Implement Git

Tutorial: https://www.leshenko.net/p/ugit

Project: https://github.com/ReZeroS/zit

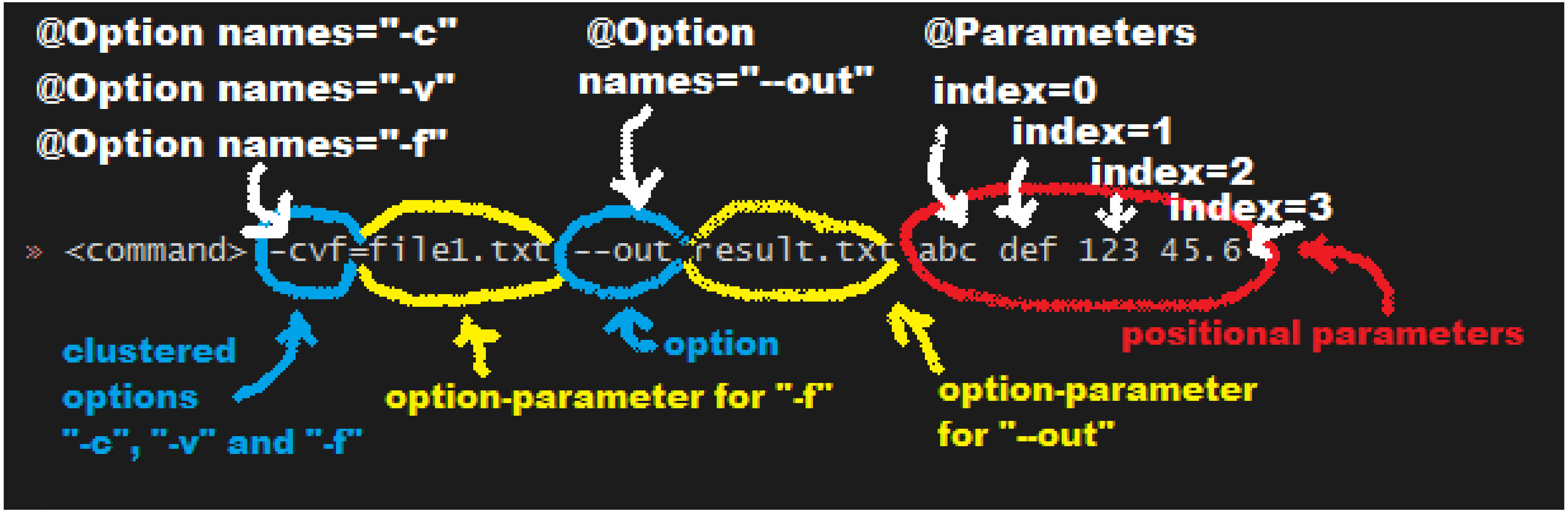
Dependency

- MAIN
  - Lombok https://projectlombok.org code enhancer
  - Guava https://github.com/google/guava common tools
  - Picocli https://picocli.info command line parser

- Others
  - Logback logs library
  - Gson JSON converter
  - JNR-POSIX cd command

Arguments parser

- Command line arguments can be separated into ***options*** and ***positional parameters***.
- Options have a name, positional parameters are usually the values that follow the options, but they may be mixed.



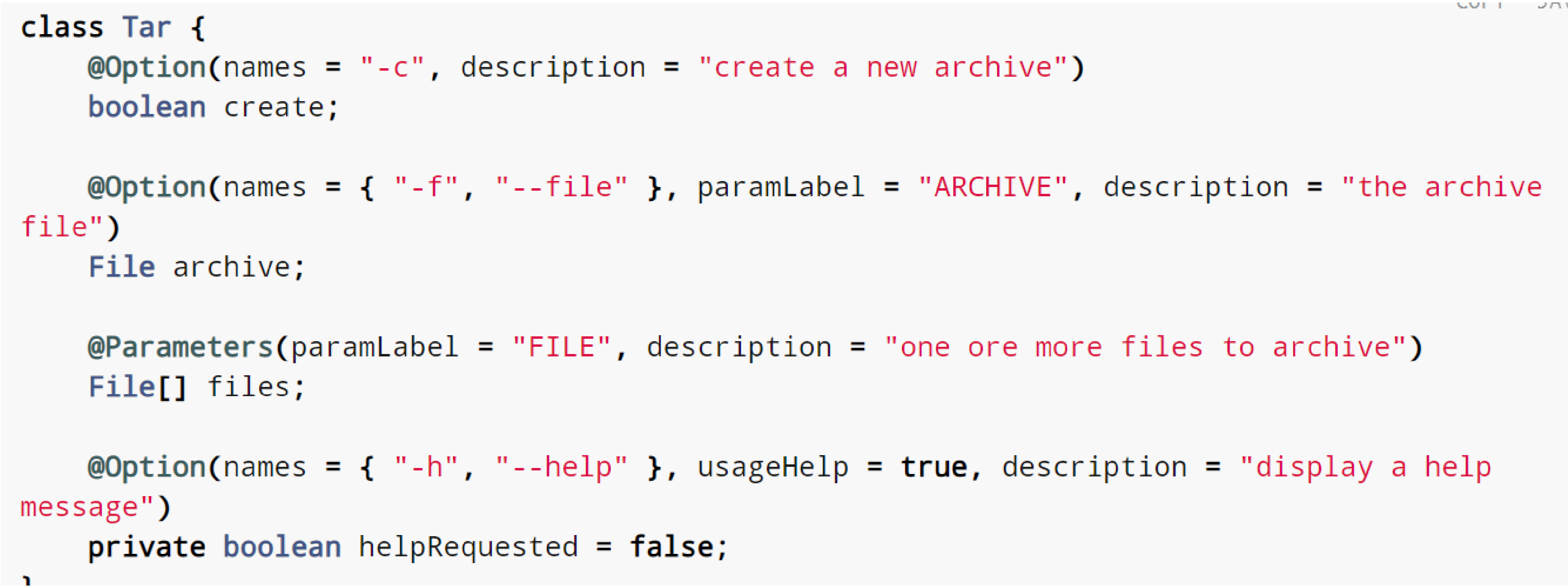
```
» <command> -cvf=file1.txt --out result.txt abc def 123 45.6
```

Example:

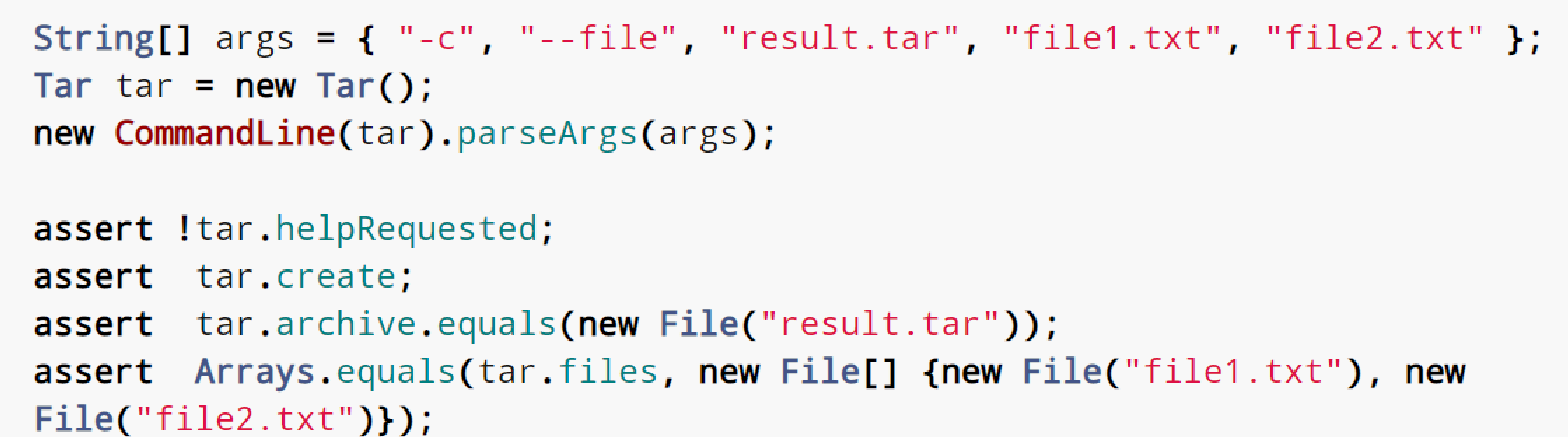
options with one or more names

options that take
an option parameter,

and a help option.



```
class Tar {
    @Option(names = "-c", description = "create a new archive")
    boolean create;

    @Option(names = { "-f", "--file" }, paramLabel = "ARCHIVE", description = "the archive
file")
    File archive;

    @Parameters(paramLabel = "FILE", description = "one ore more files to archive")
    File[] files;

    @Option(names = { "-h", "--help" }, usageHelp = true, description = "display a help
message")
    private boolean helpRequested = false;
}
```



```
String[] args = { "-c", "--file", "result.tar", "file1.txt", "file2.txt" };
Tar tar = new Tar();
new CommandLine(tar).parseArgs(args);

assert !tar.helpRequested;
assert  tar.create;
assert  tar.archive.equals(new File("result.tar"));
assert  Arrays.equals(tar.files, new File[] {new File("file1.txt"), new
File("file2.txt")});
```

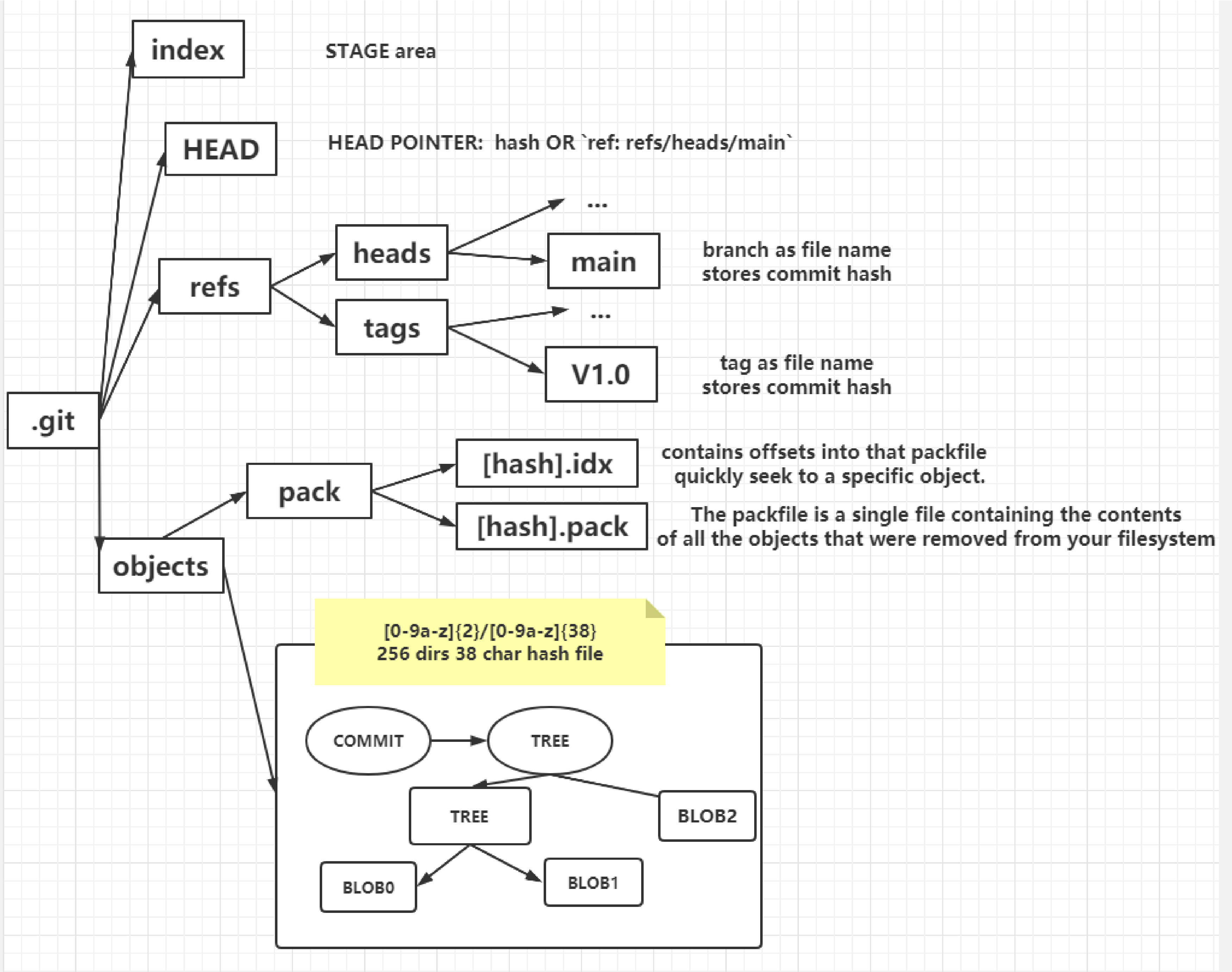
file system

Others:

remotes: remote repository
refs/remotes/remote-branch

hooks: hooks script
.git/hooks

config, description and …



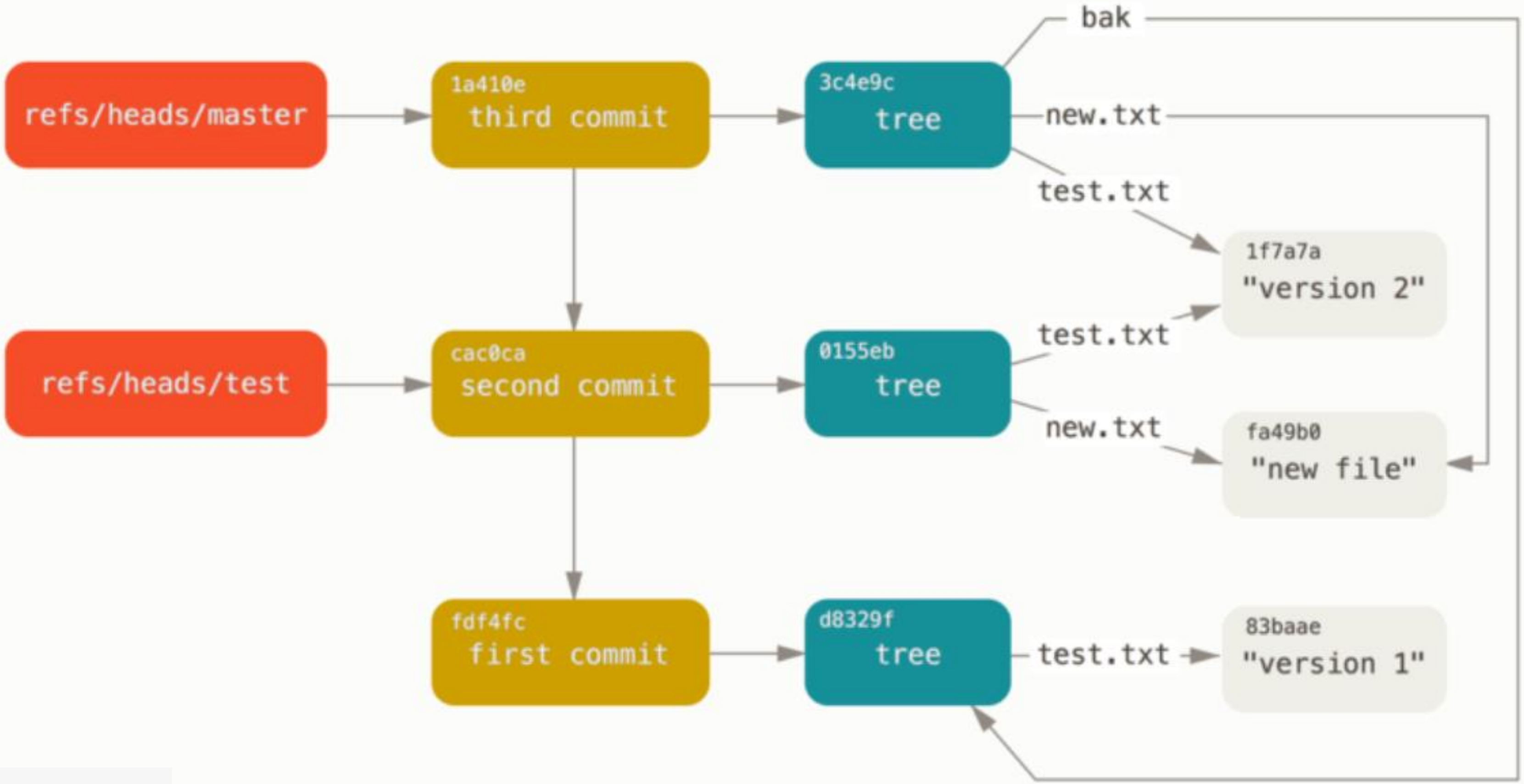
.git
index
STAGE area
HEAD
HEAD POINTER: hash OR `ref: refs/heads/main`
refs
heads
...
main
branch as file name
stores commit hash
tags
...
V1.0
tag as file name
stores commit hash
objects
pack
[hash].idx
contains offsets into that packfile
quickly seek to a specific object.
[hash].pack
The packfile is a single file containing the contents
of all the objects that were removed from your filesystem
[0-9a-z]{2}/[0-9a-z]{38}
256 dirs 38 char hash file
COMMIT
TREE
TREE
BLOB2
BLOB0
BLOB1

Git Objects & References

```
$ git update-ref refs/heads/master 1a410efbd13591db07496601ebc7a059dd55cfe9
```

```
$ find .git/refs
.git/refs
.git/refs/heads
.git/refs/tags
```

```
$ cat .git/HEAD
ref: refs/heads/master
```

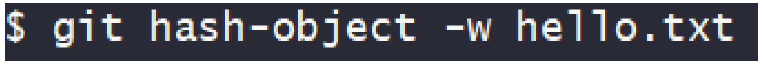


Detached HEAD

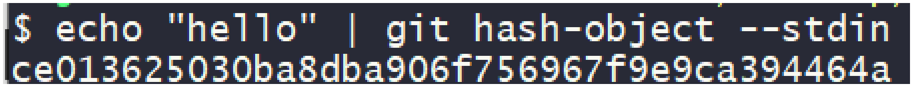
```
$ cat .git/HEAD
140a4ceae12c237f9f23321aa5e29d8d14852f6f
```

basic command

- hash-object

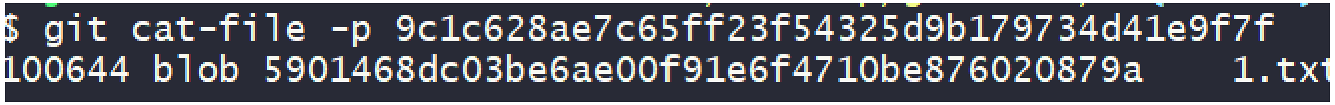


```
$ git hash-object -w hello.txt
```



```
$ echo "hello" | git hash-object --stdin
ce013625030ba8dba906f756967f9e9ca394464a
```

  - Write file into object database and return hash address
- cat-file



```
$ git cat-file -p 9c1c628ae7c65ff23f54325d9b179734d41e9f7f
100644 blob 5901468dc03be6ae00f91e6f4710be876020879a    1.txt
```

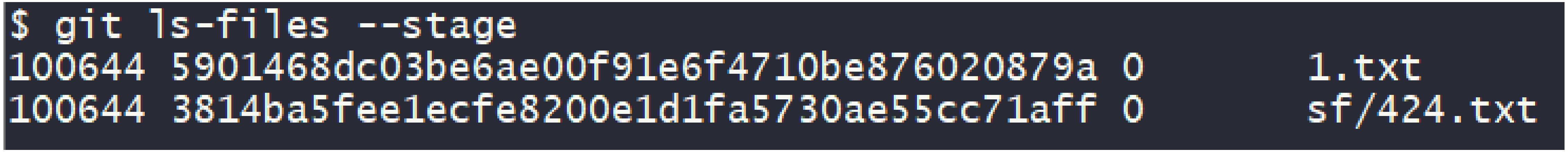
  - Read file from object database by the hash address

- write-tree
  - Do hash-object blob with files
  - Execute hash-object tree according to the stage
- read-tree
  - Read the tree information given by <tree-id> into the index
  - Does not actually **update** any of the files it "caches“

- Git add
  - do hash-object
  - rewrite INDEX

- Git update-index --add file
  - Add blob into index file

- Git ls-stages
  - check the stages file



```
$ git ls-files --stage
100644 5901468dc03be6ae00f91e6f4710be876020879a 0       1.txt
100644 3814ba5fee1ecfe8200e1d1fa5730ae55cc71aff 0       sf/424.txt
```

- INDEX file format

Git gc (push)

- Zlib to compress
- Loose object format
- Git verify-pack –v .git/objects/pack/pack-[hash].index
    - The new version of the file is the one that is stored intact
    - whereas the original version is stored as a delta

    - this is because you’re most likely to need faster access to the most recent version of the file.

- Git commit
  - call write-tree
  - Set current commit which is HEAD pointed as parent commitId
  - Save typed words about the commit as commit message
  - Do hash-object with the content (commit, tree, parent) with type called ‘commit’
  - Update head pointer to the new commit
  - Update ref[branch] file content

- commit --amend [gc] since old commit won’t be used

- Git commit-tree treeId msg
  - Every commit has a single tree
  - Generate a commit by the provided tree id

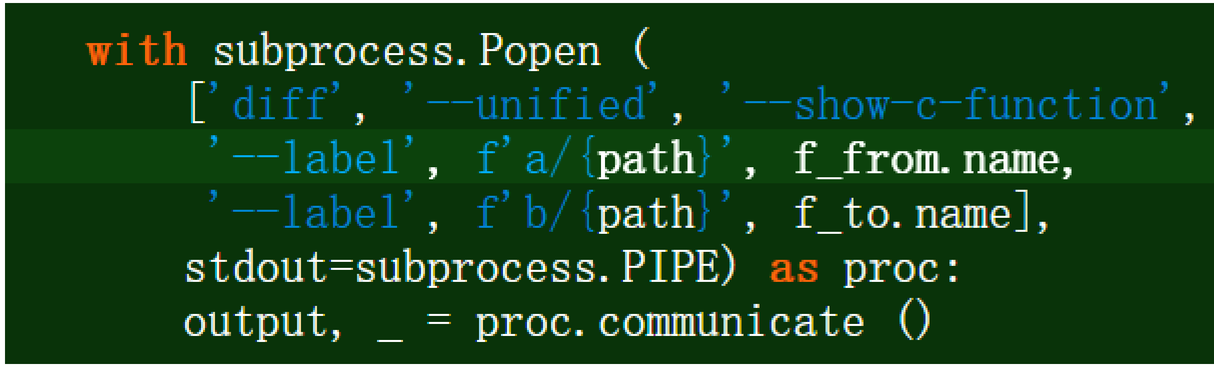
- Git branch
    - List all branch
        - List all files in the heads directory.
    - Create new branch
        - Create new branch with a target commit
        - If the target commit not provided, then set current commit as target commit.

- Git checkout ref
    - Get the ref's hash-id
    - Get the commit tree by the above hash-id
    - Read tree to update Index and reset the working directory
    - Update **HEAD** file to set content as branch ref or hash(detached HEAD)

- Git reset
    - Update HEAD as the target commit id

```
Good:    class Foo
           def initialize(name)
             @name = name
           end
      +
      +    def inspect
      +      @name
      +    end
         end
```

```
Bad:     class Foo
           def initialize(name)
             @name = name
      +    end
      +
      +    def inspect
      +      @name
           end
         end
```



```
with subprocess.Popen (
    ['diff', '--unified', '--show-c-function',
     '--label', f'a/{path}', f_from.name,
     '--label', f'b/{path}', f_to.name],
    stdout=subprocess.PIPE) as proc:
    output, _ = proc.communicate ()
```

Myers' algorithm is just one such strategy, but it's fast and it produces diffs that tend to be of good quality most of the time.

It does this by being **greedy**, that is **trying to consume as many lines** that are the same **before making a change** (therefore avoiding the "wrong end" problem), and also by **preferring deletions over insertions** when given a choice, so that deletions appear first.

Diff Definition： Shortest Edit Scripts

- **Definition**: A text and B text
- **Requirements**: Convert A text to B text
- The smallest edit **range**： single line
- **Edit script**
  - **Delete** line from **A** text
  - **Add** line from **B** text
  - **Sync** the same line at **A** and **B**
- **Weight**： sync consume 0 while each of delete and add consume 1

Requirements:

convert A to B with SES

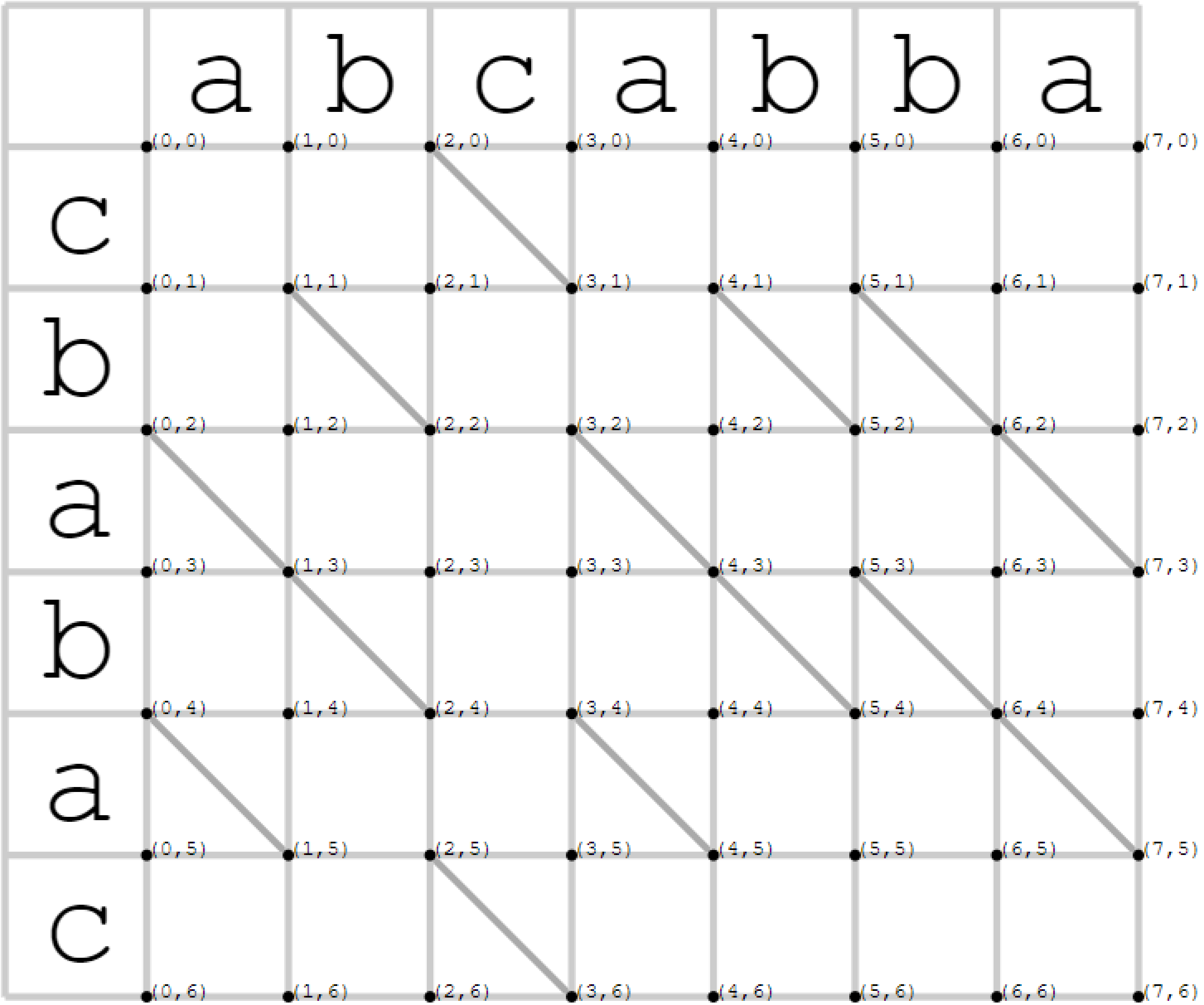
Initial:

String A: **abcabba**

String B: **cbabac**

Diagonal lines:

represent items that match

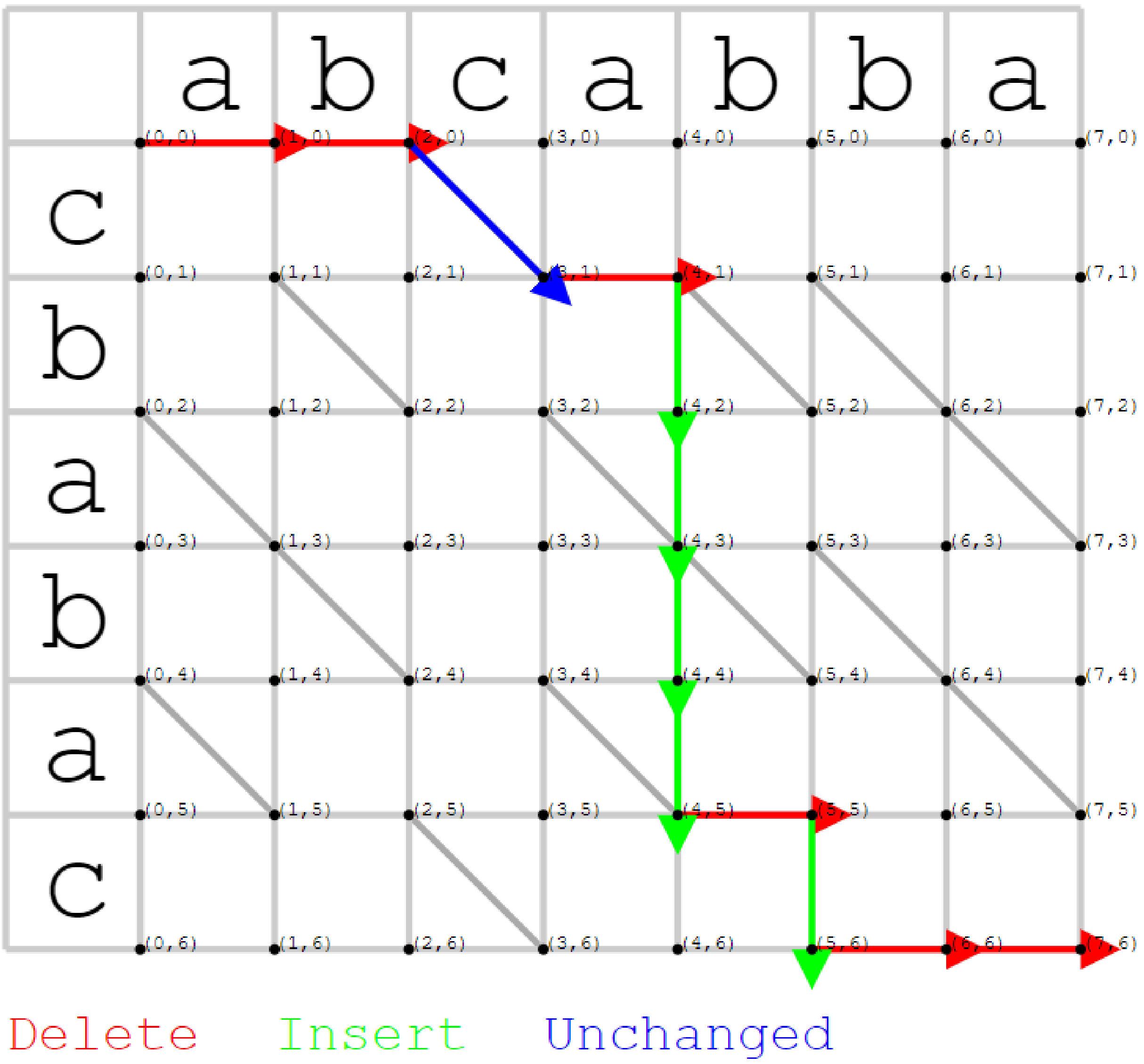


Snake Define

a single deletion or insertion
followed by zero or more diagonals.

Longest Common Subsequence (LCS)

Finding the SES is equivalent to finding
the Longest Common Subsequence of the two files.

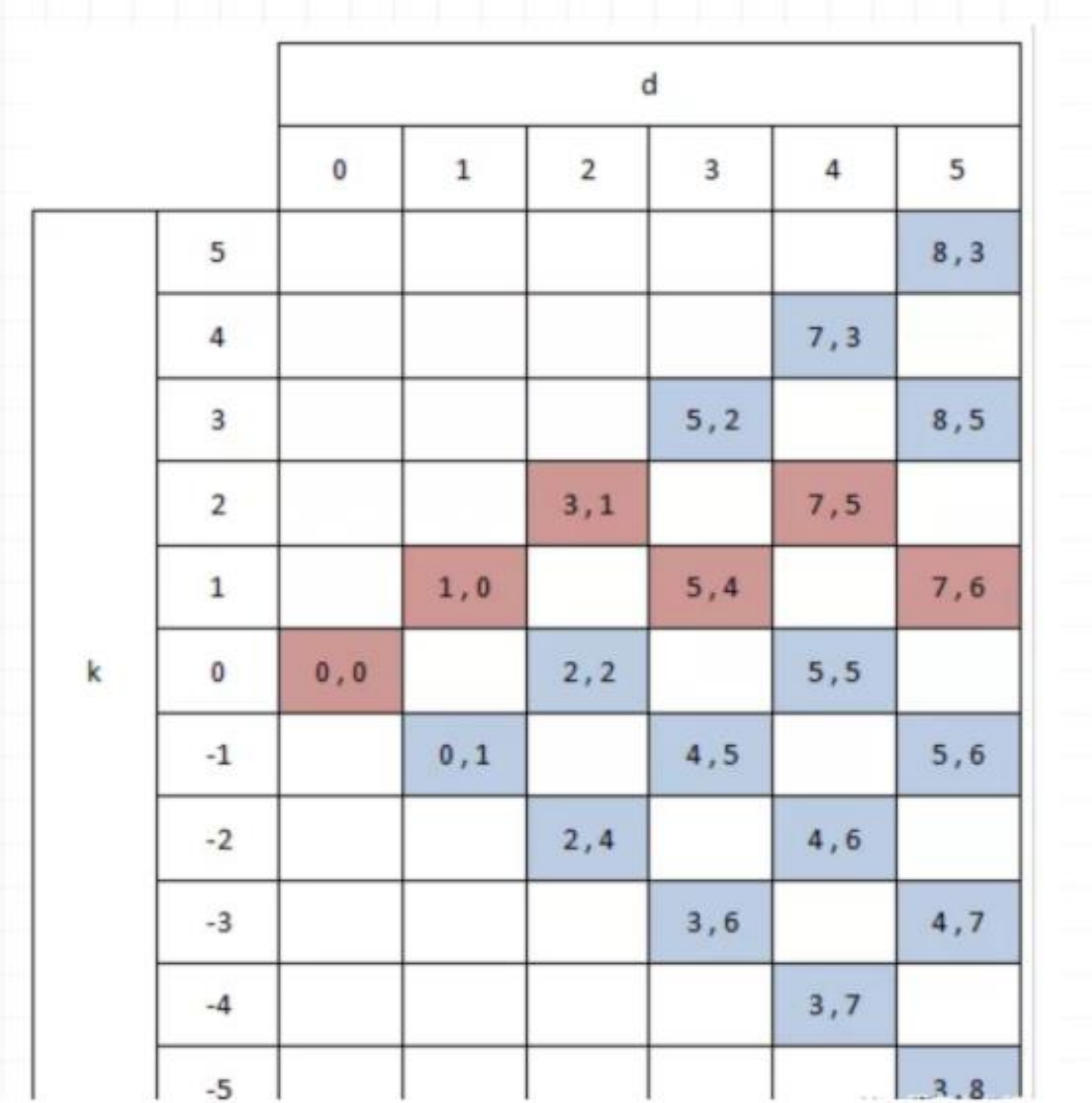


d => (i + diagonal, j + diagonal)
diagonal parameter is the count of diagonal passed

define k = x – y = i – j
then the even or odd quality of k depends on d

Explain:
If d is odd, then **i + j** will be odd since 2*diagonal is even)
So i – j will be odd => d odd => k odd

Any point arrived at k diagonal line must have gone through at least |k| operations

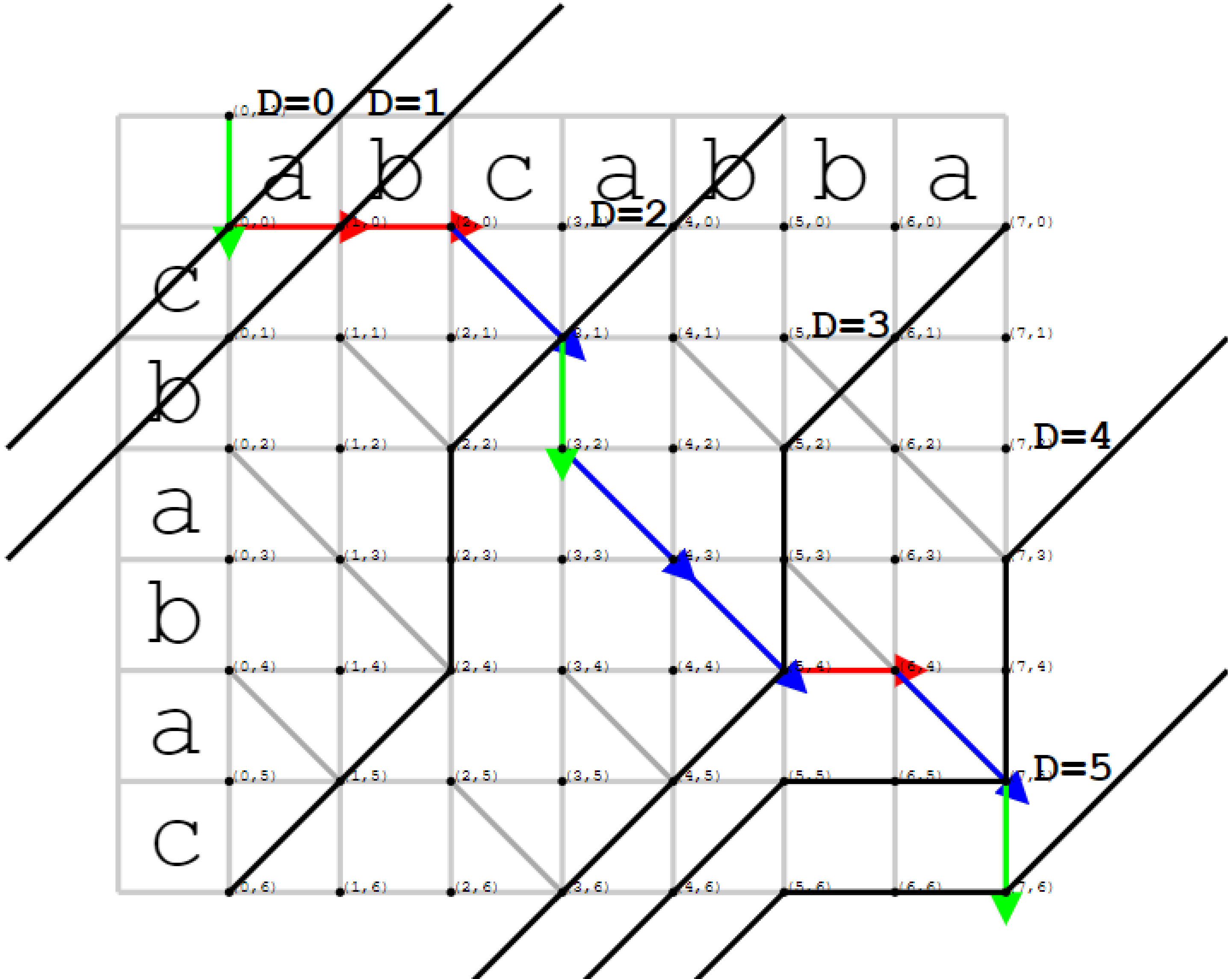


		d					
		0	1	2	3	4	5
	5						8,3
	4					7,3	
	3				5,2		8,5
	2			3,1		7,5	
	1		1,0		5,4		7,6
k	0	0,0		2,2		5,5	
	-1		0,1		4,5		5,6
	-2			2,4		4,6	
	-3				3,6		4,7
	-4					3,7	
	-5						3,8

Visualize

Best X Values (forward)

k	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13
V[k]	?	?	?	?	?	?	?	?	3	3	4	4	5	5	7	7	5	7	?	?	?	?	?	?	?	?	?



V[k] = x
=> y = x -k

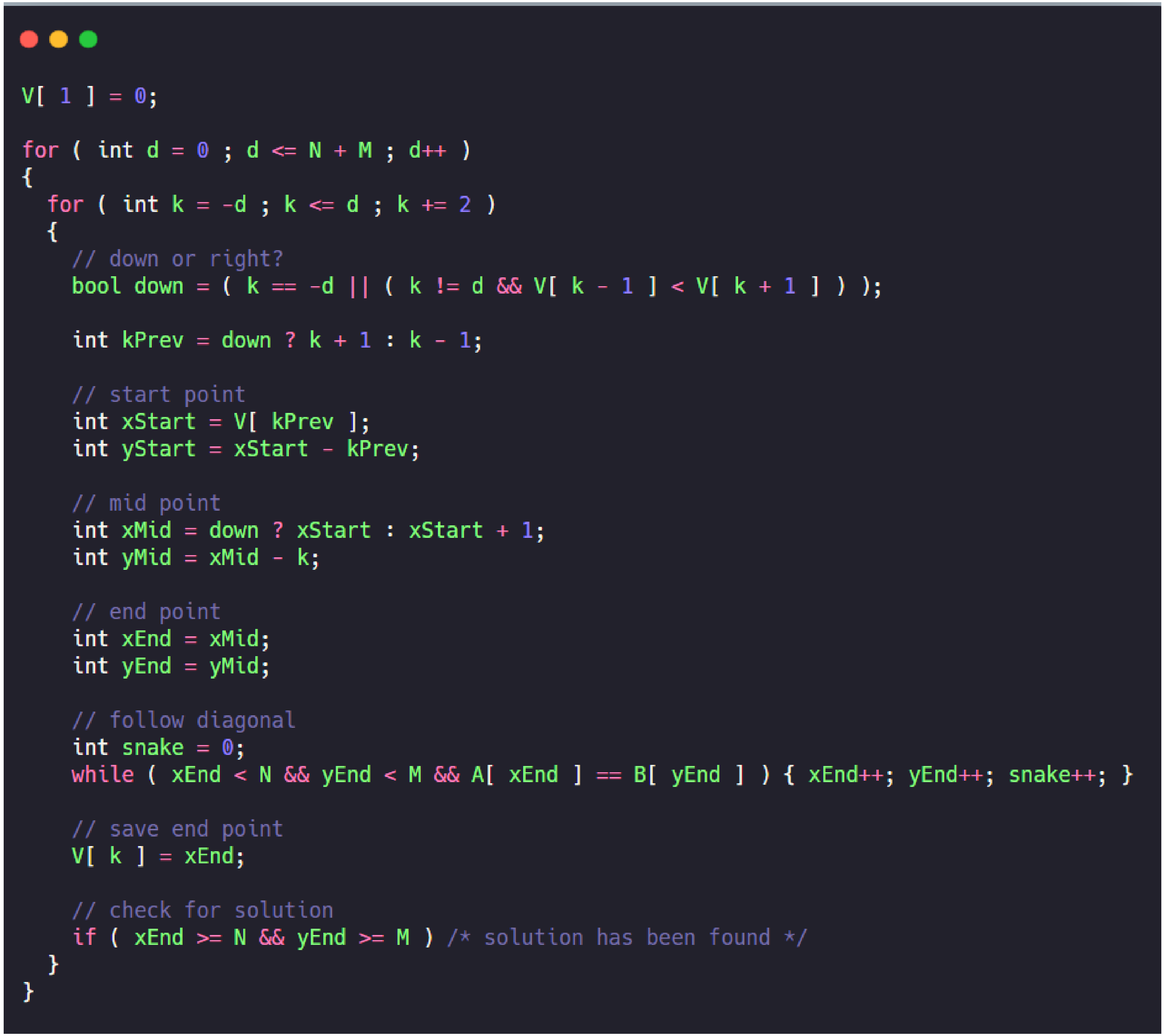
Outter loop to limit steps
【What is the status at any step?】

Inner loop decide the choice
【We are from the up or the left?】

When you know where are you from
You will get Snake’s start point and
initial middle point

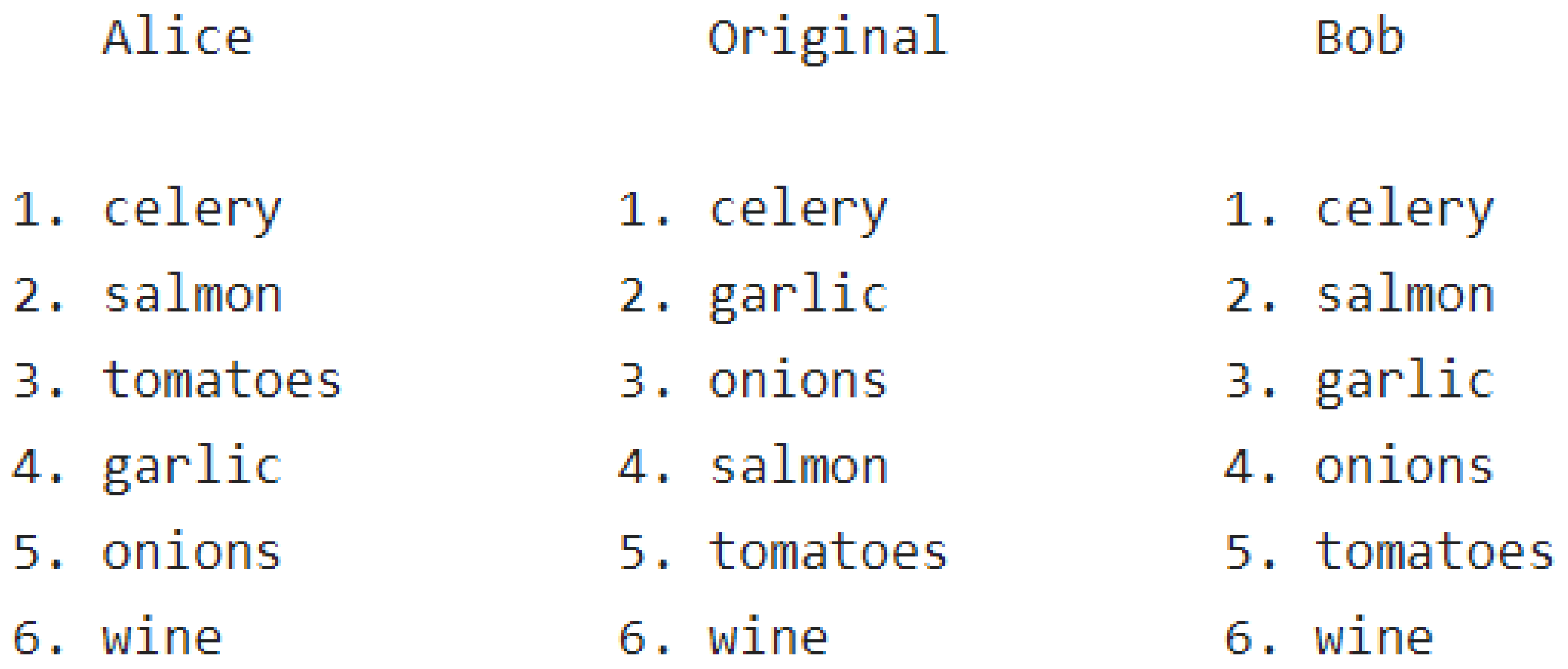
And this time middle point also is end point, they are at the same position

Then if there are many diagonal ways after the middle point, the end point will move around the way



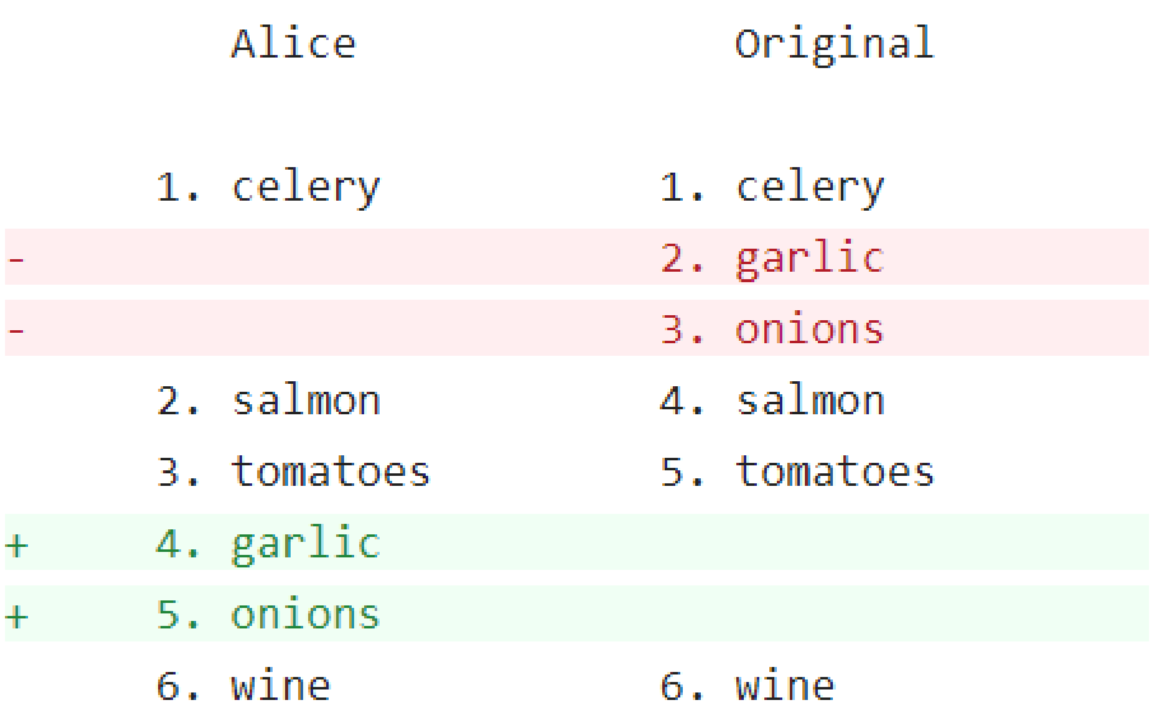
```
V[ 1 ] = 0;

for ( int d = 0 ; d <= N + M ; d++ )
{
  for ( int k = -d ; k <= d ; k += 2 )
  {
    // down or right?
    bool down = ( k == -d || ( k != d && V[ k - 1 ] < V[ k + 1 ] ) );

    int kPrev = down ? k + 1 : k - 1;

    // start point
    int xStart = V[ kPrev ];
    int yStart = xStart - kPrev;

    // mid point
    int xMid = down ? xStart : xStart + 1;
    int yMid = xMid - k;

    // end point
    int xEnd = xMid;
    int yEnd = yMid;

    // follow diagonal
    int snake = 0;
    while ( xEnd < N && yEnd < M && A[ xEnd ] == B[ yEnd ] ) { xEnd++; yEnd++; snake++; }

    // save end point
    V[ k ] = xEnd;

    // check for solution
    if ( xEnd >= N && yEnd >= M ) /* solution has been found */
  }
}
```

merge

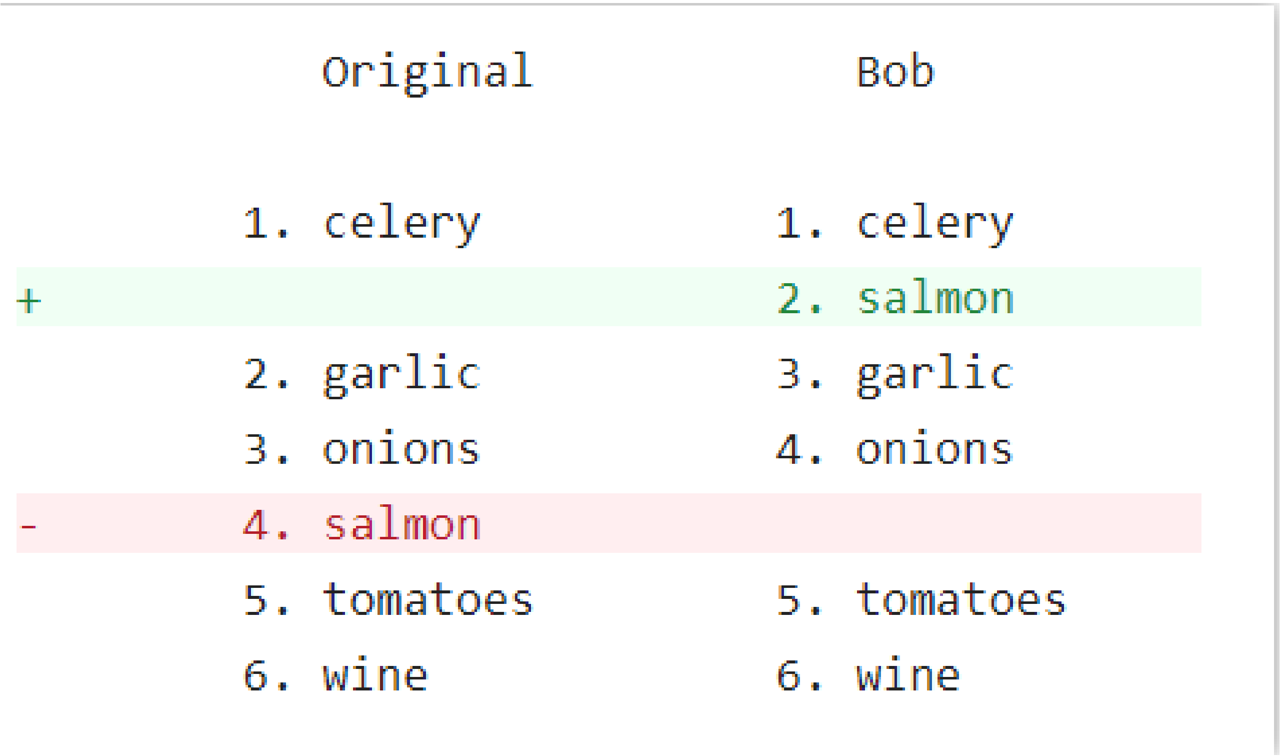


```
       Alice              Original             Bob

   1. celery          1. celery          1. celery
   2. salmon          2. garlic          2. salmon
   3. tomatoes        3. onions          3. garlic
   4. garlic          4. salmon          4. onions
   5. onions          5. tomatoes        5. tomatoes
   6. wine            6. wine            6. wine
```

Initial with diff

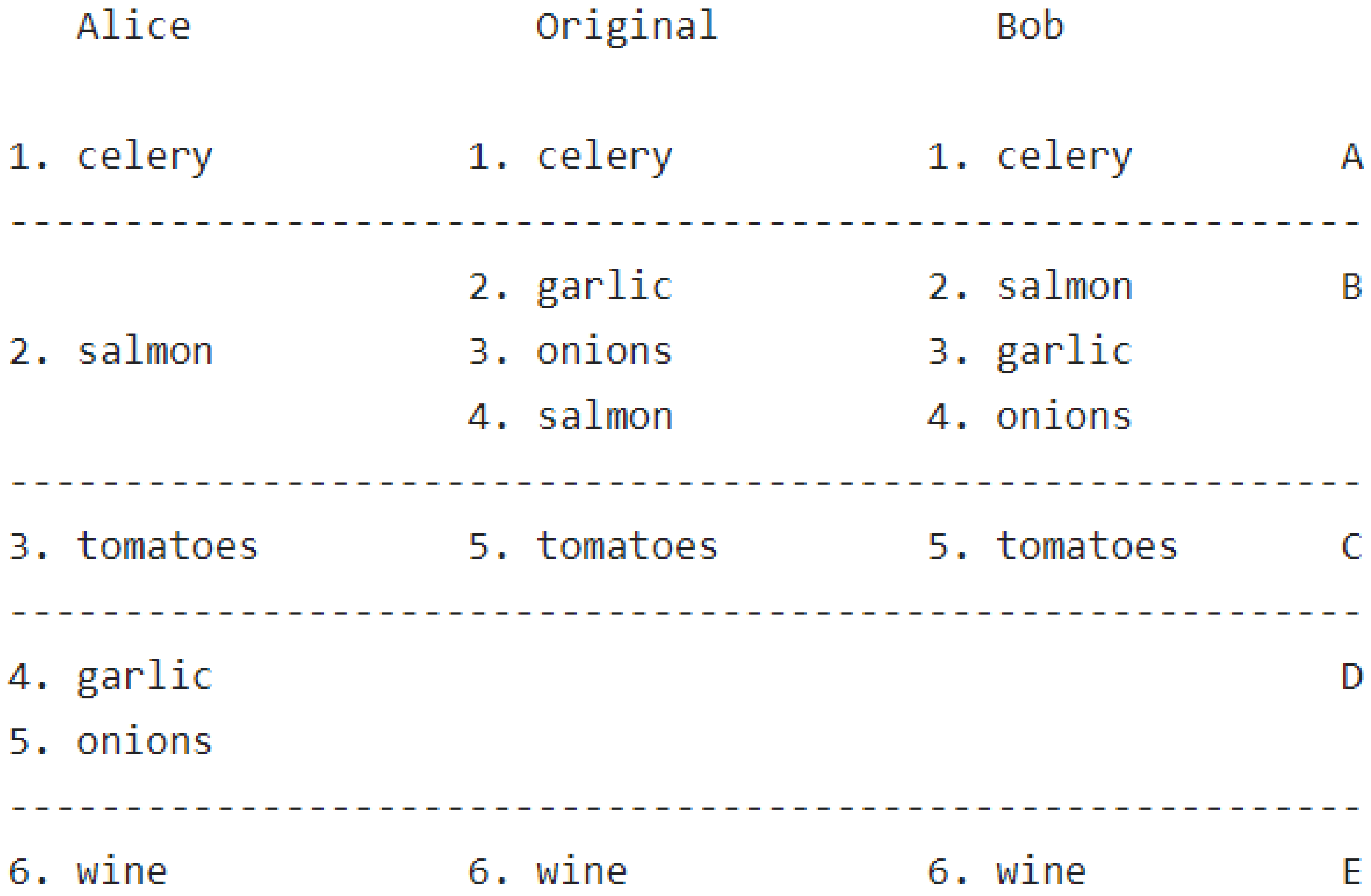


```
        Alice           Original

        1. celery       1. celery
-                       2. garlic
-                       3. onions
        2. salmon       4. salmon
        3. tomatoes     5. tomatoes
+       4. garlic
+       5. onions
        6. wine         6. wine
```



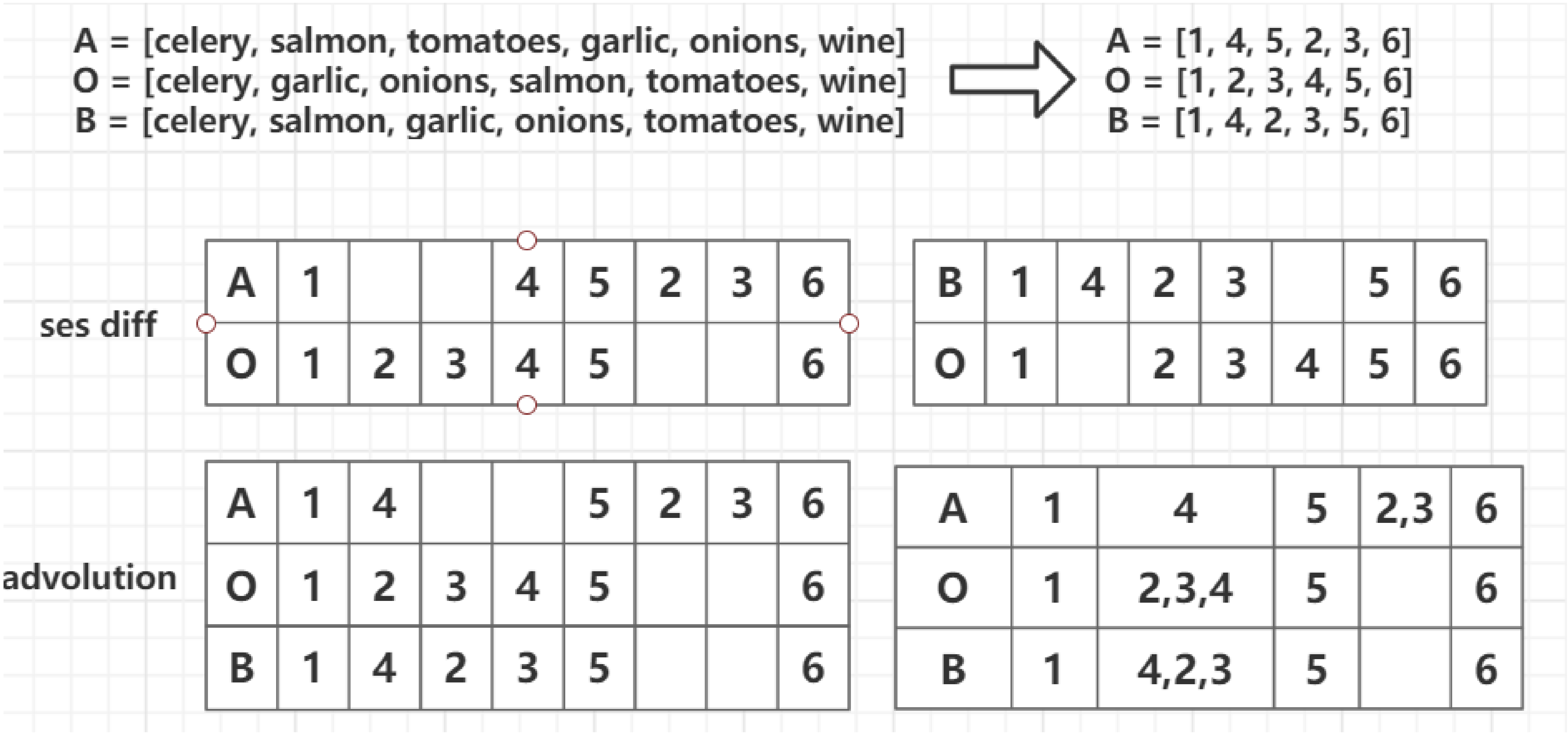
```
        Original        Bob

        1. celery       1. celery
+                       2. salmon
        2. garlic       3. garlic
        3. onions       4. onions
-       4. salmon
        5. tomatoes     5. tomatoes
        6. wine         6. wine
```

Build chunks



```
   Alice               Original            Bob

1. celery           1. celery           1. celery         A
------------------------------------------------------------
                    2. garlic           2. salmon         B
2. salmon           3. onions           3. garlic
                    4. salmon           4. onions
------------------------------------------------------------
3. tomatoes         5. tomatoes         5. tomatoes       C
------------------------------------------------------------
4. garlic                                                 D
5. onions
------------------------------------------------------------
6. wine             6. wine             6. wine           E
```

Ex, code



A = [celery, salmon, tomatoes, garlic, onions, wine]
O = [celery, garlic, onions, salmon, tomatoes, wine]
B = [celery, salmon, garlic, onions, tomatoes, wine]

A = [1, 4, 5, 2, 3, 6]
O = [1, 2, 3, 4, 5, 6]
B = [1, 4, 2, 3, 5, 6]

ses diff

A	1			4	5	2	3	6
O	1	2	3	4	5			6

B	1	4	2	3		5	6
O	1		2	3	4	5	6

advolution

A	1	4			5	2	3	6
O	1	2	3	4	5			6
B	1	4	2	3	5			6

A	1	4	5	2,3	6
O	1	2,3,4	5		6
B	1	4,2,3	5		6

References

- Nikita Thanks for this tutorial.

- Nick Butler this post is short but enough to help you have a higher level to understand diff
- jcoglan this posts make a detail description about the diff.
- visualize if you wanna have a debugger or visualization about diff algorithm, this will be a good choice.

- Real git if you wanna learn the real git, this would be a good introductory article