# 1.How to do it wrongly

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| First, let’s go through some examples that are *wrong*, because the first step to fixing things is to know what’s broken.  These examples assume default settings (e.g., there is no “set -f” or “IFS=...”):   |  |  |  | | --- | --- | --- | | |  | | --- | | **CASE1**  **#WRONG**  [root@TESTBED-VOD-CMS dath]# cat -nap.txt  cat: invalid option -- 'a'  Try `cat --help' for more information. |   This is wrong. If a filename in the current directory begins with “-”, it will be misinterpreted as an option instead of as a filename.  For example, **if there’s a file named “-n”**, it will suddenly enable cat’s “-n” option instead if it has one (GNU cat does, it numbers the lines).   * In general you should **never** have a glob that begins with “\*” — it should be prefixed with “**./**”.  |  | | --- | | * **WRIGHT**:   [root@TESTBED-VOD-CMS dath]# cat ./-nap.txt  test | |  |  |  |  | | --- | --- | --- | | **CASE 1.1**   |  | | --- | | **WRONG**:  #!/bin/bash  cd /home/dath/cat  for file in \*.txt; do  cat "$file"  done  [root@TESTBED-VOD-CMS cat]# ./catLoop.sh  cat: invalid option -- 'a'  Try `cat --help' for more information |   Also wrong, for the same reason; a file named “-n” will fool the cat program, and if the pattern does not match, it will loop once with the pattern itself as the value.   |  | | --- | | WRIGHT:  #!/bin/bash  cd /home/dath/cat  for file in ./\*.txt; do  cat "$file"  done  [root@TESTBED-VOD-CMS cat]# ./catLoop.sh  test | | |

|  |  |  |
| --- | --- | --- |
| CASE 2   |  | | --- | | **WRONG**  [root@TESTBED-VOD-CMS dath]# cat result .txt  cat: result: No such file or directory  cat: .txt: No such file or directory |   Wrong. If $file can contain whitespace, then it could broken up and interpreted as multiple file names   |  | | --- | | **WRIGHT**  dath]# cat result\ .txt  /NAS\_INGEST01/cms/repository/asset/201711/976/976\_B.png 309  /NAS\_INGEST01/cms/repository/asset/201711/977/977\_B.png 309 | |

* **Conclude**: If $file can contain **whitespace**, then it could broken up and interpreted as multiple file names, and if $file starts with **dash**, then the name will be interpreted as an option. Also, if $file contains **metacharacters** like “\*” they will be expanded first, producing the wrong set of filenames.

# 2.Doing it correctly: A quick summary

So, how can you process pathnames *correctly* in shell? Here’s a quick summary about how to do it correctly, for the impatient who “just want the answer”.

## 2.1. Basic Rule

|  |
| --- |
| 1. [Double-quote all variable references and command substitutions](https://www.dwheeler.com/essays/filenames-in-shell.html#doublequote) unless you are *certain* they can only contain alphanumeric characters or you have specially prepared things (i.e., use "$variable" instead of $variable). In particular, you should practically always put $@ inside double-quotes; POSIX defines this to be special (it expands into the positional parameters as*separate* fields even though it is inside double-quotes). 2. [Set IFS to just newline and tab](https://www.dwheeler.com/essays/filenames-in-shell.html#ifs), if you can, to reduce the risk of mishandling filenames with spaces. Use newline or tab to separate options stored in a single variable. Set IFS withIFS="$(printf '\n\t')" 3. [Prefix all pathname globs so they cannot expand to begin with “-”](https://www.dwheeler.com/essays/filenames-in-shell.html#prefixglobs). In particular, never start a glob with “?” or “\*” (such as “\*.pdf”); always prepend globs with something (like “./”) that cannot expand to a dash. So never use a pattern like “\*.pdf”; use “./\*.pdf” instead. 4. [Check if a pathname begins with “-” when accepting pathnames](https://www.dwheeler.com/essays/filenames-in-shell.html#checkdash), and then prepend “./” if it does. 5. [Be careful about displaying or storing pathnames](https://www.dwheeler.com/essays/filenames-in-shell.html#display-store), since they can include newlines, tabs, terminal control escape sequences, non-UTF-8 characters (or characters not in your locale), and so on. You can strip out control characters and non-UTF-8 characters before display using printf '%s' "$file" | LC\_ALL=POSIX tr -d '[:cntrl:]' | iconv -cs -f UTF-8 -t UTF-8 6. [Do *not* depend on always using “--”](https://www.dwheeler.com/essays/filenames-in-shell.html#dashdash) between options and pathnames as the primary countermeasure against filenames beginning with “-”. You have to do it with *every* command for this to work, but people will *not* use it consistently (they never have), and many programs (including echo) do not support “--”. Feel free to use “--” between options and pathnames, but only as an *additional* optional protective measure. 7. Use a template that is known to work correctly; below are some [tested](https://www.dwheeler.com/encodef/evil-filenames-test) templates. 8. Use a tool like [shellcheck](http://www.shellcheck.net/) to find problems you missed. |

## 2.2. Template Using Globs

|  |  |
| --- | --- |
| |  | | --- | | # **Correct portable glob use: use "for" loop, prefix glob, check for existence**:  # (remember that globs normally do NOT include files beginning with "."):  for file in ./\* ; do # **Prefix with "./\*", NEVER begin with bare "\*"**  if [[ -e "$file" ]] ; then # **Make sure it isn't an empty match**  COMMAND ... "$file" ...  fi  done  ------------------------------------------------ | |

|  |
| --- |
| # **Correct portable glob use, including hidden files (beginning with ".")**:  for file in ./\* ./.[!.]\* ./..?\* ; do # **Prefix with "./\*"**  if [ -e "$file" ] ; then # **Make sure it isn't an empty match**  COMMAND ... "$file" ...  fi  done |

|  |
| --- |
| # **Correct glob use, simpler but requires nonstandard bash extension nullglob**:  shopt -s nullglob # **Bash extension, so globs with no matches return empty**  for file in ./\* ; do # **Use "./\*", NEVER bare "\*"**  COMMAND ... "$file" ...  done |

|  |
| --- |
| # **Correct glob use, simpler but requires nonstandard bash extension nullglob;**  # you can do things on one line if you can add /dev/null as an input.  shopt -s nullglob # **Bash extension, so globs with no matches return empty**  COMMAND ... ./\* /dev/null |

## 2.3. Template Using find

|  |
| --- |
| The find command is great for recursively processing directories.  Typically you would specify other parameters to find (e.g., select only normal files using “-type f”).  For example, here's an example of using find to walk the filesystem, skipping all "hidden" directories and files (names beginning with ".") and processing only files ending in .c or .h:  find . \( -path '\*/.\*' -prune -o ! -name '.\*' \) -a -name '\*.[ch]' |

### 2.3.1. Always works

|  |
| --- |
| # Simple find -exec; unwieldy if COMMAND is large, and creates 1 process/file:  find . -exec COMMAND... {} \;  # Simple find -exec with +, faster if multiple files are okay for COMMAND:  find . -exec COMMAND... {} \+  # Use find and xargs with \0 separators  # (nonstandard common extensions -print0 and -0. Works on GNU, \*BSDs, busybox)  find . -print0 | xargs -0 COMMAND |

|  |
| --- |
| # Head-busting, but it works portably. Use '\'' for single-quote in command.  # Runs a subshell, so variable values are lost after each iteration:  find . -exec sh -c '  for file do  ... # Use "$file" not $file  done' sh {} + |

|  |
| --- |
| # find... while loop, requires find (-print0) and shell (read -d) extensions.  # Fails on Cygwin; in while loops, filenames ending in \r \n and \n look =.  # Variable values may be lost unset because loop may run in a subshell.  find . -print0 | while IFS="" read -r -d "" file ; do ...  COMMAND "$file" # Use quoted "$file", not $file, everywhere.  done |

<https://www.dwheeler.com/essays/filenames-in-shell.html#prefixglobs>