

Bandit – Stage by Stage explanation

Level 0

All you have to do here is ssh onto the bandit server. This can be done with the command `$ ssh -p 2220 bandit0@bandit.labs.overthewire.org` and then entering the password `bandit0`

Level 1

The password for the next level is stored in a file on the desktop named 'readme'.

This can be revealed with the `ls` command, and read with `cat readme`.

The password for bandit1 is `boJ9jbbUNNfktD7800psq0ltutMc3MY1`

Level 2

The password here is stored in a file called '-'

This can be revealed with the `ls` command, but running `cat -` confuses the system, as `-` normally denotes a flag. We can get around this by using the complete file path, `./-`. This gives us the command `cat ./-`

The password for bandit2 is `CV1DtqXWVFXTvM2F0k09SHz0YwRINYA9`

Level 3

The password for the next level is stored in a file called spaces in this filename located in the home directory

This cannot be accessed by simply running `cat spaces in this file name`, as the system will register this as four separate files. Instead, we can either use the escape character, `"`, in front of the spaces, or specify the file name in quotation marks. This gives us either `cat spaces\ in\ this\ filename` or `cat "spaces in this file name"`

The password for bandit3 is `UmHadQclWmgdLOKQ3YNgjWxGoRmb5luK`

Level 4

The password for the next level is stored in a hidden file in the inhere directory.

We can change into this directory using `cd inhere`, and then reveal the hidden file, which is called `.hidden`. We can then read the contents of this file with the command `cat .hidden`.

The password for bandit4 is `pIwrPrtPN36QITSp3EQaw936yaFoFgAB`

Level 5

The password for the next level is stored in the only human-readable file in the `inhere` directory. Tip: if your terminal is messed up, try the “reset” command.

Again, we can change to the `inhere` directory using `cd inhere`. Running `ls` then reveals that there are ten files in this directory. As we know that the password is contained in a human readable file, we need to somehow filter the files by filetype. This can be done with the command `find . -type f | xargs file`. This works by finding all the files in the current directory, and passing these as an argument into the `file` command, which then lists them by type. Doing so reveals to us that 9 of the 10 files are of type ‘data’, but one is of type ‘ASCII text’. Therefore, the password must be in this file.

This works given the relatively small number of files in this directory, but what if it were larger? We could extend this command by piping the output into `grep text`, giving us a complete command of

```
find . -type f | xargs file | grep text
```

This would return only the file marked as ASCII text, omitting all others.

The password for bandit5 is `koReBOKuIDDepwhWk7jZC0RTdopnAYKh`

Level 6

In this level, we are given three properties of the file containing the password. It is human readable, 1033 bytes and not executable.

Therefore, we need to somehow string these conditions together into a search command. We can again start with `find . -type f`, which is going to recursively find all files of in this directory and subdirectories. However, this gives an unusably long output. At this point, one might think to apply `| xargs file | grep text`, as in the previous level.

This was tried, but still presented the issue of giving an unworkably large number of files. For that reason, I changed the command used, and introduced the `-exec` flag to my initial `find` statement. This allows us to also run `ls` with the appropriate flags (to list more information on the file, including file size and whether it is executable). The flags I used were `-la`; `l` is the alias of long listing, giving more information, and `a` is the alias for all, meaning even hidden files are included. From here, I just needed to add a filter so that only files 1033 bytes long were

shown - this is done using the `grep 1033` command. By piping these together, we are left with

```
find . type -f -exec ls -la {} \; | grep 1033
```

The password for bandit6 is `DXjZPULLxYr17uwoI01bNLQbtFemEgo7`

Level 7

The password for the next level is stored somewhere on the server and has all of the following properties:

- Owned by user Bandit7
- Owned by group Bandit6
- 33 bytes in size

Immediately, we know that we wish to search the entire file system. This can be done with `find /`. As before, we know that we want to look for all files, so we can introduce the flag `-type f`. We can then make use of the `-user`, `-group` and `-size` flags for the `find` command, giving us:

```
$ find / -user bandit7 -group bandit6 -type f -size 33c
```

This works, but we get a slew of permission denied errors. This makes it difficult to identify where the correct file is. We want to somehow suppress these - this can be done by introducing `2>/dev/null` to the command, giving us a final command of

```
$ find / -user bandit7 -group bandit6 -type f -size 33c 2>/dev/null
```

The bandit7 password is `HKBPTKQnIay4Fw76bEy8PVxKEDQRKTzs`

Level 8

The password for the next level is stored in the file `data.txt` next to the word `millionth`

As we know that the password is next to the word `millionth`, we want to return all lines in the file that contain the word 'millionth'. This can be done by piping the contents of the file (using the `cat` command) into `grep`, as we have seen before. This gives a command of

```
cat data.txt | grep millionth
```

There is only one line containing the word `millionth`, so we immediately find our password. It is `cvX2JJJa4CFALtqS87jk27qwqGhBM9plV`

Level 9

Our guidance here is that the password for the next level is stored in the file `data.txt` and is the only line of text that occurs only once.

Immediately, my first thought was to use the `uniq` command, with the `-u` flag - this should filter a document and only return lines that occur once. As such, I ran

```
cat data.txt | uniq -u
```

However, this caused the entire file to be printed. On reading the `uniq` man page, I discovered that `uniq` only compares adjacent lines, and that I therefore needed to sort the file first. A working solution then is

```
cat data.txt | sort | uniq -q
```

The password to bandit 9 is `UsvVyFSfZZWbi6wgC7dAFyFuR6jQQUhr`

Level 10

The password for the next level is stored in the file `data.txt` in one of the few human-readable strings, preceded by several `'='` characters.

As we are told that the password is 'one of the few human-readable strings' we can use the command `strings`, which will return all human-readable strings in the document. We also know that the password is preceded by a string of equals signs, we can pipe the output of this command into a `grep` search again. This reveals the password. The final command was

```
strings data.txt | grep ==
```

This reveals that the password to bandit 10 is `truKLdjsbJ5g7yyJ2X2R0o3a5HQQJFuLk`

Level 11

The password for the next level is stored in the file `data.txt`, which contains base64 encoded data

In order to be able to read the contents of the file, we therefore need to decode it. This can be done with the command `base64 -d data.txt`. Running this reveals the password.

The password to bandit 11 is `IFukwKGSFW8M0q3IRFqrxE1hxTNEbUPR`

Level 12

The password for the next level is stored in the file `data.txt`, where all lowercase (a-z) and uppercase (A-Z) letters have been rotated by 13 positions

To decode this message, we need to shift every letter back by 13 places. This can be done using the `tr` (transliterate) command, and an appropriate filter. Again, we can use `cat data.txt |` to pipe the file contents in. Our final command is

```
$ cat data.txt | tr [n-za-mN-ZA-M] [a-zA-Z]
```

The password for bandit 12 is `5Te8Y4drgCRfCx8ugdwuEX8KFC6k2EUu`

Level 13

The password for the next level is stored in the file `data.txt`, which is a hexdump of a file that has been repeatedly compressed. For this level it may be useful to create a directory under `/tmp` in which you can work using `mkdir`. For example: `mkdir /tmp/myname123`. Then copy the datafile using `cp`, and rename it using `mv` (read the manpages!)

I started off here by following their suggestion, and creating / navigating to `/tmp/lloyd`. As we know that `data.txt` is a hexdump, the first thing to do is apply a reverse hexdump to this file. This is done using the following command

```
xxd -r > data
```

This pipes the output of the reverse hexdump command into a new file, `'data'`. We know that this file has been compressed, but we don't know how. We can find this out by running `file data`. The output of this tells us that `'data'` contains gzip compressed data. Therefore, we want to decompress it using `gzip`. Simply running `gzip -d data` throws an error, as it complains that the suffix is unknown, so we can instead use

```
zcat -d data > data2.bin
```

This decompresses the file and stores the output in `data2.bin`, which was the original file name. Again, we can run the `file` command to find out how this has been compressed. This informs us that the file was compressed with `bzip2`. We can decompress it with

```
bzip2 -d data2.bin
```

Running this decompresses the output into a new file called `'data2.bin.out'` but this is not a particularly nice name, so I renamed it to `'data3.bin'` using the `mv` command. Again, I checked how it can be compressed using `file`.

These steps were repeated several times, before I eventually reached the original ASCII text file containing the password.

Password to bandit 13 is `8ZjyCRiBWFYkneahHwxCv3wb2a10RpYL`

Level 14

The password for the next level is stored in `/etc/bandit_pass/bandit14` and can only be read by user `bandit14`. For this level, you don't get the next password, but you get a private SSH key that can be used to log into the next level. Note: `localhost` is a hostname that refers to the machine you are working on

Here, we are given bandit 14's private ssh key, which will allow us to log in as them. This is contained in a file called `sshkey.private`, in bandit 13's home directory. To pass a key in, we use the `-i` flag with our `ssh` command, like so:

```
ssh -i sshkey.private bandit14@localhost
```

This logs us in as bandit 14. From here, we can navigate to the directory given to us and simply read the password.

The password to bandit 14 is `4wcYUJFw0k0XLSH1DzztnTBHixU3b3e`

Level 15

The password for the next level can be retrieved by submitting the password of the current level to port 30000 on localhost.

The first thing we need to do is open a connection to localhost, on port 30000. This can be done with the command

```
nc localhost 30000
```

Once our connection is opened, we simply need to input the password to bandit 14, and the server gives back the password to bandit 15. This is `BfMYroe26WYalil77FoDi9qh59eK5xNr`

Level 16

The password for the next level can be retrieved by submitting the password of the current level to port 30001 on localhost using SSL encryption.

Here we need to follow a similar process to on the previous level, but here we need to use an ssl connection rather than simple TCP. This can be done with the following command:

```
openssl s_client -connect localhost:30001
```

As before, we then simply input the previous password to get the next one. The password for bandit 16 is `cluFn7wTiGryunymYOu4RcffSxQluehd`

Level 17

The credentials for the next level can be retrieved by submitting the password of the current level to a port on localhost in the range 31000 to 32000. First find out which of these ports have a server listening on them. Then find out which of those speak SSL and which don't. There is only 1 server that will give the next credentials, the others will simply send back to you whatever you send to it.

The first thing that we need to do here is to find out which ports are open. This can be done using netcat, see below

```
nc -zv localhost 3100-3200
```

This tells us that there are five open ports. Using `openssl s_client` as above, we can test to see which ports are accepting ssl. When we find the correct port, we get an sshkey in response - this can be used to log into bandit17.

Level 18

There are 2 files in the homedirectory: `passwords.old` and `passwords.new`. The password for the next level is in `passwords.new` and is the only line that has been changed between `passwords.old` and `passwords.new`

All we need to do here is find the difference between two files. Fortunately, there's a command for that - `diff`. Running the command `diff passwords.old passwords.new` will identify which line has been changed, and give us the final password (as below). `ueksS7Ubh8G3DCwVzrTd8rAVOwq3M5x` The password to Bandit 18 is `kfBf3eYk5BPBRzwjqutbbfE887SVc5Yd`

Level 19

The password for the next level is stored in a file `readme` in the homedirectory. Unfortunately, someone has modified `.bashrc` to log you out when you log in with SSH.

As the guidance said, simply logging in with SSH here was no good, because whilst we could get onto the system we are immediately kicked off. However, as we also know that the password is stored in a file called `readme` in the home directory, I realised that I didn't need to log in and move around - I could simply take the file directly. This I did using the `scp` command, to copy a remote file to my desktop. This gave me the following command :

```
scp -P 2220 bandit18@bandit.labs.overthewire.org:readme ./tmp
```

Executing this (and entering the bandit18 password) logged into the server and copied the password file into the tmp folder in my working directory.

The password to bandit 19 is `IueksS7Ubh8G3DCwVzrTd8rAV0wq3M5x`

Level 20

To gain access to the next level, you should use the `setuid` binary in the homedirectory. Execute it without arguments to find out how to use it. The password for this level can be found in the usual place (`/etc/bandit_pass`), after you have used the `setuid` binary.

On logging in, we are met with an executable file called `bandit20-do`. The first thing I did was execute it (with `./bandit20-do`). This then told me that it would let me run a command as another user. As I knew that the password was in a file called `bandit20`, at path `/etc/bandit_pass/bandit20`, I ran the command

```
$ ./bandit20-do cat /etc/bandit_pass/bandit20
```

This revealed that the password for user `bandit20` is `GbKksEFF4yrVs6il55v6gwY5aVje5f0j`

Level 21

There is a `setuid` binary in the homedirectory that does the following: it makes a connection to `localhost` on the port you specify as a commandline argument. It then reads a line of text from the connection and compares it to the password in the previous level (`bandit20`). If the password is correct, it will transmit the password for the next level (`bandit21`).

As before, the first thing I did upon finding an executable was to run it and see what happened (using `./suconnect`). This told me that I needed to give it a port number to connect to. As my initial thoughts were that there was a process running on the server that I would need to communicate with, I did a quick port scan with `netcat` (`nc -zv localhost 1-20000`). However, upon running `suconnect`, I realised that this was not going on - it was reading from the port it was given. For that reason, I realised I needed to set up a simple web server, something I again decided to do with `netcat`. Setting up a server can be done with `nc -l localhost -p 20133`. However, we also need this server to serve up some data (`bandit20`'s password). This can be done with the following command

```
echo GbKksEFF4yrVs6il55v6gwY5aVje5f0j | nc -l localhost -p 20133
&
```

Echo simply repeats the text, the pipe operator means this is passed into our server and the single `&` operator means that this process will execute in the background. Once this has been set up, we can simply run `./suconnect 20133` to connect to this. At this point, it successfully reads the password, and serves up the next one.

The password to `bandit 21` is `gE269g2h3mw3pwgrj0Ha9Uoqen1c9DGr`

Level 22

A program is running automatically at regular intervals from cron, the time-based job scheduler. Look in `/etc/cron.d/` for the configuration and see what command is being executed.

Navigating into the relevant folder, and then examining `cronjob_bandit22` using `cat` we can see that it is executing the bash script, stored at `/usr/bin/cronjob_bandit22.sh`. Inspecting this, again using `cat`, reveals to us that the script is copying the contents of the password file to a hidden file, `/tmp/t706lds9S0RqQh9aMcz6ShpAoZKF7fgv`. Reading this gives us the password.

The password for bandit 22 is `Yk7owGAcWjwMVRwrTesJEwB7WV0iILLI`

Level 23

A program is running automatically at regular intervals from cron, the time-based job scheduler. Look in `/etc/cron.d/` for the configuration and see what command is being executed.

Here, as before, we first need to navigate to our `cron.d` file, and read the relevant files. Having a look in `cronjob_bandit23`, we are once again directed to a bash script, as below:

```
#!/bin/bash

myname=$(whoami)
mytarget=$(echo I am user $myname | md5sum | cut -d ' ' -f 1)

echo "Copying passwordfile /etc/bandit_pass/$myname to /tmp/$mytarget"

cat /etc/bandit_pass/$myname > /tmp/$mytarget
```

The `whoami` command returns the username - in this case that is going to be `bandit23`. The variable `'mytarget'` is assigned to the output of `(echo I am user $myname | md5sum | cut -d ' ' -f 1)`. We can therefore find the value of `mytarget` by running this command ourselves, but putting in `bandit23` as `$myname`. This gives us the address `8ca319486bfbbc3663ea0fbe81326349`. As we know that the user's password is being saved here, we can just check out the contents of this to get the password.

The password for bandit23 is `jc1udXuA1tiHqjIsL8yaapX5XIAI6i0n`

Level 24

A program is running automatically at regular intervals from cron, the time-based job scheduler. Look in `/etc/cron.d/` for the configuration and see what command is being executed.

This level starts in the same way as those before. By finding the bash script being executed by cron, we get this:

```
#!/bin/bash

myname=$(whoami)

cd /var/spool/$myname

echo "Executing and deleting all scripts in /var/spool/$myname:"
...
```

From this, we know that we need to write a bash script that can be executed and will reveal the password. In order to write this, I first created a directory I could work in – `/tmp/lloyd`. I then wrote the following bash script:

```
#!/bin/bash
cat /etc/bandit_pass/bandit24 > /tmp/lloyd/pass
```

This I saved as `script.sh`, set as executable by all users (using `chmod 777`), moved it to `/var/spool/bandit24` and waited for cron to work its magic. After a couple of minutes, the password for bandit24 was in my conveniently named pass file, and could be read with a simple `cat` command.

The password for bandit 24 is `UoMYTrfrBFHyQXmg6gzctqAw0mw1IohZ`

Level 25

A daemon is listening on port 30002 and will give you the password for bandit25 if given the password for bandit24 and a secret numeric 4-digit pincode. There is no way to retrieve the pincode except by going through all of the 10000 combinations, called brute-forcing.

Initially, I was tempted to use python to write the brute forcing script, as this is a scripting language I am particularly fond of, and most linux distributions have it installed by default. However, I realised that in some real world situations, this would not be an option, and I am working through this wargame to improve my own skills. For that reason, I decided to write a bash script instead.

As we know that we need to try all ~ 10 000 combinations, we need some kind of script. I created a script called `bruteforce.sh` with the command `touch bruteforce.sh` and then opened it with vim, a text editor.

I spent a period of time trying to write a bash script that would open a connection and then send the pincode + password combo. This is broadly what I would have done in python. However, I ran into some difficulty with this, as I kept getting problems with timeouts and the volume of processes I was trying to start. For that reason, I changed strategy, opting instead to save all the passwords into a list, and then dump this into the connection. One iteration of my first iteration was like this:

```
1 #!/bin/bash
2 password="UoMYTrfrBFHyQXmg6gzctqAw0mw1IohZ"
3
4 for a in `seq 0 9`; do
5     for b in `seq 0 9`; do
6         for c in `seq 0 9`; do
7             for d in `seq 0 9`; do
8                 code="$a$b$c$d"
9                 echo $password\ $code | nc localhost 30002 &* ^C
10            done
11        done
12    done
13 done
```

However, this was no good, so I replaced line 9 with the following: `echo $password\ $code >> passwords.tx`. This created a list of possible combinations. This can then be passed into the listener using `cat passwords.txt | nc localhost 30002`. This revealed the password.

The password of user bandit25 is `uNG9058gUE7snukf3bvZ0rxhtnjzSGzG`

Level 26

Logging in to bandit26 from bandit25 should be fairly easy... The shell for user bandit26 is not `/bin/bash`, but something else. Find out what it is, how it works and how to break out of it.

We need to find out what shell user bandit26 is using; this information is likely to be stored in the `/etc/passwd` file. So, we take a look in here with `cat /etc/passwd | grep bandit26`. This command gives us the output:

```
bandit26:x:11026:11026:bandit level 26:/home/bandit26:/usr/bin/showtext
```

A quick google search tells us that the final item in this reading is the shell being used. So, we have a look at that with `cat /usr/bin/showtext`. This gives us the following script:

```
#!/bin/sh

export TERM=linux
```

```
more ~/text.txt
exit 0
```

Checking permissions with `chmod` tells us that we cannot edit this file - this was my first thought. Similarly, there was no mileage in editing the `/etc/passwd` file - this also had the correct permissions enabled. In the end, I was seriously stuck, so ended up googling for a hint - a quick search pointed me to the fact that the way to prevent the script from fully executing was to trigger the `more` command in said script. This can be done by making the terminal window as small as possible. How does this help? Well, from here we can enter vim (by pressing `v`), and then once in vim we have all the handy commands that come with this - including, crucially, the ability to open a different file. In this instance, the file we want to open is `/etc/bandit_pass/bandit26`. This can be opened with:

```
:e /etc/bandit_pass/bandit26
```

The password for bandit 26 is: `5czgV9L3Xx8JP0yRbXh6lQbmIOWvPT6Z`

Level 27

Good job getting a shell! Now hurry and grab the password for bandit27!

Overthewire.org made one crucial mistake here - they overestimated my success from the previous level. However, this level did give me the perfect opportunity to actually learn what they wanted me to learn from the previous level, and get a shell. This I did by running `:set shell=/bin/bash` from within vim. From here, I was free to move around and operate as if I were in a normal terminal instance, except all my commands had to be preceded with `:!.`

The first thing I did was to run `:! ls`, to know what I was dealing with. This revealed the `text.txt` file that we knew about, and a `bandit27-do` executable. I duly executed this. This was a SETUID binary, in a similar way to the one we saw a few levels ago. As such, I simply used it to read the `/etc/bandit_pass/bandit27` file using `cat`.

The password for bandit27 is `3ba3118a22e93127a4ed485be72ef5ea`

Level 28

There is a git repository at `ssh://bandit27-git@localhost/home/bandit27-git/repo`. The password for the user `bandit27-git` is the same as for the user `bandit27`.

All we need to do here is to clone the git repo to the local machine. In order to do this, we first create a new directory in `/tmp/`. Then, we can simply run `git clone` plus the address that the instructions gave us. This creates a new folder

called `repo`, navigating into here we find a file called `README`, opening this we get the password.

The password for bandit28 `0ef186ac70e04ea33b4c1853d2526fa2`

Level 29

There is a git repository at `ssh://bandit28-git@localhost/home/bandit28-git/repo`. The password for the user `bandit28-git` is the same as for the user `bandit28`.

Again, here we navigate to our `tmp` directory, and work from here. We run `git clone` and then the address that we were given. When we are prompted for a password, we simply enter the password for `bandit28` – this authenticates the connection. We can then proceed in the same way as in the previous level.

However, upon inspecting the `README` file, we find that `bandit29`'s password has been redacted, and replaced with `xxxxxxxxxxxxxxxxxxxxxx`. No trouble, however, as git is a version control system. All we need to do here is to run `git log`, to get a record of the changes that were made. Bingo - here we find out that the most recent commit was to `fix info leak`. Presumably, then, all we need to do to checkout a previous version of the file, and that will contain the password. This can be done with

```
git checkout c086d11a
```

The hash being the signature of the commit we want to take a look at. We take a look into the `README` file now and, bingo - got it.

The password for `bandit29` is `bbc96594b4e001778eee9975372716b2`

Level 30

is a git repository at `ssh://bandit29-git@localhost/home/bandit29-git/repo`. The password for the user `bandit29-git` is the same as for the user `bandit29`.

Begin as in the previous two levels. Again, simply reading the `README.md` file bears no fruits - we are told that there are “no passwords in production”. This strikes me as a suggestion that there may be another branch, containing the password, as production in git often occurs on topic branches. Unfortunately, that's not the case in this scenario - I checked with `git branch` and discovered that the only branch that existed was the one checked out, `master`. However, running `git log --all` revealed some extra commits, that hadn't been shown with a simple `git log`. One of these was conveniently titled `add some silly exploit, just for shit and giggles` so let's have a look at this, again using `git checkout <commit hash>`.

We now have an extra folder called `exploits` that we can take a look at. Unfortunately, this contains just one file `horde5.md`, which is empty. Checking the permissions on this file bears no fruit - we discover that it has permissions 644 (rw for root, r only for others) and as such doesn't contain some secret script. Moving further back in the log, we can also see a commit `add gif2ascii`. Having a look at this is, unfortunately, similarly disappointing - `gif2ascii.py` is also an empty file. However, running `git diff 786d5` from the commit that added `gif2ascii.py`, we discover that they are actually the same file, simply renamed. I continued to explore these files for a period of time, trying to run them with python and seeing if there was anything that I could do to get more out of them. There was not.

At this point, I turned back to my initial idea that there may be more branches than meet the eye - the "no passwords in production" sentence gave me this vibe, and it was reinforced by the fact that `git log --all` showed more than a simple `git log` - this suggests that there were more branches. So, a quick bit of research later, I found out that git clone doesn't automatically bring all branches to your local machine, and running `git branch -r` would allow us to check remote branches. Sure enough, there were several remote branches that hadn't come up previously. One such branch was `origin/dev` - this contained the password. Interestingly, the commit that added it `add data needed for development` had also been present when I ran `git log --all` - I just missed it as I was distracted by `add some exploit for shits n giggles`.

The password for bandit 30 is `5b90576bedb2cc04c86a9e924ce42faf`

Level 31

There is a git repository at `ssh://bandit30-git@localhost/home/bandit30-git/repo`. The password for the user `bandit30-git` is the same as for the user `bandit30`.

Start in the same way as before. Reading the `README.md` file gives us the following:

```
just an empty file... muahaha
```

Interesting. `Git log --all` shows us that there is only one commit, and `git branch -r` shows us that there are no hidden branches. What else then? After digging through the `.git` directory for a while and having no luck, I remembered about git tagging. Running `git tag` should reveal to us any tags that had been applied to the git log in the past - sure enough, this tells us of a `secret` tag. Unfortunately, this does not actually seem to point at anything - I was unable to check out to `tags/secret`. However, using `git show` I was able to get what I initially thought was the hash of a commit, but when this wasn't the case I thought maybe it was bandit 31's password. It was.

Bandit 31's password is `47e603bb428404d265f59c42920d81e5`

Level 32

There is a git repository at `ssh://bandit31-git@localhost/home/bandit31-git/repo`. The password for the user `bandit31-git` is the same as for the user `bandit31`.

Start as before. Reading the README, we are told that this time we need to push a file called `key.txt` with contents `May I come in?`. We can create this file with

```
echo "May I come in?" > key.txt
```

And then commit it + push it with

```
git commit . -m "add key file" && git push
```

We are then prompted for `bandit31`'s password; entering this we are told that there is nothing to commit, working tree is clean. Not quite what we were hoping for. Instead, we can try `git add key.txt` – doing so tells us that it is included in the `.gitignore`. We can get past this by simply running `git add key.txt -f`. Then, running `git push` sorts us out and gives us the file.

The password for bandit 32 is `56a9bf19c63d650ce78e6ec0354ee45e`

Level 33

After all this git stuff its time for another escape. Good luck!

On logging in, we are greeted with `WELCOME TO THE UPPERCASE SHELL`. Running commands is, seemingly, impossible – anything we type in is met with `sh: 1: <COMMAND>: not found`. Problematic. I realised that this uppercase shell must be a script that was started from elsewhere. I also found that the variable `$0` refers to the script from which the current shell was started – “If Bash is invoked with a file of commands (see Section 3.8 [Shell Scripts], page 39), `$0` is set to the name of that file.”

This is useful, as `$0` is not a command that will be affected by the shift to upper case. After having run it, we are back out into a more usable shell. From here, simply running `cat /etc/bandit_pass/bandit33` reveals the password. N.B. I couldn't actually work out how to exit ssh from here so I force stopped it.

The password for bandit 33 is `c9c3199ddf4121b10cf581a98d51caee`

Level 34

This level doesn't exist as of yet. Game complete :)