# A. [Looping through the content of a file in Bash](https://stackoverflow.com/questions/1521462/looping-through-the-content-of-a-file-in-bash)

|  |
| --- |
| The file peptides.txt contains:  RKEKNVQ  IPKKLLQK  QYFHQLEKMNVK  IPKKLLQK  GDLSTALEVAIDCYEK  QYFHQLEKMNVKIPENIYR  RKEKNVQ  VLAKHGKLQDAIN  ILGFMK  LEDVALQILL  One way to do it is:  while read p; do  echo $p  done <peptides.txt  **Option 1a:** While loop: Single line at a time: Input redirection  #!/bin/bash  filename='peptides.txt'  echo Start  while read p; do  echo $p  done < $filename  **Option 1b:** While loop: Single line at a time: Open the file, read from a file descriptor (in this case file descriptor #4).  #!/bin/bash  filename='peptides.txt'  exec 4<$filename  echo Start  while read -u4 p ; do  echo $p  done  **Option 2:** For loop: Read file into single variable and parse. This syntax will parse "lines" based on any white space between the tokens. This still works because the given input file lines are single work tokens. If there were more than one token per line, then this method would not work as well. Also, reading the full file into a single variable is not a good strategy for large files.  #!/bin/bash  filename='peptides.txt'  filelines=`cat $filename`  echo Start  for line in $filelines ; do  echo $line  done  or simple in 1 line:  for word in $(cat peptides.txt); do echo $word; done |

# B. Read with advance:

|  |
| --- |
| A few more things not covered by other answers: Reading from a delimited file # ':' is the delimiter here, and there are three fields on each line in the file  # IFS set below is restricted to the context of `read`, it doesn't affect any other code  while IFS=: read -r field1 field2 field3; do  # process the fields  # if the line has less than three fields, the missing fields will be set to an empty string  # if the line has more than three fields, `field3` will get all the values, including the third field plus the delimiter(s)  done < input.txt Reading from the output of another command, using process substitution while read -r line; do  # process the line  done < <(command ...)  This approach is better than command ... | while read -r line; do ... because the while loop here runs in the current shell rather than a subshell as in the case of the latter. See the related post [A variable modified inside a while loop is not remembered](https://stackoverflow.com/questions/16854280/a-variable-modified-inside-a-while-loop-is-not-remembered). Reading from a null delimited input, for example find ... -print0 while read -r -d '' line; do  # logic  # use a second 'read ... <<< "$line"' if we need to tokenize the line  done < <(find /path/to/dir -print0) Reading from more than one file at a time while read -u 3 -r line1 && read -u 4 -r line2; do  # process the lines  # note that the loop will end when we reach EOF on either of the files, because of the `&&`  done 3< input1.txt 4< input2.txt Reading a whole file into an array (Bash versions earlier to 4) while IFS= read -r line; do  my\_array+=("$line")  done < my\_file Reading a whole file into an array (Bash versions 4x and later) readarray -t my\_array < my\_file  or  mapfile -t my\_array < my\_file  And then  for line in "${my\_array[@]}"; do  # process the lines  done |