Bash Shortcuts & Commands Reference

History Shortcuts

Shortcut	What it does
!!	Repeat last command
!n	Repeat command number n from history
!-n	Repeat command n commands ago
!string	Repeat last command starting with "string"
!?string	Repeat last command containing "string"
^old^new	Repeat last command, replacing "old" with "new"
!!:p	Print last command without executing
!$	Last argument of previous command
!*	All arguments of previous command
!^	First argument of previous command

Examples:

```bash
$ ls /home/chamath
$ cd !$        # cd /home/chamath (uses last argument)

$ echo hello world
$ echo !*      # echo hello world (uses all arguments)

$ apt search vim
$ sudo !!      # sudo apt search vim (repeat with sudo)
```

Keyboard Shortcuts (Command Line Editing)

Navigation:

Shortcut	Action
Ctrl + A	Go to beginning of line
Ctrl + E	Go to end of line
Alt + F	Move forward one word
Alt + B	Move backward one word

Editing:

Shortcut	Action
Ctrl + U	Delete from cursor to beginning of line
Ctrl + K	Delete from cursor to end of line
Ctrl + W	Delete word before cursor
Alt + D	Delete word after cursor
Ctrl + Y	Paste (yank) what was deleted
Ctrl + L	Clear screen (like clear command)

Process Control:

Shortcut	Action
Ctrl + C	Kill current process (SIGINT)
Ctrl + Z	Suspend current process (SIGTSTP)
Ctrl + D	Exit shell / End of input (EOF)
Ctrl + R	Search command history (reverse search)

Special Variables

Variable	What it contains
$?	Exit status of last command (0 = success)
$$	Current shell's process ID
$!	PID of last background process
$0	Name of current script/shell
$1, $2, ...	Positional parameters (script arguments)
$#	Number of positional parameters
$@	All arguments as separate words
$*	All arguments as single string
$_	Last argument of previous command
$HOME	Your home directory path
$PWD	Current working directory
$USER	Current username
$PATH	Executable search path

Command Substitution

Syntax	What it does
$(command)	Use command output in another command (preferred)
`command`	Old style command substitution (works but less readable)

Examples:

bash

```
$ echo "Today is $(date)"
Today is Wed Oct 8 2025

$ echo "I'm in $(pwd)"
I'm in /home/chamath/docs

$ mkdir backup_$(date +%Y%m%d)  # Creates: backup_20251008

$ files=$(ls *.txt)
$ echo "Text files: $files"
```

Pipes and Redirection

Symbol	What it does
`|`	Pipe output to next command
`>`	Redirect output to file (overwrite)
`>>`	Redirect output to file (append)
`<`	Read input from file
`2>`	Redirect stderr (errors) to file
`&>`	Redirect both stdout and stderr
`2>&1`	Redirect stderr to same place as stdout
`|&`	Pipe both stdout and stderr

Examples:

bash

```
$ ls -la | grep txt              # Find .txt files
$ echo "hello" > file.txt        # Write to file (overwrite)
$ echo "world" >> file.txt       # Append to file
$ cat file.txt | wc -l           # Count lines
$ command 2> errors.log          # Save errors to file
$ command &> output.log          # Save everything to file
$ command 2>&1 | tee output.log  # Show and save everything
```

Background & Job Control

Command	What it does
`command &`	Run command in background
`jobs`	List background jobs
`fg`	Bring most recent job to foreground
`fg %n`	Bring job number n to foreground
`bg`	Resume most recent suspended job in background
`bg %n`	Resume job number n in background
`kill %n`	Kill job number n
`disown`	Remove job from shell's job table

Example:

```bash
$ sleep 100 &          # Run in background
[1] 12345              # Job number and PID

$ jobs                 # See background jobs
[1]+ Running           sleep 100 &

$ fg 1                 # Bring job 1 to foreground
# Press Ctrl+Z to suspend
[1]+ Stopped           sleep 100

$ bg 1                 # Resume in background
[1]+ Running           sleep 100 &
```

Wildcards & Pattern Matching

Pattern	Matches
`*`	Zero or more characters
`?`	Exactly one character
`[abc]`	Any one of: a, b, or c
`[a-z]`	Any character in range a to z
`[!abc]`	Any character except a, b, or c
`{a,b,c}`	Brace expansion: expands to a, b, and c

Examples:

```bash
$ ls *.txt             # All files ending in .txt
$ ls file?.txt         # file1.txt, fileA.txt, etc.
$ ls [abc]*.txt        # Files starting with a, b, or c
$ rm file{1,2,3}.txt   # Remove file1.txt, file2.txt, file3.txt
$ cp file.txt{,.bak}   # Copy file.txt to file.txt.bak
$ mkdir dir_{a,b,c}    # Create dir_a, dir_b, dir_c
```

Command Chaining

Operator	Behavior
;	Run commands sequentially (regardless of success)
&&	Run next command only if previous succeeded
||	Run next command only if previous failed
&	Run command in background

Examples:

bash

```
# Sequential execution
$ cd /tmp; ls; pwd

# Conditional execution (AND)
$ mkdir test && cd test          # cd only if mkdir succeeds

# Conditional execution (OR)
$ cd /nonexistent || echo "Failed"   # echo only if cd fails

# Combined
$ make && make test && make install  # Stop on first failure

# Background
$ long_process & other_command       # Run both simultaneously
```

Useful Command Combinations

bash

```bash
# Find and delete files
$ find . -name "*.tmp" -delete

# Search command history
$ history | grep ssh

# Count files in directory
$ ls -1 | wc -l

# Show largest files/directories
$ du -sh * | sort -rh | head -10

# Repeat command until it succeeds
$ while ! ping -c1 google.com; do sleep 1; done

# Watch command output (repeat every 2 seconds)
$ watch -n 2 'df -h'

# Quick backup
$ cp file.txt{,.bak}    # Expands to: cp file.txt file.txt.bak

# Multiple commands based on success/failure
$ command1 && echo "Success" || echo "Failed"
```

Advanced History Tricks

```bash
# Repeat command #42 from history
$ !42

# Repeat last command starting with "git"
$ !git

# Replace text in last command
$ cat /etc/hostsss    # Oops, typo
$ ^sss^s              # Changes to: cat /etc/hosts

# Get second argument from previous command
$ echo one two three four
$ echo !:2            # Prints "two"

# Get arguments 2-3 from previous command
$ echo one two three four
$ echo !:2-3          # Prints "two three"

# Get all but first argument
$ rm -rf /path/to/file
$ ls !*               # Uses: -rf /path/to/file
```

Ctrl+R (Reverse Search) Usage

1. Press Ctrl + R
2. Start typing part of a command
3. Press Ctrl + R again to find older matches
4. Press Enter to execute
5. Press → (right arrow) to edit before executing
6. Press Ctrl + C to cancel

Example:

```bash
$ # Press Ctrl+R
(reverse-i-search)`ssh': ssh user@server.com
# Press Ctrl+R again to find older ssh commands
# Press Enter to run, or → to edit
```

Differences: `!*` vs `!$` vs `$_`

Shortcut	Gets	Example
`!*`	All arguments from previous command	`echo a b c` → `!*` = `a b c`
`!$`	Last argument only	`echo a b c` → `!$` = `c`
`$_`	Last argument (variable form)	`echo a b c` → `$_` = `c`

When to use each:

```bash
# Use !* to reuse all arguments
$ touch file1.txt file2.txt file3.txt
$ rm !*    # Removes all three files

# Use !$ or $_ to reuse just the last argument
$ mkdir /home/user/documents
$ cd !$    # Changes to /home/user/documents
$ cd $_    # Same effect
```

Tips & Best Practices

1. **Use Tab completion** - Press Tab to autocomplete filenames and commands
2. **Use `Ctrl+R`** - Much faster than scrolling through history with arrow keys
3. **Use `!!` with sudo** - `sudo !!` is extremely common for re-running with elevated privileges
4. **Use `$_` for file operations** - Create then immediately navigate: `mkdir dir && cd $_`
5. **Combine `&&` and `||`** - Create smart command chains: `make && echo "OK" || echo "Failed"`
6. **Use `{,}` for quick copies** - `cp file{,.bak}` is faster than typing filename twice
7. **Check exit status** - After important commands, check `echo $?` to verify success

Quick Reference Card

Most commonly used shortcuts:

- `!!` - Repeat last command
- `sudo !!` - Repeat last command with sudo
- `!$` - Last argument of previous command
- `Ctrl+R` - Search history
- `Ctrl+A` - Go to start of line
- `Ctrl+E` - Go to end of line
- `Ctrl+U` - Clear line before cursor
- `Ctrl+L` - Clear screen
- `Ctrl+C` - Cancel current command
- `command &` - Run in background
- `cmd1 && cmd2` - Run cmd2 only if cmd1 succeeds