README.md

Gum

release v0.9.0 GO reference build passing

A tool for glamorous shell scripts. Leverage the power of Bubbles and Lip Gloss in your scripts and aliases without writing any Go code!

```
> .
```

The above example is running from a single shell script (source).

Tutorial

Gum provides highly configurable, ready-to-use utilities to help you write useful shell scripts and dotfiles aliases with just a few lines of code.

Let's build a simple script to help you write Conventional Commits for your dotfiles.

Start with a `#!/bin/sh` .

```
#!/bin/sh
```

Ask for the commit type with `gum choose` :

```
gum choose "fix" "feat" "docs" "style" "refactor" "test" "chore" "revert"
```

> Tip: this command itself will print to `stdout` which is not all that useful. To make use of the command later on you can save the stdout to a `$VARIABLE` or `file.txt` .

Prompt for an (optional) scope for the commit:

```
gum input --placeholder "scope"
```

Prompt for a commit message:

```
gum input --placeholder "Summary of this change"
```

Prompt for a detailed (multi-line) explanation of the changes:

```
gum write --placeholder "Details of this change (CTRL+D to finish)"
```

Prompt for a confirmation before committing:

> `gum confirm` exits with status `0` if confirmed and status `1` if cancelled.

```
gum confirm "Commit changes?" && git commit -m "$SUMMARY" -m "$DESCRIPTION"
```

Putting it all together...

```
#!/bin/sh
TYPE=$(gum choose "fix" "feat" "docs" "style" "refactor" "test" "chore" "revert")
SCOPE=$(gum input --placeholder "scope")

# Since the scope is optional, wrap it in parentheses if it has a value.
test -n "$SCOPE" && SCOPE="($SCOPE)"

# Pre-populate the input with the type(scope): so that the user may change it
SUMMARY=$(gum input --value "$TYPE$SCOPE: " --placeholder "Summary of this change")
DESCRIPTION=$(gum write --placeholder "Details of this change (CTRL+D to finish)")

# Commit these changes
gum confirm "Commit changes?" && git commit -m "$SUMMARY" -m "$DESCRIPTION"
```

```
> 
```

Installation

Use a package manager:

```
# macOS or Linux
brew install gum

# Arch Linux (btw)
pacman -S gum

# Nix
nix-env -iA nixpkgs.gum
# Or, with flakes
nix run "github:charmbracelet/gum" -- --help
```

```
# Debian/Ubuntu
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://repo.charm.sh/apt/gpg.key | sudo gpg --dearmor -o /etc/apt/keyrings/charm.g
echo "deb [signed-by=/etc/apt/keyrings/charm.gpg] https://repo.charm.sh/apt/ * *" | sudo tee /
sudo apt update && sudo apt install gum

# Fedora/RHEL
echo '[charm]
name=Charm
baseurl=https://repo.charm.sh/yum/
enabled=1
gpgcheck=1
gpgkey=https://repo.charm.sh/yum/gpg.key' | sudo tee /etc/yum.repos.d/charm.repo
sudo yum install gum

# Alpine
apk add gum

# Android (via termux)
pkg install gum

# Windows (via Scoop)
scoop install charm-gum
```

Or download it:

- Packages are available in Debian, RPM, and Alpine formats
- Binaries are available for Linux, macOS, Windows, FreeBSD, OpenBSD, and NetBSD

Or just install it with `go` :

```
go install github.com/charmbracelet/gum@latest
```

Customization

`gum` is designed to be embedded in scripts and supports all sorts of use cases. Components are configurable and customizable to fit your theme and use case.

You can customize with `--flags` . See `gum <command> --help` for a full view of each command's customization and configuration options.

For example, let's use an `input` and change the cursor color, prompt color, prompt indicator, placeholder text, width, and pre-populate the value:

```
gum input --cursor.foreground "#FF0" --prompt.foreground "#0FF" --prompt "* " \
    --placeholder "What's up?" --width 80 --value "Not much, hby?"
```

You can also use `ENVIRONMENT_VARIABLES` to customize `gum` by default, this is useful to keep a consistent theme for all your `gum` commands.

```
export GUM_INPUT_CURSOR_FOREGROUND="#FF0"
export GUM_INPUT_PROMPT_FOREGROUND="#0FF"
export GUM_INPUT_PLACEHOLDER="What's up?"
export GUM_INPUT_PROMPT="* "
export GUM_INPUT_WIDTH=80

# Uses values configured through environment variables above but can still be
# overridden with flags.
gum input
```

```
> g
```

Interaction

Input

Prompt for input with a simple command.

```
gum input > answer.txt
```

Prompt for sensitive input with the `--password` flag.

```
gum input --password > password.txt
```

```
> g
```

Write

Prompt for some multi-line text.

Note: `CTRL+D` and `esc` are used to complete text entry. `CTRL+C` will cancel.

```
gum write > story.txt
```

```
> g
```

Filter

Use fuzzy matching to filter a list of values:

```
echo Strawberry >> flavors.txt
echo Banana >> flavors.txt
echo Cherry >> flavors.txt
cat flavors.txt | gum filter > selection.txt
```

```
> c
```

You can also select multiple items with the `--limit` flag, which determines the maximum number of items that can be chosen.

```
cat flavors.txt | gum filter --limit 2
```

Or, allow any number of selections with the `--no-limit` flag.

```
cat flavors.txt | gum filter --no-limit
```

Choose

Choose an option from a list of choices.

```
echo "Pick a card, any card..."
CARD=$(gum choose --height 15 {{A,K,Q,J},{10..2}}" "{♠,♥,♣,♦})
echo "Was your card the $CARD?"
```

You can also select multiple items with the `--limit` flag, which determines the maximum of items that can be chosen.

```
echo "Pick your top 5 songs."
cat songs.txt | gum choose --limit 5
```

Or, allow any number of selections with the `--no-limit` flag.

```
echo "What do you need from the grocery store?"
cat foods.txt | gum choose --no-limit
```

```
> g
```

Confirm

Confirm whether to perform an action. Exits with code `0` (affirmative) or `1` (negative) depending on selection.

```
gum confirm && rm file.txt || echo "File not removed"
```

```
> 
```

File

Prompt the user to select a file from the file tree.

```
EDITOR $(gum file $HOME)
```

```
> g
```

Pager

Scroll through a long document with line numbers and a fully customizable viewport.

```
gum pager < README.md
```

```
> g
```

Spin

Display a spinner while running a script or command. The spinner will automatically stop after the given command exits.

```
gum spin --spinner dot --title "Buying Bubble Gum..." -- sleep 5
```

```
>
```

Available spinner types include: `line`, `dot`, `minidot`, `jump`, `pulse`, `points`, `globe`, `moon`, `monkey`, `meter`, `hamburger`.

Table

Select a row from some tabular data.

```
gum table < flavors.csv | cut -d ',' -f 1
```

```
> g
```

Styling

Style

Pretty print any string with any layout with one command.

```
gum style \
	--foreground 212 --border-foreground 212 --border double \
	--align center --width 50 --margin "1 2" --padding "2 4" \
	'Bubble Gum (1¢)' 'So sweet and so fresh!'
```

```
>
```

Layout

Join

Combine text vertically or horizontally. Use this command with `gum style` to build layouts and pretty output.

Tip: Always wrap the output of `gum style` in quotes to preserve newlines (`\n`) when using it as an argument in the `join` command.

```
I=$(gum style --padding "1 5" --border double --border-foreground 212 "I")
LOVE=$(gum style --padding "1 4" --border double --border-foreground 57 "LOVE")
BUBBLE=$(gum style --padding "1 8" --border double --border-foreground 255 "Bubble")
GUM=$(gum style --padding "1 5" --border double --border-foreground 240 "Gum")

I_LOVE=$(gum join "$I" "$LOVE")
BUBBLE_GUM=$(gum join "$BUBBLE" "$GUM")
gum join --align center --vertical "$I_LOVE" "$BUBBLE_GUM"
```

Format

`format` processes and formats bodies of text. `gum format` can parse markdown, template strings, and named emojis.

```
# Format some markdown
gum format -- "# Gum Formats" "- Markdown" "- Code" "- Template" "- Emoji"
echo "# Gum Formats\n- Markdown\n- Code\n- Template\n- Emoji" | gum format

# Syntax highlight some code
cat main.go | gum format -t code

# Render text any way you want with templates
echo '{{ Bold "Tasty" }} {{ Italic "Bubble" }} {{ Color "99" "0" " Gum " }}' \
    | gum format -t template

# Display your favorite emojis!
echo 'I :heart: Bubble Gum :candy:' | gum format -t emoji
```

For more information on template helpers, see the Termenv docs. For a full list of named emojis see the GitHub API.

```
> g
```

Examples

See the examples directory for more real world use cases.

How to use `gum` in your daily workflows:

Write a commit message

Prompt for input to write git commit messages with a short summary and longer details with `gum input` and `gum write`.

Bonus points: use `gum filter` with the Conventional Commits Specification as a prefix for your commit message.

```
git commit -m "$(gum input --width 50 --placeholder "Summary of changes")" \
           -m "$(gum write --width 80 --placeholder "Details of changes (CTRL+D to finish)")"
```

Open files in your `$EDITOR`

By default, `gum filter` will display a list of all files (searched recursively) through your current directory, with some sensible ignore settings (`.git` , `node_modules`). You can use this command to easily to pick a file and open it in your `$EDITOR` .

```
$EDITOR $(gum filter)
```

Connect to a TMUX session

Pick from a running `tmux` session and attach to it. Or, if you're already in a `tmux` session, switch sessions.

```
SESSION=$(tmux list-sessions -F \#S | gum filter --placeholder "Pick session...")
tmux switch-client -t $SESSION || tmux attach -t $SESSION
```

```
> 
```

Pick commit hash from your Git history

Filter through your git history searching for commit messages, copying the commit hash of the commit you select.

```
git log --oneline | gum filter | cut -d' ' -f1 # | copy
```

```
> c
```

Skate Passwords

Build a simple (encrypted) password selector with Skate.

Save all your passwords to Skate with `skate set github@pass.db PASSWORD` , etc...

```
skate list -k | gum filter | xargs skate get
```

```
> s
```

Choose packages to uninstall

List all packages installed by your package manager (we'll use `brew`) and choose which packages to uninstall.

```
brew list | gum choose --no-limit | xargs brew uninstall
```

Choose branches to delete

List all branches and choose which branches to delete.

```
git branch | cut -c 3- | gum choose --no-limit | xargs git branch -D
```

Choose pull request to checkout

List all PRs for the current GitHub repository and checkout the chosen PR (using `gh`).

```
gh pr list | cut -f1,2 | gum choose | cut -f1 | xargs gh pr checkout
```

Pick command from shell history

Pick a previously executed command from your shell history to execute, copy, edit, etc...

```
gum filter < $HISTFILE --height 20
```

Sudo password input

See visual feedback when entering password with masked characters with `gum input --password`.

```
alias please="gum input --password | sudo -nS"
```

Feedback

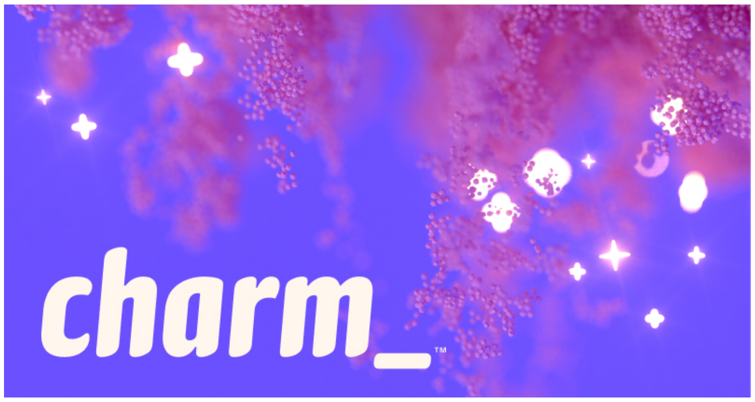
We’d love to hear your thoughts on this project. Feel free to drop us a note!

- Twitter
- The Fediverse
- Discord

License

MIT

Part of Charm.



Charm • Charm loves open source

Releases 9

v0.9.0 Latest
on Jan 12

+ 8 releases

Packages 1

gum

Used by 3

@KloudLite / **kl**

@SuperDynamix / **CLI-gite**

@areThereAnyUserNamesLeft / **typereader**

Contributors 38

+ 27 contributors

Languages

● **Go** 99.3% ● **Other** 0.7%