Part (b):

Provide the appropriate git command to perform the following operations:

1. Stage an untracked file to be committed. The file is called 'hw1q2b.cpp'.
2. Display the details of the last three commits in the repository.

Part (c)

What is the full command to re-clone your private CSCI104 repo to your VM? Assume you are in an appropriate folder.

Problem 3 - Runtime Analysis (24%)

In Big-Θ notation, analyze the running time of the following three pieces of code/pseudo-code. Describe it as a function of the input (here, `n`). Submit your answers as a PDF (using some kind of illustration software or scanned handwritten notes where you use your phone to convert to PDF) showing your work and derivations supporting your final answer. **You must name the file `q3_answers.pdf`**. As usual, answers without supporting work will receive 0 credit.

Part (a)

```
void f1(int n)
{
  int t = sqrt(n);
  for(int i = 0; i < n; i++){
    for(int j = 0; j < n; j++){
      // do something O(1)

    }
    n -= t;
  }
}
```

Part (b)

```
Assume A is an array of size n+1.

void f2(int* A, int n)
{
  for(int i=1; i <= n; i++){
    for(int k=1; k <= n; k++){
      if( A[k] == i){
        for(int m=1; m <= n; m=m+m){
          // do something that takes O(1) time
          // Assume the contents of the A[] array ar
        }
      }
    }
```

3) A)

$$T(n) = \sum_{i=0}^{\sqrt{n}-1} \left(O(1) + \sum_{j=0}^{n-1} O(1) \right)$$

$$T(n) = O(\sqrt{n}) + \sum_{i=0}^{\sqrt{n}-1} O(n)$$ Not data dependent

$$T(n) = O(n) + O(n^{\frac{3}{2}})$$

$$\boxed{T(n) = O(n^{\frac{3}{2}})}$$

3) B)

$$T(n) = \sum_{i=1}^{n} \left(\sum_{k=1}^{n} \left[O\left(\sum_{m=1}^{\log n} O(1) \right) \right] \right)$$

$$T(n) = \sum_{i=1}^{n} \left(\sum_{k=1}^{n} \left(O(1) + (1) O(\log n) \right) \right)$$

$$T(n) = \sum_{i=1}^{n} \left(\sum_{k=1}^{n} (O(1)) \right) + \sum_{k=1}^{n} O(\log n)$$

$$T(n) = O(n^2) + O(n \log n)$$

$$\boxed{T(n) = O(n^2)}$$

3) C)

$T(n-4)$ → $T(n-4)$ → $T(n-4)$

$O(1) + T(n-2) + T(n-2)$

$O(3) + 4 \cdot T(n-4)$

$O(7) + 8 \cdot T(n-6)$

This process keeps continuing

k^{th} iteration $= O(c) + 2^k \cdot T(n-2k)$

stops when $= 0$

$n - 2k = 0$

$n = 2k$

$k = \frac{n}{2}$

$2^{\frac{n}{2}} \cdot (n - 2\frac{n}{2})$

$2^{\frac{n}{2}} \cdot (n-n)$

$= 2^{\frac{n}{2}}$

$= \boxed{O(2^n)}$

The function contains an if else statement. The else which will run 2^n times because the recursion occurs twice in each iteration so, eventually each recursion has 2 recursions, thus causing the runtime to become exponential 2^n. This continues until the k^{th} iteration $= 0$. So that's why solve for $n-2k=0$, plugging in gives us $O(2^{\frac{n}{2}}(n-n))$, cancels out to $O(2^n)$

```
  }
}
```

Part (c)

```
void f3(int* A, int n)
{
  if(n <= 1) return;
  else {
    f3(A, n-2);
    // do something that takes O(1) time
    f3(A, n-2);
  }
}
```

Part (d)

Notice that this code is very similar to what will happen if you keep inserting into an ArrayList (e.g. `vector`). Notice that this is **NOT** an example of amortized analysis because you are only analyzing 1 call to the function `f()`. If you have discussed amortized analysis, realize that does NOT apply here since amortized analysis applies to *multiple* calls to a function. But you may use similar ideas/approaches as amortized analysis to analyze this runtime. If you have NOT discussed amortized analysis, simply ignore it's mention.

```
int f (int n)
{
  int *a = new int [10];
  int size = 10;
  for (int i = 0; i < n; i ++)
     {
        if (i == size)
          {
             int newsize = 4*size;
             int *b = new int [newsize];
             for (int j = 0; j < size; j ++) b[j] = 
             delete [] a;
             a = b;
             size = newsize;
          }
        a[i] = i*i;
     }
}
```

Problem 4 - NOT GRADED (PRACTICE ONLY) - Constructors and Destructors (0%)

Place your answers to the questions below in the same file as your answers to problem 1 and 2 (i.e. `hw1.txt`). You do not need to test and compile the code below. You are just writing out your answers in the `hw1.txt` file.

3) D) $10 \cdot 4^k$

$i=0$, size $= 1$

$i=1$, size $= 40$

$i=2$, size $= 160$

$i=3$, size $= 2560$

$i=k$, size $= 10(4)^k$

$$\sum_{i=0}^{n-1} \Theta(1) + \sum_{j=0}^{\log_4 \frac{n}{10}} \Theta(1)$$

$$T(n) = \Theta(n) + \Theta(\log_4 n)$$

$$\boxed{T(n) = \Theta(n)}$$

$n = 10(4)^k$

$\frac{n}{10} = 4^k$

$\log_4 \frac{n}{10} = k$

The outside runs n times while the inside will run in the k^{th} iteration $\log_4 \frac{n}{10}$ times. When we split the summation n is a greater function than any $c \cdot \log_k n$ function, thus we only care about n. Size also will only even in the worst case not be called every time.