

Moving Around the Command Line

| | |
|-------------------|----|
| Forward one word | %f |
| Back one word | %b |
| Beginning of line | ^a |
| End of line | ^e |

Editing the Command Line

| | |
|---|----|
| Delete to end of line | ^k |
| Delete from beginning of line to here | ^u |
| Delete one letter backwards until space | ^w |
| Swap this and prev letter | ^t |
| Swap this and prev word | %t |
| Clear screen (Lower case L) | ^l |

Misc Command Line

| | |
|---|---------------|
| Run command "cmd" in the background | cmd & |
| Suspend the current process | ^z |
| └-Bring that process back | fg |
| └-Continue that process but in the background | bg |
| Search prev commands (type and it'll auto complete) | ^r |
| └-No, not this one, a different one | ^r (again) |

Directing input/output

| | |
|---|-----------------------------------|
| Direct input | fileContent= \$(<filename.txt) |
| Write all output to a file | ls -lah > filename.txt |
| Append output to a file | echo "hello" >> filename.txt |
| Redirect standard output to filename | ls -lah 1>filename |
| Redirect and append standard output to filename | ls -lah 1>>filename |

Directing input/output (cont)

| | |
|--|-----------------------------|
| Redirect stderr to filename | ls -lah 2>filename |
| Redirect and append stderr to filename | ls -lah 2>>filename |
| Redirect both stdout and stderr to filename | ls -lah &>filename |
| Redirect stderr to stdout | command 2>&1 |
| Redirect stdout to stdout | command >&1 |
| Redirect stdout to stderr | command >&2 |
| Send output from one command to input of another | one command another |

Misc

| | |
|---|---|
| Inject .env into your bash session | export \$(cat .env xargs) |
| Prompt the user | read -sp "Prompt" varName |
| Read in command line options / parameters | https://linuxconfig.org/how-to-use-getopts-to-parse-a-script-options-parameters |

Arrays

| | |
|-------------------------------------|---|
| Create | a=("one" "two" "three" "four" "five" "six") |
| └- | declare -a a |
| └- | declare -a a=("one" "two" "three") |
| └- | a[index]="one" |
| └- | a=(\$(echo "\${space_delim_str}")) |
| └-Assign command output to an array | files=(\$(ls)) |

Arrays (cont)

| | |
|---------------------------------------|--|
| Add multiple items | a+=("seven" "eight" "nine") |
| Copy | b=("\${a[@]}") |
| Print | echo "\${a[@]}" |
| Length/Number of elements | "\${#a[@]}" |
| Get an element (Zero indexed) | "\${a[3]}" |
| Slice | "\${a[@]:2:4}" |
| Search (works with regex) and Replace | "\${a[@]/one/zero}" |
| Search and Remove | "\${a[@]/two/}" |
| Delete an element - leaves a hole | unset "\${a[2]}" |
| Delete an element - no hole | pos=3; a=("\${a[@]:0:\$pos}" "\${a[@]:(\$pos+1)}") |
| Delete entire array | unset a |
| Concat | c=("\${a[@]}" "\${b[@]}") |
| Load file content into array | a=(`cat filename.txt`) |
| Loop through array | for item in "\${a[@]}; do ... done |
| └-by index | for index in "\${!a[@]}; do ... done |
| └-use a range instead | for num in {8..45}; do ... done |

Always include the double quotes when dealing with arrays. If you don't, there's a good chance something will break unexpectedly.

If you try to take a slice from indexes that don't exist in the array, you'll either get what *is* there, or nothing if you completely miss it. There will be no error.



Hashes / Associative Arrays

| | |
|---------------|--|
| Create | declare -A a |
| L- | declare -A a=(["ONE"]="one" ["TWO"]="two" ["THREE"]="three") |
| Set a value | a["KEY"]="value" |
| Print a value | echo a[key] |

Requires Bash 4 or higher. Doesn't seem to work in OSX Catalina, even with the right version of Bash. An alternative that's less awful than the <4 bash way is to use two arrays with matching indexes.

```
if [ "${BASH_VERSINFO:-0}" -lt 4 ]; then ... fi
```

Aside from creation, they work just like regular arrays. When you use a key, it doesn't

Parameter Expansion

| | |
|--|----------------------|
| If parameter is unset or null, the expansion of word is substituted. Otherwise, the value of parameter is substituted. | `\${parameter:-word} |
| If parameter is unset or null, the expansion of word is assigned to parameter. The value of parameter is then substituted. Positional parameters and special parameters may not be assigned to in this way. | `\${parameter:=word} |
| If parameter is null or unset, the expansion of word (or a message to that effect if word is not present) is written to the standard error and the shell, if it is not interactive, exits. Otherwise, the value of parameter is substituted. | `\${parameter:?word} |

Parameter Expansion (cont)

If parameter is null or unset, nothing is substituted, otherwise the expansion of word is substituted.

Substring

`\${parameter:offset:length}

Last char in string

`\${parameter:1:1}

Expands to the names of variables whose names begin with prefix, separated by the first character of the IFS special variable. When '@' is used and the expansion appears within double quotes, each variable name expands to a separate word.

If name is an array variable, expands to the list of array indices (keys) assigned in name. If name is not an array, expands to 0 if name is set and null otherwise. When '@' is used and the expansion appears within double quotes, each key expands to a separate word.

Brackets

| | |
|---|---|
| Run commands in a subshell | (ls -la) |
| Create an array | x=("a" "b" "c") |
| Split string on character (space) | IFS=' ' names="mary joe bob" |
| Integer arithmetic (does not return result) | i=0; ((i += 1)) |
| Integer arithmetic (returns result) | i=\$((1 + 1)) |
| Process substitution - pipe the stdout of multiple commands | comm <(ls -l) <(ls -al) |
| Turn subshell command result into string | echo "My name is \$(whoami)" |
| Truthiness check (Use for the -z -x -n type checks) | [-z \$x] |
| True/False testing | [[\$a =~ /s/]] |
| Expansion | mkdir something/{sibling1,sibling2,sibling3} |
| Range | {0..5} {0..8..2} |
| Command grouping | [[\$a =~ /s/]] && { echo "hey!"; echo "newline" } |
| Variables in a string | "Some string \${variable1:-default value}" |
| String manipulation | -Remove from the front, matching the pattern */, non-greedy # => /example.com/wat url=https://-example.com/wat \${url#*/} |

Brackets (cont)

|-Remove from the front, matching the pattern `*`, greedy `# => /wat`

```
url=https://example.com/wat echo ${url##*/}
```

|-Remove from the back, matching the pattern `*`, non-greedy `# => https://example.com`

```
url=https://example.com/wat echo ${url%/*}
```

|-Remove from the back, matching the pattern `*`, greedy `# => https://example.com`

```
url=https://example.com/wat echo ${url%%/*}
```

|-Replace pattern `=> ftp://example.com`

```
url=https://example.com echo ${url/https/ftp}
```

↳Global replace pattern `=> https://XxampIX.com`

```
url=https://example.com echo ${url//[e]/X}
```

Multiline strings/here-docs

```
x=<<EOF ...
many lines ...
EOF
```

Truth checks

True if variable is set or empty (No error if not set)

```
[[ -z ${varName+x} ]]
```

True if variable is NOT set

```
[[ -n $varName ]]
```

True if variable is set (Bash 4.5+)

```
[[ -v $varName ]]
```

True if file exists

```
[[ -f /file/path ]]
```

True if directory exists

```
[[ -d /directory/path ]]
```

True if symbolic link exists

```
[[ -L /symbolic/link/path ]]
```

Files

Truth checks (cont)

|- `-e` Exists

|- `-d` Directory

|- `-f` Non-directory file

|- `-r` Readable file

|- `-w` Writeable file

|- `-x` Executable file

|- `-L` Symbolic link

|- `-S` Socket

↳ `-s` File exists and has nonzero size



By **tanglisha**
cheatography.com/tanglisha/

Not published yet.
 Last updated 29th June, 2021.
 Page 3 of 3.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>