises all multipliers. Furthermore, its effect stacks multiplicatively with itself and other Source-Files/Augmentations, which gets better as time goes on
3. Do **BitNode-5: Artificial Intelligence** once or twice. The intelligence stat it unlocks will gradually build up as you continue to play the game, and will be helpful in the future. The Source-File also provides hacking multipliers, which are strong because hacking is typically one of the best ways of earning money.
4. (Optional) Consider doing **BitNode-4: The Singularity**. Its Source-File does not directly make you more powerful in any way, but it does unlock *Netscript Singularity Functions* which let you automate significantly more aspects of the game.
5. Do **BitNode-3: Corporatocracy** once to unlock the Corporation mechanic. This mechanic has high profit potential.
6. Do **BitNode-6: Bladeburners** once to unlock the Bladeburners mechanic. The Bladeburner mechanic is useful for some of the future BitNodes (such as 9 and 10).
7. Do **BitNode-9: Hacktocracy** to unlock the Hacknet Server mechanic. You can consider repeating it as well, as its Level 2 and 3 effects are pretty helpful as well.

Todo: To be continued as more BitNodes get added

1.7.2.2.2 For the strongest Source-Files

Note that the strongest Source-Files are typically rewarded by the hardest BitNodes.

The strongest Source-File is that from **BitNode-1: Source Genesis**, as it raises all multipliers by a significant amount.

Similarly, the Source-File from **BitNode-12: The Recursion** is also very strong because it raises all multipliers. Each level of Source-File 12 is fairly weak, but its effectiveness gets better over time since the effects of Source-Files and Augmentations are multiplicative with each other.

The Source-File from **BitNode-9: Hacktocracy** is good because it unlocks the Hacknet Server mechanic. The Hacknet Server mechanic causes Hacknet Nodes to produce a new currency called *hashes*, rather than money. *Hashes* can be spent on powerful upgrades that benefit your hacking, Corporation, Bladeburner, etc.

The Duplicate Sleeves granted by the Source-File from **BitNode-10: Digital Carbon** are strong, but only after you have several of them and have spent some time/money upgrading them.

1.7.2.2.3 For more scripting/hacking

BitNode-4: The Singularity unlocks the *Netscript Singularity Functions*, which can be used to automate many different aspects of the game, including working for factions/companies, purchasing & installing Augmentations, and creating programs

BitNode-6 and **BitNode-7** unlock Bladeburner and its corresponding *Netscript API*. This allows you to automate an entire new mechanic.

BitNode-2: Rise of the Underworld also unlocks a new mechanic and Netscript API for automating it (the Gang mechanic). However, it is not as interesting as Bladeburner (in my opinion)

BitNode-9: Hacktocracy unlocks the Hacknet Server mechanic and several new functions in the *Hacknet Node API* for using it.

1.7.2.2.4 For new mechanics

BitNode-2: Rise of the Underworld unlocks a new mechanic in which you can manage a gang. Gangs earn you money and can be very profitable once they get large and powerful. The biggest benefit of gangs, however, is that they make all Augmentations available to you through their corresponding faction.

BitNode-3: Corporatocracy unlocks a new mechanic in which you can manage a corporation. You can earn money through Corporations by selling your stocks, or by configuring your corporation to pay dividends to shareholders. If your Corporation gets big enough, it can also bribe factions in exchange for faction reputation.

BitNode-6: Bladeburners unlocks a new mechanic that centers around combat rather than hacking. The main benefit of the Bladeburner mechanic is that it offers a new method of destroying a BitNode.

BitNode-9: Hacktocracy unlocks the Hacknet Server, which is an upgraded version of a Hacknet Node. The Hacknet Server generates a computational unit called a *hash*. *Hashes* can be spent on a variety of different upgrades that can benefit your hacking, Corporation, Bladeburner progress, and more. It transforms the Hacknet Node from a simple money-generator to a more interesting mechanic.

BitNode-10: Digital Carbon unlocks two new mechanics: Re-Sleeving and Duplicate Sleeves.

1.7.2.2.5 For a Challenge

In general, the higher BitNodes are more difficult than the lower ones. **BitNode-12: The Recursion** is an obvious exception as it gets progressively harder.

BitNode-8: Ghost of Wall Street provides a unique challenge as the only method of earning money in that BitNode is through trading at the stock market.

1.8 Tools & Resources

1.8.1 Official Script Repository

There are plans to create an official repository of Bitburner scripts. As of right now, this is not a priority and has not been started. However, if you'd like to contribute scripts now, you can find the repository here and submit pull requests.

1.8.2 Visual Studio Code Extension

One user created a Bitburner extension for the Visual Studio Code (VSCode) editor.

This extension includes several features such as:

- Dynamic RAM calculation
- RAM Usage breakdown
- Typescript definition files with jsdoc comments
- Netscript syntax highlighting

You can find more information and download links on this reddit post.

1.9 Changelog

1.9.1 v0.47.2 - 7/15/2019

Netscript Changes

- Added tail() Netscript function
- hacknet.getNodeStats() function now returns an additional property for Hacknet Servers: hashCapacity
- When writing to a file, the write() function now casts the data being written to a string (using String())
- BitNode-selection page now shows what Source-File level you have for each BitNode
- Overloaded kill() function so that you can kill a script by its PID
- spawn() now only takes 10 seconds to run (decreased from 20 seconds)
- **run() and exec() now return the PID of the newly-executed scripts, rather than a boolean**
  - (A PID is just a positive integer)
- run(), exec(), and spawn() no longer need to be await-ed in NetscriptJS
- Script parsing and RAM calculations now support ES9
- installAugmentations() no longer has a return value since it causes all scripts to die
- isBusy() now returns true if you are in a Hacking Mission
- Bug fix: workForFaction() function now properly accounts for disabled logs

- Bug fix: RAM should now be properly calculated when running a callback script with installAugmentations()
- Bug fix: Fixed bug that caused scripts killed by exit()/spawn() to "clean up" twice

Misc Changes

- The 'kill' Terminal command can now kill a script by its PID
- Added 'Solarized Dark' theme to CodeMirror editor
- After Infiltration, you will now return to the company page rather than the city page
- Bug fix: Stock Market UI should no longer crash for certain locale settings
- Bug fix: You can now properly remove unfinished programs (the *.exe-N%-INC files)
- Bug fix: Fixed an issue that allowed you to increase money on servers with a 'maxMoney' of 0 (like CSEC)
- Bug fix: Scripts no longer persist if they were started with syntax/import errors
- Bug fix: 'hack' and 'analyze' Terminal commands are now blocking
- Bug fix: Exp earned by duplicate sleeves at universities/gyms now takes hash upgrades into account

1.9.2 v0.47.1 - 6/27/2019

- **Stock Market changes:**
  - Transactions no longer influence stock prices (but they still influence forecast)
  - Changed the way stocks behave, particularly with regard to how the stock forecast occasionally "flips"
  - Hacking & growing a server can potentially affect the way the corresponding stock's forecast changes
  - Working for a company positively affects the way the corresponding stock's forecast changes
- Scripts now start/stop instantly
- Improved performance when starting up many copies of a new NetscriptJS script (by Ornedan)
- Improved performance when killing scripts
- Dialog boxes can now be closed with the ESC key (by jaguilar)
- NetscriptJS scripts should now be "re-compiled" if their dependencies change (by jaguilar)
- write() function should now properly cause NetscriptJS scripts to "re-compile" (by jaguilar)

1.9.3 v0.47.0 - 5/17/2019

- **Stock Market changes:**
  - Implemented spread. Stock's now have bid and ask prices at which transactions occur
  - Large transactions will now influence a stock's price and forecast
  - This "influencing" can take effect in the middle of a transaction
  - See documentation for more details on these changes
  - Added getStockAskPrice(), getStockBidPrice() Netscript functions to the TIX API
  - Added getStockPurchaseCost(), getStockSaleGain() Netscript functions to the TIX API
- **Re-sleeves can no longer have the NeuroFlux Governor augmentation**
  - This is just a temporary patch until the mechanic gets re-worked

- hack(), grow(), and weaken() functions now take optional arguments for number of threads to use (by MasonD)
- codingcontract.attempt() now takes an optional argument that allows you to configure the function to return a contract's reward
- Adjusted RAM costs of Netscript Singularity functions (mostly increased)
- Adjusted RAM cost of codingcontract.getNumTriesRemaining() Netscript function
- Netscript Singularity functions no longer cost extra RAM outside of BitNode-4
- Corporation employees no longer have an "age" stat
- Gang Wanted level gain rate capped at 100 (per employee)
- Script startup/kill is now processed every 3 seconds, instead of 6 seconds
- getHackTime(), getGrowTime(), and getWeakenTime() now return Infinity if called on a Hacknet Server
- Money/Income tracker now displays money lost from hospitalizations
- Exported saves now have a unique filename based on current BitNode and timestamp
- Maximum number of Hacknet Servers decreased from 25 to 20
- Bug Fix: Corporation employees stats should no longer become negative
- Bug Fix: Fixed sleeve.getInformation() throwing error in certain scenarios
- Bug Fix: Coding contracts should no longer generate on the w0r1d_d43m0n server
- Bug Fix: Duplicate Sleeves now properly have access to all Augmentations if you have a gang
- Bug Fix: getAugmentationsFromFaction() & purchaseAugmentation() functions should now work properly if you have a gang
- Bug Fix: Fixed issue that caused messages (.msg) to be sent when refreshing/reloading the game
- Bug Fix: Purchasing hash upgrades for Bladeburner/Corporation when you don't actually have access to those mechanics no longer gives hashes
- Bug Fix: run(), exec(), and spawn() Netscript functions now throw if called with 0 threads
- Bug Fix: Faction UI should now automatically update reputation
- Bug Fix: Fixed purchase4SMarketData()
- Bug Fix: Netscript1.0 now works properly for multiple 'namespace' imports (import * as namespace from "script")
- Bug Fix: Terminal 'wget' command now correctly evaluates directory paths
- Bug Fix: wget(), write(), and scp() Netscript functions now fail if an invalid filepath is passed in
- Bug Fix: Having Corporation warehouses at full capacity should no longer freeze game in certain conditions
- Bug Fix: Prevented an exploit that allows you to buy multiple copies of an Augmentation by holding the 'Enter' button
- Bug Fix: gang.getOtherGangInformation() now properly returns a deep copy
- Bug Fix: Fixed getScriptIncome() returning an undefined value
- Bug Fix: Fixed an issue with Hacknet Server hash rate not always updating

1.9.4 v0.46.3 - 4/20/2019

- Added a new Augmentation: The Shadow's Simulacrum
- Improved tab autocompletion feature in Terminal so that it works better with directories
- Bug Fix: Tech vendor location UI now properly refreshed when purchasing a TOR router
- Bug Fix: Fixed UI issue with faction donations
- Bug Fix: The money statistics & breakdown should now properly track money earned from Hacknet Server (hashes -> money)
- Bug Fix: Fixed issue with changing input in 'Minimum Path Sum in a Triangle' coding contract problem
- Fixed several typos in various places

1.9.5 v0.46.2 - 4/14/2019

- **Source-File 2 now allows you to form gangs in other BitNodes when your karma reaches a very large negative value**
  - (Karma is a hidden stat and is lowered by committing crimes)
- **Gang changes:**
  - Bug Fix: Gangs can no longer clash with themselve
  - Bug Fix: Winning against another gang should properly reduce their power
- Bug Fix: Terminal 'wget' command now works properly
- Bug Fix: Hacknet Server Hash upgrades now properly reset upon installing Augs/switching BitNodes
- Bug Fix: Fixed button for creating Corporations

1.9.6 v0.46.1 - 4/12/2019

- **Added a very rudimentary directory system to the Terminal**
  - Details here: https://bitburner.readthedocs.io/en/latest/basicgameplay/terminal.html#filesystem-directories
- Added numHashes(), hashCost(), and spendHashes() functions to the Netscript Hacknet Node API
- 'Generate Coding Contract' hash upgrade is now more expensive
- 'Generate Coding Contract' hash upgrade now generates the contract randomly on the server, rather than on home computer
- The cost of selling hashes for money no longer increases each time
- Selling hashes for money now costs 4 hashes (in exchange for $1m)
- Bug Fix: Hacknet Node earnings should work properly when game is inactive/offline
- Bug Fix: Duplicate Sleeve augmentations are now properly reset when switching to a new BitNode

1.9.7 v0.46.0 - 4/3/2019

- Added BitNode-9: Hacktocracy
- Changed BitNode-11's multipliers to make it slightly harder overall
- Source-File 11 is now slightly stronger
- Added several functions to Netscript Sleeve API for buying Sleeve augmentations (by hydroflame)
- Added a new stat for Duplicate Sleeves: Memory
- Increase baseline experience earned from Infiltration, but it now gives diminishing returns (on exp) as you get to higher difficulties/levels
- In Bladeburner, stamina gained from Hyperbolic Regeneration Chamber is now a percentage of your max stamina
- **Corporation Changes:**
  - 'Demand' value of products decreases more slowly
  - Bug Fix: Fixed a Corporation issue that broke the Market-TA2 Research
  - Bug Fix: Issuing New Shares now works properly
- Bug Fix: Money Statistics tracker was incorrectly recording profits when selling stocks manually
- Bug Fix: Fixed an issue with the job requirement tooltip for security jobs

1.9.8 v0.45.1 - 3/23/2019

- Added two new Corporation Researches
- General UI improvements (by hydroflame and koriar)
- Bug Fix: Sleeve Netscript API should no longer cause Dynamic RAM errors
- Bug Fix: sleeve.getSleeveStats() should now work properly

1.9.9 v0.45.0 - 3/22/2019

- **Corporation changes:**
  - Decreased the time of a full market cycle from 15 seconds to 10 seconds.
  - This means that each Corporation 'state' will now only take 2 seconds, rather than 3
  - Increased initial salaries for newly-hired employees
  - Increased the cost multiplier for upgrading office size (the cost will increase faster)
  - The stats of your employees now has a slightly larger effect on production & sales
  - Added several new Research upgrades
  - Market-TA research now allows you to automatically set sale price at optimal values
  - Market-TA research now works for Products (not just Materials)
  - Reduced the amount of Scientific Research needed to unlock the Hi-Tech R&D Laboratory from 10k to 5k
  - Energy Material requirement of the Software industry reduced from 1 to 0.5

  - It is now slightly easier to increase the Software industry's production multiplier
  - Industries now have a maximum number of allowed products, starting at 3. This can be increased through research.
  - You can now see an approximation of how each material affects an industry's production multiplier by clicking the "?" help tip next to it
  - Significantly changed the effects of the different employee positions. See updated descriptions
  - Reduced the amount of money you gain from private investors
  - Training employees is now 3x more effective
  - Bug Fix: An industry's products are now properly separated between different cities
- The QLink Augemntation is now significantly stronger, but also significantly more expensive (by hydroflame)
- Added a Netscript API for Duplicate Sleeves (by hydroflame)
- Modified the multipliers of BitNode-3 and BitNode-8 to make them slightly harder
- After installing Augmentations, Duplicate Sleeves will now default to Synchronize if their Shock is 0
- Bug Fix: Bladeburner's Hyperbolic Regeneration Chamber should no longer instantly refill all stamina
- Bug Fix: growthAnalyze() function now properly accounts for BitNode multipliers
- Bug Fix: The cost of purchasing Augmentations for Duplicate Sleeves no longer scales with how many Augs you've purchased for yourself

1.9.10 v0.44.1 - 3/4/2019

- **Duplicate Sleeve changes:**
  - You can now purchase Augmentations for your Duplicate Sleeves
  - Sleeves are now assigned to Shock Recovery task by default
  - Shock Recovery and Synchronize tasks are now twice as effective
- Changed documentation so that Netscript functions are own their own pages. Sorry if this is annoying, it was necessary for properly cross-referencing
- Officially deprecated the Wiki (the fandom site). Use the 'readthedocs' Documentation instead
- Bug Fix: 'rm' Terminal and Netscript commands now work on non-program files that have '.exe' in the name (by Github user MasonD)
- Bug Fix: The 'Find All Valid Math Expressions' Coding Contract should now properly ignore whitespace in answers
- Bug Fix: The 'Merge Overlapping Intervals' Coding Contract should now properly accept 2D arrays when being attempted through Netscript

1.9.11 v0.44.0 - 2/26/2019

- **Bladeburner Changes:**
  - Reduced the amount of rank needed to earn a skill point
  - Reduced the effects of the "Reaper" and "Evasive System" skills
  - Increased the effect of the "Hyperdrive" and "Hands of Midas" skills

  - Slightly increased the rate which the skill point cost rises for almost all skills
  - The "Overlock" Skill now has a maximum level of 90 instead of 95
  - Money earned from Contracts increased by 400%
  - Changed the way population affects success rate. Extreme populations now have less dramatic effects
  - Added two new General Actions: Diplomacy and Hyperbolic Regeneration Chamber
  - Lowered the rep and money cost of the "Blade's Simulacrum" augmentation
  - Significantly decreased the initial amount of Contracts/Operations (the "Contracts/Operations remaining" value)
  - Decreased the rate at which the amount of Contracts/Operations increases over time
  - Decreased the number of successes you need to increase the max level of a Contract/Operation
  - Increased the average number of Synthoid communities each city has
  - Reduced the amount by which a successful raid will decrease the population of a city
  - The "riots" event will now increase the chaos of a city by a greater amount
  - Significantly increased the effect that Agility and Dexterity have on action time
- **Added new BitNode multipliers:**
  - HomeComputerRamCost - Affects how much it costs to upgrade home computer's RAM
  - DaedalusAugsRequirement - Affects how many Augmentations you need in order to get invited to Daedalus
  - FourSigmaMarketDataCost - Affects how much it costs to unlock the stock market's 4S Market Data
  - FourSigmaMarketDataApiCost - Affects how much it costs to unlock the stock market's 4S Market Data API
- A few minor changes to BitNode multipliers across the board (mostly for the new multipliers)
- 'The Covenant' now requires 20 total Augmentations to get invited, rather than 30
- You can now purchase permanent Duplicate Sleeves from 'The Covenant'. This requires Source-File 10, and you must be in BN-10 or after
- You can now track where all of your money comes from in the 'Stats' page
- Increased the money gained from Coding Contracts by 50%
- getCharacterInformation() function now returns the player's HP and max HP
- Bug Fix: You can no longer disconnect the enemy's connections in Hacking Missions
- Bug Fix: Duplicate Sleeve faction reputation gain is now properly affected by faction favor
- Bug Fix: After installing Augmentations, the Terminal display will now correctly show the current server as "home"
- Bug Fix: Fixed an exploit where you could change the duration of timed functions (e.g. hack, weaken) in NetscriptJS
- Bug Fix: You should now properly be able to use the ServerProfile.exe program
- Bug Fix: Prevented exploit that allowed you to accept faction invites programmatically through NetscriptJS
- Bug Fix: Faction invitations for megacorporations should now work properly

1.9.12 v0.43.1 - 2/11/2019

- **Terminal changes:**
  - Quoted arguments are now properly parsed. (e.g. 'run f.script "this is one argument"' will be correctly parsed)
  - Errors are now shown in red text
  - 'unalias' command now has a different format and no longer needs the quotations
  - Bug Fix: Fixed several edge cases where autocomplete wasn't working properly
- Added two new Bladeburner skills for increasing money and experience gain
- Made some minor adjustments to Bladeburner UI
- Corporation "Smart Factories" and "Smart Storage" upgrades have slightly lower price multipliers
- Added nFormat Netscript function
- Added 6 new Coding Contract problems
- Updated documentation with list of all Coding Contract problems
- Minor improvements for 'Active Scripts' UI
- Implemented several optimizations for active scripts. The game should now use less memory and the savefile should be slightly smaller when there are many scripts running
- Bug Fix: A Stock Forecast should no longer go above 1 (i.e. 100%)
- Bug Fix: The cost of Resleeves should no longer be affected by buying Augs
- Bug Fix: Duplicate Sleeves now use their own stats to determine crime success rate, instead of the host consciousness' stats
- Bug Fix: You can now call the prompt() Netscript function from multiple scripts simultaneously

1.9.13 v0.43.0 - 2/4/2019

- Added BitNode-10: Digital Carbon
- **Stock Market Changes:**
  - Each stock now has a maximum number of shares you can purchase (both Long and Short positions combined)
  - Added getStockMaxShares() Netscript function to the TIX API
  - The cost of 4S Market Data TIX API Access increased from $20b to $25b
- **Job Changes:**
  - You can now hold multiple jobs at once. This means you no longer lose reputation when leaving a company
  - Because of this change, the getCharacterInformation() Netscript function returns a slightly different value
- **Script Editor Changes:**
  - Added new script editor: CodeMirror. You can choose between the old editor (Ace) or CodeMirror
  - Navigation keyboard shortcuts no longer work if the script editor is focused

- Trying to programmatically run a script (run(), exec()) with a 'threads' argument of 0 will now cause the function to return false without running the script
- Home Computer RAM is now capped at 2 ^ 30 GB (1073741824 GB)
- The maximum amount, maximum RAM, and cost of purchasing servers can now vary between different BitNodes (new BitNode multipliers)
- Pop-up dialog boxes are a little bit bigger
- Bug Fix: When importing scripts, "./" will now be properly ignored (e.g. import { foo } from "./lib.script")

1.9.14 v0.42.0 - 1/8/2019

- **Corporation Changes:**
  - Corporation can now be self-funded with $150b or using seed money in exchange for 500m newly-issued shares
  - In BitNode-3, you no longer start with $150b
  - Changed initial market prices for many materials
  - Changed the way a material's demand, competition, and market price change over time
  - The sale price of materials can no longer be marked-up as high
  - Added a Research Tree mechanic. Spend Scientific Research on permanent upgrades for each industry
  - You can now redistribute earnings to shareholders (including yourself) as dividends
  - Cost of "Smart Supply" upgraded reduced from $50b to $25b
  - Now has offline progress, which works similarly to the Gang/Bladeburner mechanics
  - Slightly reduced the amount of money offered to you by investment firms
  - Employee salaries now slowly increase over time
  - Slightly reduced the effect "Real Estate" has on the Production Multiplier for the Agriculture industry
  - Changed the way your Corporation's value is calculated (this is what determines stock price)
  - After taking your corporation public, it is now possible to issue new shares to raise capital
  - Issuing new shares can only be done once every 12 hours
  - Buying back shares must now be done at a premium
  - Selling shares can now only be done once per hour
  - Selling large amounts of shares now immediately impacts stock price (during the transaction)
  - Reduced the initial cost of the DreamSense upgrade from $8b to $4b, but increased its price multiplier
  - Reduced the price multiplier for ABC SalesBots upgrade
- Added getOrders() Netscript function to the TIX API
- Added getAugmentationPrereq() Singularity function (by havocmayhem)
- Added hackAnalyzePercent() and hackAnalyzeThreads() Netscript functions
- Stock Market, Travel, and Corporation main menu links are now properly styled
- Many pop-up/dialog boxes now support the 'Enter' and 'Esc' hotkeys. If you find a pop-up/dialog box that doesnt support this, let me know specifically which one ('Enter' for the default option, 'Esc' for cancelling and closing the pop-up box)

- Added "brace_style = preserve_inline" configuration to Script Editor Beautifier
- ServerProfiler.exe can now be purchased from the Dark Web
- Added an option to copy save data to clipboard
- Added total multiplier information on the "Augmentations" page
- Bug Fix: gymWorkout() Singularity function should now work properly with Millenium Fitness Gym
- Began migrating gameplay information to the ReadTheDocs documentation

1.9.15 v0.41.2 - 11/23/2018

- **IMPORTANT - Netscript Changes:**
  - rm() now takes an optional parameter that lets you specify on which server to delete the file
  - Added growthAnalyze() Netscript function
- **Gang Changes:**
  - UI now displays your chance to win a clash with other gangs
  - Added getChanceToWinClash() function to the Gang API
  - Added getEquipmentType() function to the Gang API
  - Added several new hacking-based equipment and Augmentations
  - Rebalanced several equipment/upgrades to give less defense
  - Wanted level gain rate is now be slightly higher for all tasks
  - Rebalanced parameters for "hacking" tasks
- Added new Main Menu configuration in .fconf: "compact"
- Added the terminal command 'expr', which can be used to evaluate simple mathematical expressions
- Bug Fix: Can no longer purchase duplicate equipment/Augmentations through gang.purchaseEquipment()
- Bug Fix: scp() should no longer throw errors when used with 2-arguments and an array of files
- Bug Fix: Coding Contracts no longer give money in BitNode-8
- Bug Fix: In Bladeburner, you can no longer start a BlackOp through the Netscript API if it has already been completed
- Bug Fix: In Bladeburner, fixed a bug which caused the configured 'automate' actions to occasionally be switched to other actions
- Bug Fix: 'Return to World' button at locations no longer accumulates event listeners
- Bug Fix: Working & taking classes now continuously add/subtract money during the action, instead of doing it at completion
- Bug Fix: Top-right overview panel now displays negative money using '-' instead of '()'
- Bug Fix: Stock Market UI should no longer show 'NaN' profit immediately after buying a stock

1.9.16 v0.41.1 - 11/5/2018

- **IMPORTANT - Netscript Changes:**
  - purchaseTor() now returns true if you already have a TOR router (it used to return false)
  - getPurchasedServerCost() now returns Infinity if the specified RAM is an invalid amount or is greater than the max amount of RAM (2 ^ 20 GB)
  - Added purchase4SMarketData() and purchase4SMarketDataTixApi() functions
  - getScriptLogs() now takes in optional arguments that let you get the logs of another script
- **Stock Market changes:**
  - Stocks now have "maximum prices". These are hidden from the player
  - If a stock reaches its "maximum price", it will most likely drop in value (although it might still rise)
  - Each stock has its own, unique maximum price
  - Maximum price for each stock are randomly generated and change during each 'reset'
  - Stock Market cycles are now accumulated/stored, much like it is for Gangs and Bladeburners
  - **Accumulated/stored cycles cause stock prices to update up to 50% faster (from every 6 seconds to 4 seconds)**
    - This means that after coming back from being offline, stock prices will update faster to make up for offline time
- Decreased the Hacking Level multiplier for BitNodes 6 and 7 to 0.4 (from 0.5)
- Bladeburner console history is now saved and persists when switching screens or closing/reopening the game
- In Bladeburner, if your stamina reaches 0 your current action will be cancelled
- b1t_flum3.exe is no longer removed from your home computer upon reset
- Added main menu link for the Stock Market (once you've purchased an account)
- Job main menu link only appears if you actually have a job
- Bug Fix: Netscript Gang API functions purchaseEquipment() and ascendMember() should now work properly
- Bug Fix: After installing Augs, the "Portfolio Mode" button on the Stock Market page should be properly reset
- Bug Fix: bladeburner.getActionCountRemaining()'s return value is now rounded down (by Kline-)

1.9.17 v0.41.0 - 10/29/2018

- WARNING: In NetscriptJS, defining a function called print() is no longer possible
- **Gang Mechanic Changes (BitNode-2):**
  - Added a Gang Netscript API
  - Added new 'ascension' mechanic for Gang Members
  - The first three gang members are now 'free' (can be recruited instantly)
  - Maximum number of increased Gang Members increased from 20 to 30
  - Changed the formula for calculating respect needed to recruit the next gang member
  - Added a new category of upgrades for Gang Members: Augmentations

  - Non-Augmentation Gang member upgrades are now significantly weaker
  - Reputation for your Gang faction can no longer be gained through Infiltration
  - Re-worked the territory 'warfare' mechanic so that player can choose when to engage in it
  - Gang Members can now be killed during territory 'warfare'
  - Changed BitNode-2 Multipliers to make hacking slightly less profitable
  - Gang Member Equipment + Upgrades now get cheaper as your gang grows in power and respect
  - The effects of Source-File 2 are now slightly more powerful
- RAM Cost of accessing the global document object lowered from 100 GB to 25 GB
- RAM Cost to use Singularity Functions outside of BitNode-4 lowered by 75%. They now only cost twice as much as they do in BitNode-4
- b1t_flum3.exe now takes significantly less time to create
- Crimes commited through Singularity function no longer give half money/exp (there is now no penalty)
- Improved number formatting for Player 'work' actions (including crimes, etc.). These numbers should also adhere to locale settings now (by Kline-)
- The order that Augmentations are listed in (when purchasing from Faction and viewing your Augmentations) is now saved and persists when choosing different orders
- getCharacterInformation() Singularity function now returns multiplier information (from Augmentations/Source Files)
- Bug Fix: Calling print() in NetscriptJS no longer brings up the print dialog
- Bug Fix: Fixed a bug that sometimes caused a blank black screen when destroying/resetting/switching BitNodes
- Bug Fix: Netscript calls that throw errors will now no longer cause the 'concurrent calls' error if they are caught in the script. i.e. try/catch should now work properly in scripts
- Bug Fix: Fixed a bug where sometimes the NeuroFlux Governor Augmentation level would be incorrectly calculated when the game was loaded
- Bug Fix: Fixed a bug where calling the scp() Netscript function with invalid hostname/ips would throw an unclear error message
- Bug Fix: Bladeburner API function getActionCountRemaining() should now work properly for BlackOps
- Bug Fix: Black Ops can no longer be attempted out-of-order or without the required rank via Bladeburner API
- Bug Fix: Dynamic RAM Calculation now properly accounts for number of threads
- RAM cost for basic Netscript functions added to documentation (by CBJamo)

1.9.18 v0.40.5 - 10/09/2018

- Added codingcontract.getContractType() Netscript function
- Bug Fix: codingcontract.getData() Netscript function now returns arrays by value rather than reference
- Bug Fix: Decreased highest possible data value for 'Find Largest Prime Factor' Coding Contract (to avoid hangs when solving it)
- Bug Fix: Fixed a bug that caused game to freeze during Coding Contract generation

1.9.19 v0.40.4 - 9/29/2018

- Added new Coding Contracts mechanic. Solve programming problems to earn rewards
- The write() and read() Netscript functions now work on scripts
- Added getStockSymbols() Netscript function to the TIX API (by InfraK)
- Added wget() Netscript function
- Added bladeburner.getActionRepGain() function to the Netscript Bladeburner API
- The getLevelUpgradeCost(), getRamUpgradeCost(), and getCoreUpgradeCost() functions in the Hacknet API now return Infinity if the node is at max level. See documentation
- It is now possible to use freely use angled bracket (<, >) and create DOM elements using tprint()
- The game's theme colors can now be set through the Terminal configuration (.fconf).
- You can now switch to the old left-hand main menu bar through the Terminal configuration (.fconf)
- Bug Fix: grow() percentage is no longer reported as Infinity when a server's money is grown from 0 to X
- Bug Fix: Infiltration popup now displays the correct amount of exp gained

1.9.20 v0.40.3 - 9/15/2018

- **Bladeburner Changes:**
  - Increased the effect that agi and dexterity have on action time
  - Starting number of contracts/operations available will be slightly lower
  - Random events will now happen slightly more often
  - Slightly increased the rate at which the Overclock skill point cost increases
- The maximum volatility of stocks is now randomized (randomly generated within a certain range every time the game resets)
- Increased the range of possible values for initial stock prices
- b1t_flum3.exe program can now be created immediately at Hacking level 1 (rather than hacking level 5)
- UI improvements for the character overview panel and the left-hand menu (by mat-jaworski)
- General UI improvements for displays and Terminal (by mat-jaworski)
- Added optional parameters to the getHackTime(), getGrowTime(), and getWeakenTime() Netscript functions
- Added isLogEnabled() and getScriptLogs() Netscript functions
- Added donateToFaction() Singularity function
- Updated documentation to reflect the fact that Netscript port handles (getPortHandle()) only works in NetscriptJS (2.0), NOT Netscript 1.0
- Added tryWrite() Netscript function
- When working (for a company/faction), experience is gained immediately/continuously rather than all at once when the work is finished
- Added a setting in .fconf for enabling line-wrap in the Terminal input
- Adding a game option for changing the locale that most numbers are displayed in (this mostly applies for whenever money is displayed)

- The randomized parameters of many high-level servers can now take on a higher range of values
- Many 'foreign' servers (hackable servers that you don't own) now have a randomized amount of RAM
- Added 'wget' Terminal command
- Improved the introductory tutorial

1.9.21 v0.40.2 - 8/27/2018

- **Bladeburner Changes:**
  - Added getBonusTime(), getSkillUpgradeCost(), and getCity() Netscript functions to the API
  - Buffed the effects of many Bladeburner Augmentations
  - The Blade's Simulacrum Augmentation requires significantly less reputation but slightly more money
  - Slightly increased the amount of successes needed for a Contract/Operation in order to increase its max level
  - Increased the amount of money gained from Contracts by ~25%
  - Increased the base amount of rank gained from Operations by 10%
  - Significantly increased the 'randomness' in determining a Contract/Operation's initial count and rate of count increase
  - The number (count) of Operations should now increase significantly faster
  - There are now, on average, more Synthoid communities in a city
  - If automation is enabled (the feature in Bladeburner console), then switching to another action such as working for a company will now disable the automation
- **Stock Market Changes:**
  - Added a watchlist filter feature to the UI that allows you to specify which stocks to show
  - Added the Four Sigma (4S) Market Data feed, which provides volatility and price forecast information about stocks
  - Added the 4S Market Data TIX API, which lets you access the aforementioned data through Netscript
- There is now a setting for enabling/disabling the popup that appears when you are hospitalized
- Bug Fix: Stock market should now be correctly initialized in BitNode-8 (by Kline-)
- Bug Fix: bladeburner.getCurrentAction() should now properly an 'Idle' object rather than null (by Kline-)
- Bug Fix: Bladeburner skill cost multiplier should now properly increase in BitNode-12 (by hydroflame)
- Bug Fix: 'document', 'hacknet', and 'window' keywords should no longer be counted multiple times in RAM calculations
- Bug Fix: Joining factions through Singularity functions should now prevent you from joining opposing factions
- Bug Fix: Four Sigma should no longer have two 'Speech Enhancement' Augmentations (by Kline-)

1.9.22 v0.40.1 - 8/5/2018 - Community Update

- Added getPurchasedServerCost() Netscript function (by kopelli)
- Added getFavorToDonate() Netscript function (by hydroflame)

- Added getFactionFavorGain() and getCompanyFavorGain() Singularity functions (by hydroflame)
- Accumulated 'bonus' time in Bladeburner is now displayed in the UI (by hydroflame)
- The Red Pill can now be purchased with negative money (since its supposed to be free) (by hydroflame)
- Cranial Signal Processor Augmentations now have the previous generation as a prerequisite. i.e. Cranial Signal Processor - Gen II requires Gen I (by Kline-)
- Terminal now supports semicolon usage (end of command). This allows chaining multiple Terminal commands (by hydroflame)
- Bladeburner Raid operations can no longer be performed if your estimate of Synthoid communities is zero (by hydroflame)
- The difficulty of BN-12 now scales faster (by hydroflame)
- Active Scripts UI now shows a RAM Usage bar for each server (by kopelli)
- Bug Fix: Corrected terminal timestamp format (by kopelli)
- Bug Fix: NetscriptJS scripts should now die properly if they don't have a 'main' function (by hydroflame)
- Bug Fix: write(), read(), and tryWrite() Netscript functions should now work properly for writing Arrays/objects to Netscript Ports
- Various minor UI/QOL fixes by hydroflame, kopelli, and Kline-

1.9.23 v0.40.0 - 7/28/2018

- **WARNING: This update makes some significant changes to Netscript and therefore you may need to make some changes to your scripts. See** this post **this post for details**
- Netscript 1.0 (NS1) now uses a fully-fledged ES5 JavaScript Interpreter. This means many new features are now available in NS1, and this also fixes several bugs. However this also means any ES6+ features are no longer supported in NS1
- When a server is hacked with a very large number of threads and left with no money, the server's security level now only increases by however many threads were needed to drain the server. For example, if you hack a server with 5000 threads but it only needed 2000 threads to deplete the server's money, then the server's security will only increase as if you had hacked it with 2000 threads (change by hydroflame)
- Added getCurrentAction() to Bladeburner API
- Added a variety of functions to Bladeburner API that deal with action levels (change by hydroflame)
- Added getPurchasedServerLimit() and getPurchasedServerMaxRam() functions to Netscript (change by hydroflame & kopelli)
- Added getOwnedSourceFiles() Singularity function (by hydroflame)
- Completely re-designed the Hacknet Node API
- getSkillLevel() in Bladeburner API now returns an error if no argument is passed in (as opposed to an object with all skill levels). This may break scripts
- Minimum Netscript execution time reduced from 15ms to 10ms (configurable in Options)
- Company reputation needed to get invited to Megacorporation factions decreased from 250k to 200k
- HP is now reset (restored) when Augmenting
- Source-File 6 now increases both the level and experience gain of all combat stats (it was only experience gain previously)

- Reverted a previous change for Source-File 12. It's benefits are now multiplicative rather than additive
- Starting Infiltration security level for almost every location decreased by ~10%
- Changed 'fl1ght.exe' message when its listed conditions are fulfilled (by hydroflame)
- The 'Save Game' button in the top-right overview panel now flashes red if autosave is disabled
- Bug Fix: Infiltration buttons can no longer be clicked through NetscriptJS
- Bug Fix: Bladeburner 'Overclock' skill can no longer be leveled above max level through the API (by hydroflame)
- Bug Fix: Healthcare division in Bladeburner should no longer cause game to crash

1.9.24 v0.39.1 - 7/4/2018

- Bladeburner Rank gain in BN-7 is now reduced by 40% instead of 50%
- Quadrupled the amount of money gained from Bladeburner contracts
- Added joinBladeburnerDivision() Netscript function to Bladeburner API
- Doubled the effects of Source-File 5. Now gives 8%, 12%, and 14% increase to all hacking multipliers at levels 1, 2, and 3, respectively (increased from 4%/6%, 7%)
- Increased the effect of Source-File 8. It now gives a 12%, 18% and 21% to your hacking growth multiplier at levels 1, 2, and 3, respectively (increased from 8%, 12%, 14%)
- The effect of Source-File 12 is now additive with itself, rather than multiplicative. This means that level N of Source-File 12 now increases all multipliers by N%
- The setting to suppress the confirmation box when purchasing Augmentations was moved into the main Options menu (by Github user hydroflame)
- Bug Fix: Crime Success rates were being calculated incorrectly (by Github user hydroflame)
- When an Infiltration is finished, you will now return back to the company's page, rather than the city
- Infiltration faction reputation selector now remembers your last choice
- Significantly increased the amount of money gained from Infiltration
- Bug Fix: Copying a NetscriptJS script to another server using scp now properly takes into account the script's changes.
- Bug Fix: Fixed an issue where game would not load in Edge due to incompatible features
- travelToCity() Singularity function no longer grants Intelligence exp"

1.9.25 v0.39.0 - 6/25/2018

- Added BitNode-7: Bladeburner 2079
- Infiltration base difficulty decreased by 10% for most locations
- Experience gains from Infiltration slightly increased
- Money gained from Infiltration increased by 20%
- Added 'var' declarations in Netscript 1.0 (only works with 'var', not 'let' or 'const')
- Script base RAM cost is now 1.6 GB (increased from 1.4 GB)
- While/for loops and if statements no longer cost RAM in scripts

- Made short-circuit evaluation logic more consistent in Netscript 1.0 (see https://github.com/danielyxie/bitburner/issues/308)
- Changelog button in the Options menu now links to the new Changelog URL (by Github user thePalindrome)
- Skill level calculation is now 'smoother' (by Github user hydroflame)
- Added a button to 'beautify' scripts in the text editor (by Github user hydroflame)
- Added favicon (by Github user kopelli)

1.9.26 v0.38.1 - 6/15/2018

- Bug Fix: Using 'Object.prototype' functions like toLocaleString() or toString() should no longer cause errors in NetscriptJS
- **Implemented by Github user hydroflame:**
  - Accessing the 'window' and 'document' objects in Netscript JS now requires a large amount of RAM (100 GB)
  - Added game option to suppress travel confirmation
  - Text on buttons can no longer be highlighted
  - Bug Fix: Fixed an issue that caused NaN values when exporting Real Estate in Corporations
  - Bug Fix: Competition and Demand displays in Corporation are now correct (were reversed before)
  - Added ps() Netscript function
  - Bug Fix: grow() should no longer return/log a negative value when it runs on a server that's already at max money
  - Bug Fix: serverExists() Netscript function should now properly return false for non-existent hostname/ips
  - Bug Fix: Sever's security level should now properly increase when its money is grown to max value

1.9.27 v0.38.0 - 6/12/2018

- New BitNode: BN-12 The Recursion - Implemented by Github user hydroflame
- **Bladeburner Changes:**
  - Bladeburner progress is no longer reset when installing Augmentations
  - The number of successess needed to increase a Contract/Operation's max level now scales with the current max level (gradually gets harder)
  - All Bladeburner Augmentations are now slightly more expensive and require more reputation
  - Black Operations now give higher rank rewards
  - Doubled the base amount of money gained from Contracts
  - Increased the amount of experience gained from Contracts/Actions
  - Added a new Augmentation: The Blade's Simulacrum
  - Bladeburner faction reputation gain is now properly affected by favor
- Hacking is now slightly less profitable in BitNode-3
- Updated Hacknet Nodes UI - Implemented by Github user kopelli

- Bug Fix: Fixed an exploit that allowed calling any Netscript function without incurring any RAM Cost in NetscriptJS

1.9.28 v0.37.2 - 6/2/2018

- After joining the Bladeburners division, there is now a button to go to the Bladeburner content in the 'City' page
- You now start with $250m in BitNode-8 (increased from $100m)
- Bug Fix: You can now no longer directly edit Hacknet Node values through NetscriptJS (hopefully)
- Bug Fix: Bladeburners is no longer accessible in BN-8
- Bug Fix: getBitNodeMultipliers() Netscript function now returns a copy rather than the original object

1.9.29 v0.37.1 - 5/22/2018

- You now earn money from successfully completing Bladeburner contracts. The amount you earn is based on the difficulty of the contract.
- Completing Field Analysis in Bladeburner now grants 0.1 rank
- The maximum RAM you can get on a purchased server is now 1,048,576 GB (2^20)
- Bug Fix: Fixed Netscript syntax highlighting issues with the new NetscriptJS
- Bug Fix: Netscript Functions now properly incur RAM costs in NetscriptJS
- Bug Fix: deleteServer() now fails if its called on the server you are currently connected to
- Removed in-game Netscript documentation, since it was outdated and difficult to maintain.
- Bug Fix: Updated the gymWorkout() Singularity function with the new exp/cost values for gyms

1.9.30 v0.37.0 - 5/20/2018

- NetscriptJS (Netscript 2.0) released (Documentation here: http://bitburner.readthedocs.io/en/latest/netscriptjs.html)
- Running the game with the '?noScripts' query will start the game without loading any of your scripts. This should be used if you accidentally write a script that crashes your game

1.9.31 v0.36.1 - 5/11/2018

- **Bladeburner Changes:**
  - Bug Fix: You can no longer get Bladeburner faction reputation through Infiltration
  - Initial difficulty of Tracking contracts reduced
  - Datamancer skill effect increased from 4% per level to 5%
  - Slightly decreased the base stamina cost of contracts/operations
  - Slightly increased the effects of the Tracer, Digital Observer, Short Circuit, Cloak, and Blade's Intuition skills
  - Overclock skill capped at level 95, rather than 99
  - Training gives significantly more exp/s

- Crime, Infiltration, and Hacking are now slightly more profitable in BN-6
- Gyms are now more expensive, but give slightly more exp
- Added getScriptName() and getHacknetMultipliers() Netscript functions (added by Github user hydroflame)
- getScriptRam() Netscript function now has default value for the second argument, which is hostname/ip (implemented by Github user hydroflame)
- There is now a soft-cap on stock price, which means it's no longer possible for the price of a stock to reach insanely-high values
- The ctrl+b hotkey in the text editor should now also be triggered by command+b on OSX (I don't have OSX so I can't confirm if this works)
- Many servers now have additional RAM
- Added an option to disable hotkeys/keyboard shortcuts
- Refactored 'Active Scripts' UI page to optimize its performance
- Added a new .fconf Terminal setting: ENABLE_TIMESTAMP
- 'Netscript Execution Time', which can be found in the Options, now has a minimum value of 15ms rather than 25ms
- Bug Fix: Fixed a typo in the Fulcrum Technologies company name (Technolgies -> Technologies)
- Bug Fix: hacknetnodes keyword should no longer incur RAM cost if its in a comment
- Bug Fix: disableLog() now works for the commitCrime() Netscript function (fixed by Github user hydroflame)

1.9.32 v0.36.0 - 5/2/2018

- Added BN-6: Bladeburners
- Rebalanced many combat Augmentations so that they are slightly less powerful
- Bug Fix: When faction invites are suppressed, an invitation will no longer load the Faction page

1.9.33 v0.35.2 - 3/26/2018

- **Corporation Changes:**
  - Fixed an issue with Warehouse upgrade cost. Should now be significantly cheaper than before.
  - Scientific Research now has a slightly more significant effect on Product quality
  - The Energy and Water Utilities industries are now slightly more profitable
  - The Robotics and Computer Hardware industries are now less profitable
  - The Software industry is slightly less profitable
  - When selling Materials and Products, the 'PROD' qualifier can now be used to set dynamic sell amounts based on your production
  - Exporting MAX should now work properly
  - You can no longer export past storage limits
  - Scientific Research production reduced

  - Effects of AdVert. Inc upgrade were reduced, but the effect that popularity and awareness have on sales was increased to compensate (popularity/awareness numbers were getting too big with Advert. Inc)
  - Bug Fix: Products from Computer Hardware division should now properly have ratings
- Improved Augmentation UI/UX. Now contains collapsible headers and sort buttons
- Improved Faction Augmentations display UI/UX. Now contains sort buttons. There is also an option to disable confirmation when purchasing Augmentations

1.9.34 v0.35.1 - 3/12/2018

- You can now easily download all of your scripts/text files as zip folders. Use the 'help download' Terminal command for details
- Scripts are now downloaded with the .script.js extension at the end of their filename
- **Corporation Management Changes:**
  - Implemented Smart Supply unlock
  - Changed the way a division's Production Multiplier is calculated. It is now the sum of the individual Production Multiplier for every city. Therefore, it is now beneficial to open offices in different cities
  - Several small UI/UX improvements
  - Numerous balance changes. The significant ones are listed below.
  - Product descriptions will now display their estimated market price
  - The sale price of Products can no longer be marked up as high as before
  - Scientific Research now affects the rating of Products
  - In general, the maximum amount of product you are able to sell is reduced
  - Sale bonus from advertising (popularity/awareness) now has diminishing returns rather than scaling linearly
- Experience gained during Infiltration now scales linearly based on the clearance level you reach. Compared to before, the experience gained will be much less at lower clearance levels, but much more at higher clearance levels
- The editor can now be used to edit both scripts and text files
- New Terminal config file that can be edited using the command 'nano .fconf'. Right now there is only one option, but there will be more in the future.
- You can now enable Bash-style Terminal hotkeys using the .fconf file referenced above
- Bug Fix: Fixed an issue with the UI elements of Gang Management persisting across different instances of BitNode-2

1.9.35 v0.35.0 - 3/3/2018

- Minor rebalancing of BitNodes due to the fact that Corporations provide a (relatively) new method of progressing
- **Corporation Management Changes:**
  - Once your Corporation gets big/powerful enough, you can now bribe Factions for reputation using company funds an/or stock shares

  - You can now only create one Division for every Industry type
  - Added several new UI/UX elements
  - Wilson Analytics multiplier was significantly reduced to 1% per level (additive).
  - Reduced the effect of Advert Inc upgrade. Advert Inc. upgrade price increases faster
  - Materials can now be marked up at higher prices
- Added Javascript's built-in Number object to Netscript
- Added getCharacterInformation(), getCompanyFavor(), and getFactionFavor() Netscript Singularity functions
- Rebalanced Singularity Function RAM Costs. They now cost x8 as much when outside of BN-4 (rather than x10). Also, many of the functions now use significantly less RAM
- Refactored Netscript Ports. You can now get a handle for a Netscript port using the getPortHandle() Netscript function. This allows you to access a port's underlying queue (which is just an array) and also makes several new functions available such as tryWrite(), full(), and empty().
- Number of Netscript Ports increased from 10 to 20
- Netscript assignments should now return proper values. i.e. i = 5 should return 5.
- Added throw statements to Netscript. It's not super useful since 'catch' isn't implemented, but it can be used to generate custom runtime error messages.
- Added import declaration to Netscript. With this, you are able to import functions (and only functions) from other files. Using export declarations is not necessary
- Most Netscript Runtime errors (the ones that cause your script to crash) should now include the line number where the error occured
- When working for a company, your current company reputation is now displayed
- Whenever you get a Faction Invite it will be immediately appended to your 'invited factions' list. Therefore the checkFactionInvitations() Singularity Function should now be properly useable since you no longer need to decline a Faction Invitation before it shows up in the result.
- Bug Fix: When purchasing servers, whitespace should now automatically be removed from the hostname
- Bug Fix: Can no longer have whitespace in the filename of text files created using write()
- Bug Fix: In Netscript, you can no longer assign a Hacknet Node handle (hacknetnodes[i]) to another value
- Bug Fix: If you are in the Factions tab when you accept an invitation from a Faction, the page will now properly 'refresh'
- Bug Fix: Scripts that run recursive functions should now be killed properly

1.9.36 v0.34.5 - 2/24/2018

- **Corporation Management Changes:**
  - Market Research unlocks are now cheaper
  - New 'VeChain' upgrade: displays useful statistics about Corporation
  - Corporation cycles are processed 25% faster
  - Corporation valuation was lowered by ~10% (this affects stock price and investments)
  - Rebalanced the effects of advertising. Should now be more effective for every Industry
  - Fixed several bugs/exploits involving selling and buying back stock shares

  - You will now receive a Corporation Handbook (.lit file) when starting out BitNode-3. It contains a brief guide to help you get started. This same handbook can be viewed from the Corporation management screen
  - Slightly decreased the amount by which a Product's sell price can be marked up
  - Employees can now be assigned to a 'Training' task, during which they will slowly increase several of their stats
- Hopefully fixed an exploit with Array.forEach(). If there are any issues with using forEach, let me know
- Arguments passed into a script are now passed by value. This means modifying the 'args' array in a script should no longer cause issues
- Scripts executed programatically (via run(), exec(), etc.) will now fail if null/undefined is passed in as an argument
- Added peek() Netscript function
- killall() Netscript function now returns true if any scripts were killed, and false otherwise.
- hack() Netscript function now returns the amount of money gained for successful hacks, and 0 for failed hacks
- scp Terminal command and Netscript function now work for txt files
- **Changes courtesy of Wraithan:**
  - Text files are now displayed using 'pre' rather than 'p' elements when using the 'cat' Terminal command. This means tabs are retained and lines don't automatically wrap
  - ls() Netscript function now returns text files as well
- Removed round() Netscript function, since you can just use Math.round() instead
- Added disableLog() and enableLog() Netscript functions
- Removed the 'log' argument from sleep(), since you can now use the new disableLog function
- 'Netscript Documentation' button on script editor now points to new readthedocs documentation rather than wiki
- When working for a faction, your current faction reputation is now displayed
- Bug Fix: Hacking Missions should no longer break when dragging an existing connection to another Node
- Bug Fix: Fixed RAM usage of getNextHacknetNodeCost() (is not 1.5GB instead of 4GB)

1.9.37 v0.34.4 - 2/14/2018

- Added several new features to Gang UI to make it easier to manage your Gang.
- Changed the Gang Member upgrade mechanic. Now, rather than only being able to have one weapon/armor/vehicle/etc., you can purchase all the upgrades for each Gang member and their multipliers will stack. To balance this out, the effects (AKA multipliers) of each Gang member upgrade were reduced.
- Added a new script editor option: Max Error Count. This affects how many approximate lines the script editor will process (JSHint) for common errors. Increasing this option can affect negatively affect performance
- Game theme colors (set using 'theme' Terminal command) are now saved when re-opening the game
- 'download' Terminal command now works on scripts
- Added stopAction() Singularity function and the spawn() Netscript function
- The 'Purchase Augmentations' UI screen will now tell you if you need a certain prerequisite for Augmentations.

- Augmentations with prerequisites can now be purchased as long as their prerequisites are puchased (before, you had to actually install the prerequisites before being able to purchase)

1.9.38 v0.34.3 - 1/31/2018

- **Minor balance changes to Corporations:**
  - Upgrades are generally cheaper and/or have more powerful effects.
  - You will receive more funding while your are a private company.
  - Product demand decreases at a slower rate.
  - Production multiplier for Industries (receives for owning real estate/hardware/robots/etc.) is slightly higher
- Accessing the hacknetnodes array in Netscript now costs 4.0GB of RAM (only counts against RAM usage once)
- Bug Fix: Corporation oustanding shares should now be numeric rather than a string
- Bug Fix: Corporation production now properly calculated for industries that dont produce materials.
- Bug Fix: Gangs should now properly reset when switching BitNodes
- Bug Fix: Corporation UI should now properly reset when you go public

1.9.39 v0.34.2 - 1/27/2018

- **Corporation Management Changes:**
  - Added advertising mechanics
  - Added Industry-specific purchases
  - Re-designed employee management UI
  - Rebalancing: Made many upgrades/purchases cheaper. Receive more money from investors in early stage. Company valuation is higher after going public
  - Multiple bug fixes
- Added rm() Netscript function
- Updated the way script RAM usage is calculated. Now, a function only increases RAM usage the first time it is called. i.e. even if you call hack() multiple times in a script, it only counts against RAM usage once. The same change applies for while/for loops and if conditionals.
- **The RAM cost of the following were increased:**
  - If statements: increased by 0.05GB
  - run() and exec(): increased by 0.2GB
  - scp(): increased by 0.1GB
  - purchaseServer(): increased by 0.25GB
- Note: You may need to re-save all of your scripts in order to re-calculate their RAM usages. Otherwise, it should automatically be re-calculated when you reset/prestige
- The cost to upgrade your home computer's RAM has been increased (both the base cost and the exponential upgrade multiplier)
- The cost of purchasing a server was increased by 10% (it is now $55k per RAM)

- Bug fix: (Hopefully) removed an exploit where you could avoid RAM usage for Netscript function calls by assigning functions to a variable (foo = hack(); foo('helios');)
- Bug fix: (Hopefully) removed an exploit where you could run arbitrary Javascript code using the constructor() method
- Thanks to Github user mateon1 and Reddit users havoc_mayhem and spaceglace for notifying me of the above exploits
- The fileExists() Netscript function now works on text files (.txt). Thanks to Github user devoidfury for this

1.9.40 v0.34.1 - 1/19/2018

- **Updates to Corporation Management:**
  - Added a number of upgrades to various aspects of your Corporation
  - Rebalanced the properties of Materials and the formula for determining the valuation of the Corporation
  - Fixed a number of bugs
- 'Stats' page now shows information about current BitNode
- You should now be able to create Corporations in other BitNodes if you have Source-File 3
- Added a new create-able program called b1t_flum3.exe. This program can be used to reset and switch BitNodes
- Added an option to adjust autosave interval
- Line feeds, newlines, and tabs will now work with the tprint() Netscript function
- Bug fix: 'check' Terminal command was broken
- Bug fix: 'theme' Terminal command was broken when manually specifying hex codes
- Bug fix: Incorrect promotion requirement for 'Business'-type jobs
- Bug fix: Settings input bars were incorrectly formatted when loading game

1.9.41 v0.34.0 - 12/6/2017

- Added clear() and exit() Netscript functions
- When starting out or prestiging, you will now receive a 'Hacking Starter Guide'. It provides tips/pointers for new players
- Doubled the amount of RAM on low-level servers (up to required hacking level 150)
- Slightly increased experience gain from Infiltration
- buyStock(), sellStock(), shortStock(), and sellShort() Netscript function now return the stock price at which the transaction occurred, rather than a boolean. If the function fails for some reason, 0 will be returned.
- **Hacking Mission Changes:**
  - You can now select multiple Nodes of the same type by double clicking. This allows you to set the action of all of selected nodes at once (e.g. set all Transfer Nodes to Fortify). Creating connections does not work with this multi-select functionality yet
  - Shield and Firewall Nodes can now fortify
  - The effects of Fortifying are now ~5% lower

  - Conquering a Spam Node now increases your time limit by 25 seconds instead of 15
  - Damage dealt by Attacking was slightly reduced
  - The effect of Scanning was slightly reduced
  - Enemy CPU Core Nodes start with slightly more attack. Misc Nodes start with slightly less defense
- **Corporation Management changes:**
  - Added several upgrades that unlock new features
  - Implemented Exporting mechanic
  - Fixed many bugs

1.9.42 v0.33.0 - 12/1/2017

- Added BitNode-3: Corporatocracy. In this BitNode you can start and manage your own corporation. This feature is incomplete. Much more will be added to it in the near future
- Minor bug fixes

1.9.43 v0.32.1 - 11/2/2017

- Updated Netscript's 'interpreter/engine' to use the Bluebird promise library instead of native promises. It should now be faster and more memory-efficient. If this has broken any Netscript features please report it through Github or the subreddit (reddit.com/r/bitburner)
- Rebalanced stock market (adjusted parameters such as the volatility/trends/starting price of certain stocks)
- Added prompt() Netscript function
- Added 'Buy Max' and 'Sell All' functions to Stock Market UI
- Added 'Portfolio' Mode to Stock Market UI so you can only view stocks you have a position/order in
- Added a button to kill a script from its log display box

1.9.44 v0.32.0 - 10/25/2017

- Added BitNode-8: Ghost of Wall Street
- Re-designed Stock Market UI
- Minor bug fixes

1.9.45 v0.31.0 - 10/15/2017

- Game now saves to IndexedDb (if your browser supports it). This means you should no longer have trouble saving the game when your save file gets too big (from running too many scripts). The game will still be saved to localStorage as well
- New file type: text files (.txt). You can read or write to text files using the read()/write() Netscript commands. You can view text files in Terminal using 'cat'. Eventually I will make it so you can edit them in the editor but that's not available yet. You can also download files to your real computer using the 'download' Terminal command

- Added a new Crime: Bond Forgery. This crime takes 5 minutes to attempt and gives $4,500,000 if successful. It is meant for mid game.
- Added commitCrime(), getCrimeChance(), isBusy(), and getStats() Singularity Functions.
- Removed getIntelligence() Netscript function
- Added sprintf and vsprintf to Netscript. See [https://github.com/alexei/sprintf.js this Github page for details]
- Increased the amount of money gained from Infiltration by 20%, and the amount of faction reputation by 12%
- Rebalanced BitNode-2 so that Crime and Infiltration are more profitable but hacking is less profitable. Infiltration also gives more faction rep
- Rebalanced BitNode-4 so that hacking is slightly less profitable
- Rebalanced BitNode-5 so that Infiltration is more profitable and gives more faction rep
- Rebalanced BitNode-11 so that Crime and Infiltration are more profitable. Infiltration also gives more faction rep.
- Fixed an annoying issue in Hacking Missions where sometimes you would click a Node but it wouldnt actually get selected
- Made the Hacking Mission gameplay a bit slower by lowering the effect of Scan and reducing Attack damage
- Slightly increased the base reputation gain rate for factions when doing Field Work and Security Work

1.9.46 v0.30.0 - 10/9/2017

- Added getAugmentations() and getAugmentationsFromFaction() Netscript Singularity Functions
- Increased the rate of Intelligence exp gain
- Added a new upgrade for home computers: CPU Cores. Each CPU core on the home computer grants an additional starting Core Node in Hacking Missions. I may add in other benefits later. Like RAM upgrades, upgrading the CPU Core on your home computer persists until you enter a new BitNode.
- Added lscpu Terminal command to check number of CPU Cores
- Changed the effect of Source-File 11 and made BitNode-11 a little bit harder
- Fixed a bug with Netscript functions (the ones you create yourself)
- **Hacking Missions officially released (they give reputation now). Notable changes in the last few updates:**
  - Misc Nodes slowly gain hp/defense over time
  - Conquering a Misc Node will increase the defense of all remaining Misc Nodes that are not being targeted by a certain percentage
  - Reputation reward for winning a Mission is now affected by faction favor and Player's faction rep multiplier
  - Whenever a Node is conquered, its stats are reduced

1.9.47 v0.29.3 - 10/3/2017

- Fixed bug for killing scripts and showing error messages when there are errors in a player-defined function
- Added function name autocompletion in Script Editor. Press Ctrl+space on a prefix to show autocompletion options.

- Minor rebalancing and bug fixes for Infiltration and Hacking Missions

1.9.48 v0.29.2 - 10/1/2017

- installAugmentations() Singularity Function now takes a callback script as an argument. This is a script that gets ran automatically after Augmentations are installed. The script is run with no arguments and only a single thread, and must be found on your home computer.
- Added the ability to create your own functions in Netscript. See [[Netscript Functions|this link]] for details
- Added :q, :x, and :wq Vim Ex Commands when using the Vim script editor keybindings. :w, :x, and :wq will all save the script and return to Terminal. :q will quit (return to Terminal) WITHOUT saving. If anyone thinks theres an issue with this please let me know, I don't use Vim
- Added a new Augmentation: ADR-V2 Pheromone Gene
- In Hacking Missions, enemy nodes will now automatically target Nodes and perform actions.
- Re-balanced Hacking Missions through minor tweaking of many numbers
- The faction reputation reward for Hacking Missions was slightly increased

1.9.49 v0.29.1 - 9/27/2017

- New gameplay feature that is currently in BETA: Hacking Missions. Hacking Missions is an active gameplay mechanic (its a minigame) that is meant to be used to earn faction reputation. However, since this is currently in beta, hacking missions will NOT grant reputation for the time being, since the feature likely has many bugs, balance problems, and other issues. If you have any feedback regarding the new feature, feel free to let me know
- CHANGED THE RETURN VALUE OF getScriptIncome() WHEN RAN WITH NO ARGUMENTS. It will now return an array of two values rather than a single value. This may break your scripts, so make sure to update them!
- Added continue statement for for/while loops
- Added getServerMinSecurityLevel(), getPurchasedServers(), and getTimeSinceLastAug() Netscript functions
- Netscript scp() function can now take an array as the first argument, and will try to copy every file specified in the array (it will just call scp() normally for every element in the array). If an array is passed in, then the scp() function returns true if at least one element from the array is successfully copied
- Added Javascript's Date module to Netscript. Since 'new' is not supported in Netscript yet, only the Date module's static methods will work (now(), UTC(), parse(), etc.).
- Failing a crime now gives half the experience it did before
- The forced repeated 'Find The-Cave' message after installing The Red Pill Augmentation now only happens if you've never destroyed a BitNode before, and will only popup every 15 minutes. If you have already destroyed a BitNode, the message will not pop up if you have messages suppressed (if you don't have messages suppressed it WILL still repeatedly popup)
- fileExists() function now works on literature files

1.9.50 v0.29.0 - 9/19/2017

- Added BitNode-5: Artificial Intelligence
- Added getIp(), getIntelligence(), getHackingMultipliers(), and getBitNodeMultipliers() Netscript functions (requires Source-File 5)

- Updated scan() Netscript function so that you can choose to have it print IPs rather than hostnames
- Refactored scp() Netscript function so that it takes an optional 'source server' argument
- For Infiltration, decreased the percentage by which the security level increases by about 10% for every location
- Using :w in the script editor's Vim keybinding mode should now save and quit to Terminal
- Some minor optimizations that should reduce the size of the save file
- scan-analyze Terminal command will no longer show your purchased servers, unless you pass a '-a' flag into the command
- After installing the Red Pill augmentation from Daedalus, the message telling you to find 'The-Cave' will now repeatedly pop up regardless of whether or not you have messages suppressed
- Various bugfixes

1.9.51 v0.28.6 - 9/15/2017

- Time required to create programs now scales better with hacking level, and should generally be much faster
- Added serverExists(hostname/ip) and getScriptExpGain(scriptname, ip, args...) Netscript functions
- Short circuiting && and || logical operators should now work
- Assigning to multidimensional arrays should now work
- Scripts will no longer wait for hack/grow/weaken functions to finish if they are killed. They will die immediately
- The script loop that checks whether any scripts need to be started/stopped now runs every 6 seconds rather than 10 (resulting in less delays when stopping/starting scripts)
- Fixed several bugs/exploits
- Added some description for BitNode-5 (not implemented yet, should be soon though)

1.9.52 v0.28.5 - 9/13/2017

- The fl1ght.exe program that is received from jump3r is now sent very early on in the game, rather than at hacking level 1000
- Hostname is now displayed in Terminal
- Syntax highlighting now works for all Netscript functions
- Export should now work on Edge/IE

1.9.53 v0.28.4 - 9/11/2017

- Added getScriptIncome() Netscript function
- Added Javascript's math module to Netscript. See [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math this link for details]
- Added several member variables for the Hacknet Node API that allow you to access info about their income
- All valid Netscript functions are now syntax highlighted as keywords in the editor. This means they will a different color than invalid netscript functions. The color will depend on your theme. Note that right now, this only applies for normal Netscript functions, not functions in the TIX API, Hacknet Node API, or Singularity Functions.

- Comments and operators no longer count towards RAM usage in scripts.
- Variety of bug fixes and updates to informational text in the game

1.9.54 v0.28.3 - 9/7/2017

- Added ls() Netscript function
- Increased company wages by about ~10% across the board
- The scp() Netsction function and Terminal command now works for .lit files
- Increased the amount of RAM on many lower level servers (up to level 200 hacking level required).

1.9.55 v0.28.2 - 9/4/2017

- Added several configuration options for script editor (key bindings, themes, etc.)
- Certain menu options will now be hidden until their relevant gameplay is unlocked. This includes the Factions, Augmentations, Create Program, Travel, and Job tabs. This will only affect newer players.
- Most unrecognize or un-implemented syntax errors in Netscript will now include the line number in the error message

1.9.56 v0.28.1 - 9/1/2017

- The script editor now uses the open-source Ace editor, which provides a much better experience when coding!
- Added tprint() Netscript function

1.9.57 v0.28.0 - 8/30/2017

- Added BitNode-4: The Singularity
- Added BitNode-11: The Big Crash
- Migrated the codebase to use webpack (doesn't affect any in game content, except maybe some slight performance improvements and there may be bugs that result from dependency errors

1.9.58 v0.27.3 - 8/19/2017

- You can now purchase upgrades for Gang Members (BitNode 2 only)
- Decreased Gang respect gains and slightly increased wanted gains (BitNode 2 only)
- Other gangs will increase in power faster (BitNode 2 only)
- Added getHackTime(), getGrowTime(), and getWeakenTime() Netscript functions

1.9.59 v0.27.2 - 8/18/2017

- Added getServerGrowth() Netscript function
- Added getNextHacknetNodeCost() Netscript function
- Added new 'literature' files (.lit extension) that are used to build lore for the game. These .lit files can be found in certain servers throughout the game. They can be viewed with the 'cat' Terminal command and copied over to other servers using the 'scp' command. These .lit files won't be found until you reset by installing Augmentations
- Fixed some bugs with Gang Territory(BitNode 2 only)

1.9.60 v0.27.1 - 8/15/2017

- Changed the way Gang power was calculated to make it scale better late game (BitNode 2 only)
- Lowered the respect gain rate in Gangs (Bitnode 2 only)
- Added '| grep pattern' option for ls Terminal command. This allows you to only list files that contain a certain pattern
- Added break statement in Netscript
- Display for some numerical values is now done in shorthand (e.g 1.000m instead of 1,000,000)

1.9.61 v0.27.0 - 8/13/2017

- Added secondary 'prestige' system - featuring Source Files and BitNodes
- MILD SPOILERS HERE: Installing 'The Red Pill' Augmentation from Daedalus will unlock a special server called w0r1d_d43m0n. Finding and manually hacking this server through Terminal will destroy the Player's current BitNode, and allow the player to enter a new one. When destroying a BitNode, the player loses everything except the scripts on his/her home computer. The player will then gain a powerful second-tier persistent upgrade called a Source File. The player can then enter a new BitNode to start the game over. Each BitNode has different characteristics, and many will have new content/mechanics as well. Right now there are only 2 BitNodes. Each BitNode grants its own unique Source File. Restarting and destroying a BitNode you already have a Source File for will upgrade your Source File up to a maximum level of 3.
- Reputation gain with factions and companies is no longer a linear conversion, but an exponential one. It will be much easier to gain faction favor at first, but much harder later on.
- Significantly increased Infiltration exp gains
- Fixed a bug with company job requirement tooltips
- Added scriptRunning(), scriptKill(), and getScriptRam() Netscript functions. See documentation for details
- Fixed a bug with deleteServer() Netscript function

1.9.62 v0.26.4 - 8/1/2017

- All of the 'low-level servers' in early game that have a required hacking level now have 8GB of RAM instead of 4GB
- Increased the amount of experience given at university
- Slightly increased the production of Hacknet Nodes and made them cheaper to upgrade

- Infiltration now gives slightly more EXP and faction reputation
- Added two new crimes. These crimes are viable to attempt early on in the game and are relatively passive (each take 60+ seconds to complete)
- Crimes give more exp and more money
- Max money available on a server decreased from 50x the server's starting money to 25x
- Significantly increased wages for all jobs

1.9.63 v0.26.3

- Added support for large numbers using Decimal.js. Right now it only applies for the player's money
- Purchasing servers with the Netscript function purchaseServer() is no longer 2x as expensive as doing manually it now costs the same
- Early game servers have more starting money

1.9.64 v0.26.2

- Major rebalancing and randomization of the amount of money that servers start with
- Significantly lowered hacking exp gain from hacking servers. The exp gain for higher-level servers was lowered more than that of low level servers. (~16% for lower level servers, up to ~25% for higher-level servers)
- Added deleteServer() Netscript function
- You can now purchase a maximum of 25 servers each run (Deleting a server will allow you to purchase a new one)
- Added autocompletion for './' Terminal command
- Darkweb prices now displayed properly using toLocaleString()
- Added NOT operator (!) and negation operator(-) in Netscript, so negative numbers should be functional now
- Rejected faction invitations will now show up as 'Outstanding Faction Invites' in the Factions page. These can be accepted at any point in the future
- Added a few more configurable game settings for suppressing messages and faction invitations
- Added tooltips for company job requirements

1.9.65 v0.26.1

- Added autocompletion for aliases
- Added getServerRam() Netscript function()
- Added getLevelUpgradeCost(n), getRamUpgradeCost(), getCoreUpgradeCost() functions for Netscript Hacknet Node API
- Added some configurable settings (See Game Options menu)

1.9.66 v0.26.0

- Game now has a real ending, although it's not very interesting/satisfying right now. It sets up the framework for the secondary prestige system in the future
- Forgot to mention that since last update, comments now work in Netscript. Use // for single line comments or /* and */ for multiline comments just like in Javascript
- Added ports to Netscript. These ports are essentially serialized queues. You can use the write() Netscript function to write a value to a queue, and then you can use the read() Netscript function to read the value from the queue. Once you read a value from the queue it will be removed. There are only 10 queues (1-10), and each has a maximum capacity of 50 entries. If you try to write to a queue that is full, the the first value is removed. See wiki/Netscript documentation for more details
- You can now use the 'help' Terminal command for specific commands
- You can now use './' to run a script/program (./NUKE.exe). However, tab completion currently doesn't work for it (I'm working on it)
- Decreased the base growth rate of servers by ~25%
- Both the effect of weaken() and its time to execute were halved. In other words, calling weaken() on a server only lowers its security by 0.05 (was 0.1 before) but the time to execute the function is half of what it was before. Therefore, the effective rate of weaken() should be about the same
- Increased all Infiltration rewards by ~10%, and increased infiltration rep gains by an additional 20% (~32% total for rep gains)
- The rate at which the security level of a facility increases during Infiltration was decreased significantly (~33%)
- Getting treated at the Hospital is now 33% more expensive
- Slightly increased the amount of time it takes to hack a server
- Slightly decreased the amount of money gained when hacking a server (~6%)
- Slightly decreased the base cost for RAM on home computer, but increased the cost multiplier. This means that upgrading RAM on the home computer should be slightly cheaper at the start, but slightly more expensive later on
- Increased the required hacking level for many late game servers
- The sleep() Netscript function now takes an optional 'log' argument that specifies whether or not the 'Sleeping for N milliseconds' will be logged for the script
- Added clearLog() Netscript function
- Deleted a few stocks. Didn't see a reason for having so many, and it just affects performance. Won't take effect until you reset by installing Augmentations
- There was a typo with Zeus Medical's server hostname. It is now 'zeus-med' rather than 'zeud-med'
- Added keyboard shortcuts to quickly navigate between different menus. See wiki link (http://bitburner.wikia.com/wiki/Shortcuts)
- Changed the Navigation Menu UI

1.9.67 v0.25.0

- Refactored Netscript to use the open-source Acorns Parser. This re-implementation was done by [https://github.com/MrNuggelz Github user MrNuggelz]. This has resulted in several changes in the Netscript language. Some scripts might break because of these changes. Changes listed below:

- Arrays are now fully functional Javascript arrays. You no longer need to use the 'Array' keyword to declare them.
- The length(), clear/clear(), insert(), and remove() functions no longer work for arrays.
- All Javascript array methods are available (splice(), push(), pop(), join(), shift(), indexOf(), etc. See documentation)
- Variables assigned to arrays are now passed by value rather than reference
- Incrementing/Decrementing are now available (i++, ++i)
- You no longer need semicolons at the end of block statements
- Elif is no longer valid. Use 'else if' instead
- Netscript's Hacknet Node API functions no longer log anything
- Stock prices now update every ~6 seconds when the game is active (was 10 seconds before)
- Added a new mechanic that affects how stock prices change
- Script editor now has dynamic indicators for RAM Usage and Line number
- Augmentation Rebalancing - Many late game augmentations are now slightly more expensive. Several early game augmentations had their effects slightly decreased
- Increased the amount of rewards (both money and rep) you get from infiltration
- Purchasing servers is now slightly more expensive
- Calling the Netscript function getServerMoneyAvailable('home') now return's the player's money
- Added round(n) Netscript function - Rounds a number
- Added purchaseServer(hostname, ram) Netscript function
- Added the TIX API. This must be purchased in the WSE. It persists through resets. Access to the TIX API allows you to write scripts that perform automated algorithmic trading. See Netscript documentation
- Minor rebalancing in a lot of different areas
- Changed the format of IP Addresses so that they are smaller (will consist mostly of single digit numbers now). This will reduce the size of the game's save file.

1.9.68 v0.24.1

- Adjusted cost of upgrading home computer RAM. Should be a little cheaper for the first few upgrades (up to ~64GB), and then will start being more expensive than before. High RAM upgrades should now be significantly more expensive than before.
- Slightly lowered the starting money available on most mid-game and end-game servers (servers with required hacking level greater than 200) by about 10-15%
- Rebalanced company/company position reputation gains and requirements
- Studying at a university now gives slightly more EXP and early jobs give slightly less EXP
- Studying at a university is now considerably more expensive
- Rebalanced stock market
- Significantly increased cost multiplier for purchasing additional Hacknet Nodes
- The rate at which facility security level increases during infiltration for each clearance level was lowered slightly for all companies

- Updated Faction descriptions
- Changed the way alias works. Normal aliases now only work at the start of a Terminal command (they will only replace the first word in the Terminal command). You can also create global aliases that work on any part of the command, like before. Declare global aliases by entering the optional -g flag: alias -g name="value" -[https://github.com/MrNuggelz Courtesy of Github user MrNuggelz]
- 'top' Terminal command implemented courtesy of [https://github.com/LTCNugget Github user LTCNugget]. Currently, the formatting gets screwed up if your script names are really long.

1.9.69 v0.24.0

- Players now have HP, which is displayed in the top right. To regain HP, visit the hospital. Currently the only way to lose HP is through infiltration
- Infiltration - Attempt to infiltrate a company and steal their classified secrets. See 'Companies' documentation for more details
- Stock Market - Added the World Stock Exchange (WSE), a brokerage that lets you buy/sell stocks. To begin trading you must first purchase an account. A WSE account will persist even after resetting by installing Augmentations. How the stock market works should hopefully be self explanatory. There is no documentation about it currently, I will add some later. NOTE: Stock prices only change when the game is open. The Stock Market is reset when installing Augmentations, which means you will lose all your stocks
- Decreased money gained from hacking by ~12%
- Increased reputation required for all Augmentations by ~40%
- Cost increase when purchasing multiple augmentations increased from 75% to 90%
- Added basic variable runtime to Netscript operations. Basic commands run in 100ms. Any function incurs another 100ms in runtime (200ms total). Any function that starts with getServer incurs another 100ms runtime (300ms total). exec() and scp() require 400ms total.
- Slightly reduced the amount of experience gained from hacking

1.9.70 v0.23.1

- scan() Netscript function now takes a single argument representing the server from which to scan.

1.9.71 v0.23.0

- You can now purchase multiple Augmentations in a run. When you purchase an Augmentation you will lose money equal to the price and then the cost of purchasing another Augmentation during this run will be increased by 75%. You do not gain the benefits of your purchased Augmentations until you install them. This installation can be done through the 'Augmentation' tab. When you install your Augmentations, your game will reset like before.
- Reputation needed to gain a favor from faction decreased from 7500 to 6500
- Reputation needed to gain a favor from company increased from 5000 to 6000
- Reputation cost of all Augmentations increased by 16%
- Higher positions at companies now grant slightly more reputation for working
- Added getServerMaxMoney() Netscript function
- Added scan() Netscript function

- Added getServerNumPortsRequired() Netscript function
- There is now no additional RAM cost incurred when multithreading a script

1.9.72 v0.22.1

- You no longer lose progress on creating programs when cancelling your work. Your progress will be saved and you will pick up where you left off when you start working on it again
- Added two new programs: AutoLink.exe and ServerProfiler.exe
- Fixed bug with Faction Field work reputation gain

1.9.73 v0.22.0 - Major rebalancing, optimization, and favor system

- Significantly nerfed most augmentations
- Almost every server with a required hacking level of 200 or more now has slightly randomized server parameters. This means that after every Augmentation purchase, the required hacking level, base security level, and growth factor of these servers will all be slightly different
- The hacking speed multiplier now increases rather than decreases. The hacking time is now divided by your hacking speed multiplier rather than multiplied. In other words, a higher hacking speed multiplier is better
- Servers now have a minimum server security, which is approximately one third of their starting ('base') server security
- If you do not steal any money from a server, then you gain hacking experience equal to the amount you would have gained had you failed the hack
- The effects of grow() were increased by 50%
- grow() and weaken() now give hacking experience based on the server's base security level, rather than a flat exp amount
- Slightly reduced amount of exp gained from hack(), weaken(), and grow()
- Rebalanced formulas that determine crime success
- Reduced RAM cost for multithreading a script. The RAM multiplier for each thread was reduced from 1.02 to 1.005
- Optimized Script objects so they take less space in the save file
- Added getServerBaseSecurityLevel() Netscript function
- New favor system for companies and factions. Earning reputation at a company/faction will give you favor for that entity when you reset after installing an Augmentation. This favor persists through the rest of the game. The more favor you have, the faster you will earn reputation with that faction/company
- You can no longer donate to a faction for reputation until you have 150 favor with that faction
- Added unalias Terminal command
- Changed requirements for endgame Factions

1.9.74 v0.21.1

- IF YOUR GAME BREAKS, DO THE FOLLOWING: Options -> Soft Reset -> Save Game -> Reload Page. Sorry about that!

- Autocompletion for aliases - courtesy of [https://github.com/LTCNugget Github user LTCNugget]

1.9.75 v0.21.0

- Added dynamic arrays. See Netscript documentation
- Added ability to pass arguments into scripts. See documentation
- The implementation/function signature of functions that deal with scripts have changed. Therefore, some old scripts might not work anymore. Some of these functions include run(), exec(), isRunning(), kill(), and some others I may have forgot about. Please check the updated Netscript documentation if you run into issues.-Note that scripts are now uniquely identified by the script name and their arguments. For example, you can run a script using:

```
run foodnstuff.script 1
```

and you can also run the same script with a different argument:

```
run foodnstuff.script 2
```

These will be considered two different scripts. To kill the first script you must run:

```
kill foodnstuff.script 1
```

and to kill the second you must run:

```
kill foodnstuff.script 2
```

Similar concepts apply for Terminal Commands such as tail, and Netscript commands such as run(), exec(), kill(), isRunning(), etc.

- Added basic theme functionality using the 'theme' Terminal command - All credit goes to /u/0x726564646974 who implemented the awesome feature
- Optimized Script objects, which were causing save errors when the player had too many scripts
- Formula for determining exp gained from hacking was changed
- Fixed bug where you could purchase Darkweb items without TOR router
- Slightly increased cost multiplier for Home Computer RAM
- Fixed bug where you could hack too much money from a server (and bring its money available below zero)
- Changed tail command so that it brings up a display box with dynamic log contents. To get old functionality where the logs are printed to the Terminal, use the new 'check' command
- As a result of the change above, you can no longer call tail/check on scripts that are not running
- Added autocompletion for buying Programs in Darkweb

1.9.76 v0.20.2

- Fixed several small bugs
- Added basic array functionality to Netscript
- Added ability to run scripts with multiple threads. Running a script with n threads will multiply the effects of all hack(), grow(), and weaken() commands by n. However, running a script with multiple threads has drawbacks in terms of RAM usage. A script's ram usage when it is 'multithreaded' is calculated as: base cost * numThreads

* (1.02 ^ numThreads). A script can be run multithreaded using the 'run [script] -t n' Terminal command or by passing in an argument to the run() and exec() Netscript commands. See documentation.

- RAM is slightly (~10%) more expensive (affects purchasing server and upgrading RAM on home computer)
- NeuroFlux Governor augmentation cost multiplier decreased
- Netscript default operation runtime lowered to 200ms (was 500ms previously)

1.9.77 v0.20.1

- Fixed bug where sometimes scripts would crash without showing the error
- Added Deepscan programs to Dark Web
- Declining a faction invite will stop you from receiving invitations from that faction for the rest of the run
- (BETA) Added functionality to export/import saves. WARNING This is only lightly tested. You cannot choose where to save your file it just goes to the default save location. Also I have no idea what will happen if you try to import a file that is not a valid save. I will address these in later updates

1.9.78 v0.20.0

- Refactored Netscript Interpreter code. Operations in Netscript should now run significantly faster (Every operation such as a variable assignment, a function call, a binary operator, getting a variable's value, etc. used to take up to several seconds, now each one should only take ~500 milliseconds).
- Percentage money stolen when hacking lowered to compensate for faster script speeds
- Hacking experience granted by grow() halved
- Weaken() is now ~11% faster, but only grants 3 base hacking exp upon completion instead of 5
- Rebalancing of script RAM costs. Base RAM Cost for a script increased from 1GB to 1.5GB. Loops, hack(), grow() and weaken() all cost slightly less RAM than before
- Added getServerRequiredHackingLevel(server) Netscript command.
- Added fileExists(file, [server]) Netscript command, which is used to check if a script/program exists on a specified server
- Added isRunning(script, [server]) Netscript command, which is used to check if a script is running on the specified server
- Added killall Terminal command. Kills all running scripts on the current machine
- Added kill() and killall() Netscript commands. Used to kill scripts on specified machines. See Netscript documentation
- Re-designed 'Active Scripts' tab
- Hacknet Node base production rate lowered from 1.6 to 1.55 ($/second)
- Increased monetary cost of RAM (Upgrading home computer and purchasing servers will now be more expensive)
- NEW GROWTH MECHANICS - The rate of growth on a server now depends on a server's security level. A higher security level will result in lower growth on a server when using the grow() command. Furthermore, calling grow() on a server raises that server's security level by 0.004. For reference, if a server has a security level of 10 it will have approximately the same growth rate as before.
- Server growth no longer happens naturally

- Servers now have a maximum limit to their money. This limit is 50 times it's starting money
- Hacking now grants 10% less hacking experience
- You can now edit scripts that are running
- Augmentations cost ~11% more money and 25% more faction reputation

1.9.79 v0.19.7

- Added changelog to Options menu
- Bug fix with autocompletion (wasn't working properly for capitalized filenames/programs

1.9.80 v0.19.6

- Script editor now saves its state even when you change tabs
- scp() command in Terminal/script will now overwrite files at the destination
- Terminal commands are no longer case-sensitive (only the commands themselves such as 'run' or 'nano'. Filenames are still case sensitive
- Tab automcompletion will now work on commands

1.9.81 v0.19.0

- Hacknet Nodes have slightly higher base production, and slightly increased RAM multiplier. But they are also a bit more expensive at higher levels
- Calling grow() and weaken() in a script will now work offline, at slower rates than while online (The script now keeps track of the rate at which grow() and weaken() are called when the game is open. These calculated rates are used to determine how many times the calls would be made while the game is offline)
- Augmentations now cost 20% more reputation and 50% more money
- Changed the mechanic for getting invited to the hacking factions (CyberSec, NiteSec, The Black Hand, BitRunners) Now when you get to the required level to join these factions you will get a message giving you instructions on what to do in order to get invited.
- Added a bit of backstory/plot into the game. It's not fully fleshed out yet but it will be used in the future
- Made the effects of many Augmentations slightly more powerful
- Slightly increased company job wages across the board (~5-10% for each position)
- Gyms and classes are now significantly more expensive
- Doubled the amount by which a server's security increases when it is hacked. Now, it will increase by 0.002. Calling weaken() on a server will lower the security by 0.1.

1.9.82 v0.18.0

- Major rebalancing (sorry didn't record specifics. But in general hacking gives more money and hacknet nodes give less)
- Server growth rate (both natural and manual using grow()) doubled
- Added option to Soft Reset

- Cancelling a full time job early now only results in halved gains for reputation. Exp and money earnings are gained in full
- Added exec() Netscript command, used to run scripts on other servers.
- NEW HACKING MECHANICS: Whenever a server is hacked, its 'security level' is increased by a very small amount. The security level is denoted by a number between 1-100. A higher security level makes it harder to hack a server and also decreases the amount of money you steal from it. Two Netscript functions, weaken() and getServerSecurityLevel() level, were added. The weaken(server) function lowers a server's security level. See the Netscript documentation for more details
- When donating to factions, the base rate is now $1,000,000 for 1 reputation point. Before, it was $1,000 for 1 reputation point.
- Monetary costs for all Augmentations increased. They are now about ~3.3 - 3.75 times more expensive than before

1.9.83 v0.17.1

- Fixed issue with purchasing Augmentations that are 'upgrades' and require previous Augmentations to be installed
- Increased the percentage of money stolen from servers when hacking

1.9.84 v0.17.0

- Greatly increased amount of money gained for crimes (by about 400% for most crimes)
- Criminal factions require slightly less negative karma to get invited to
- Increased the percentage of money stolen from servers when hacking
- Increased the starting amount of money available on beginning servers (servers with <50 required hacking))
- Increased the growth rate of servers (both naturally and manually when using the grow() command in a script)
- Added getHostname() command in Netscript that returns the hostname of the server a script is running on
- jQuery preventDefault() called when pressing ctrl+b in script editor
- The Neuroflux Governor augmentation (the one that can be repeatedly leveled up) now increases ALL multipliers by 1%. To balance it out, it's price multiplier when it levels up was increased
- Hacknet Node base production decreased from $1.75/s to $1.65/s
- Fixed issue with nested for loops in Netscript (stupid Javascript references)
- Added 'scp' command to Terminal and Netscript
- Slightly nerfed Hacknet Node Kernel DNI and Hacknet Node Core DNI Augmentations
- Increased TOR Router cost to $200k

1.9.85 v0.16.0

- New Script Editor interface
- Rebalanced hacknet node - Increased base production but halved the multiplier from additional cores. This should boost its early-game production but nerf its late-game production
- Player now starts with 8GB of RAM on home computer

- 'scan-analyze' terminal command displays RAM on servers
- Slightly buffed the amount of money the player steals when hacking servers (by about ~8%)
- Time to execute grow() now depends on hacking skill and server security, rather than taking a flat 2 minutes.
- Clicking outside of a pop-up dialog box will now close it
- BruteSSH.exe takes 33% less time to create
- 'iron-gym' and 'max-hardware' servers now have 2GB of RAM
- Buffed job salaries across the board
- Updated Tutorial
- Created a Hacknet Node API for Netscript that allows you to access and upgrade your Hacknet Nodes. See the Netscript documentation for more details. WARNING The old upgradeHacknetNode() and getNumHacknetNodes() functions waere removed so any script that has these will no longer work

1.9.86 v0.15.0

- Slightly reduced production multiplier for Hacknet Node RAM
- Faction pages now scroll
- Slightly increased amount of money gained from hacking
- Added 'alias' command
- Added 'scan-analyze' terminal command - used to get basic hacking info about all immediate network connections
- Fixed bugs with upgradeHacknetNode() and purchaseHacknetNode() commands
- Added getNumHacknetNodes() and hasRootAccess(hostname/ip) commands to Netscript
- Increased Cost of university classes/gym
- You can now see what an Augmentation does and its price even while its locked

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Index

H

I

J

K