

cs-send

Handbuch

Mail (SMTP/TLS) und Chat-Benachrichtigungen (Discord, Slack, Telegram, ntfy, Gotify) von der Kommandozeile - ein statisches Binary

Version 0.3.0 · 13. Oktober 2026

Inhalt

1. Überblick	3
2. Installation	3
3. Mail senden	3
3.1 Der TLS-Modus folgt dem Port	3
3.2 Gmail, Outlook und andere Provider	3
3.3 Eingebettete Bilder und Anhänge	4
3.4 Logdatei	4
4. Chat-Benachrichtigungen senden	4
4.1 Nicht enthalten	4
5. Einsatz in napp-it CS	5
6. Aus dem Quellcode bauen	5
7. Tests	5
8. Lizenz und Gewährleistung	5

1. Überblick

cs-send verschickt E-Mails (SMTP über TLS, HTML-Text mit eingebetteten Bildern und normalen Anhängen) und Chat-Benachrichtigungen (Discord, Telegram, Slack, ntfy, Gotify) von der Kommandozeile. Es ist ein einzelnes statisches Binary, das nur die Go-Standardbibliothek nutzt - es gibt keinerlei externe Abhängigkeiten.

cs-send gehört zur napp-it CS Tool-Familie und ist dort fest in die Report-Job-Infrastruktur eingebunden, um Status- und Fehlermeldungen zu verschicken. Es ist aber auch eigenständig nutzbar, zum Beispiel aus cron oder aus eigenen Skripten.

KISS: ein Binary, zwei Unterbefehle (**mail** und **chat**), keine Konfigurationsdatei. Alles wird per Kommandozeilen-Option angegeben.

2. Installation

Fertige Binaries für alle Plattformen hängen an jedem GitHub-Release (**cs-send-*<os>.<arch>.tar.gz***):

Plattform	Archiv
Windows	mswin.amd64
Linux	linux.amd64, linux.arm64
illumos / OmniOS	illumos.amd64
Solaris	solaris.amd64
FreeBSD	freebsd.amd64
macOS	darwin.amd64, darwin.arm64

Entpacken und starten - es gibt keinen Installationsschritt und keine Laufzeit-Abhängigkeit außer dem TLS-/CA-Zertifikatspeicher des Betriebssystems.

In napp-it CS wird das Binary über das Menü **About > Download cs-tools** auf die Member verteilt; es wird nicht automatisch heruntergeladen.

3. Mail senden

```
cs-send mail \  
  --to alice@example.com,bob@example.com \  
  --subject "Backup finished" \  
  --html-file report.html \  
  --attach chart.png \  
  --smtp-host smtp.gmail.com --smtp-port 587 \  
  --user you@gmail.com --key <app-password>
```

3.1 Der TLS-Modus folgt dem Port

Der TLS-Modus ergibt sich aus dem Port, es gibt keine eigene Option: Port **465** bedeutet implizites TLS ab dem ersten Byte (der "SSL"-Port der meisten Provider). Bei jedem anderen Port wird zuerst unverschlüsselt verbunden und dann STARTTLS versucht; **587** ist der fast überall gebräuchliche STARTTLS-Submission-Port.

3.2 Gmail, Outlook und andere Provider

Den "Zugriff weniger sicherer Apps" gibt es bei Google nicht mehr. Man braucht ein Google-Konto mit aktivierter 2-Faktor-Anmeldung und ein dafür erzeugtes App-Passwort (<https://myaccount.google.com/apppasswords>). Dieses App-Passwort wird als **--key** angegeben, nicht das normale Kontopasswort. Outlook/Office365 und andere SMTP-Relays funktionieren genauso, mit dem jeweiligen Passwort- oder App-Passwort-Verfahren des Providers.

3.3 Eingebettete Bilder und Anhänge

Eingebettete Bilder (inline) stehen direkt im HTML-Text (``) und sind kein herunterladbarer Anhang:

```
cs-send mail --to a@b.com --subject "Status" \
--html '<html><body><h1>All good</h1></body></html>' \
--inline chart.png=chart \
--smtp-host smtp.example.com --smtp-port 587 --user u --key p
```

`--inline pfad=cid` kann wiederholt oder kommagetrennt angegeben werden (`--inline a.png=img1,b.png=img2`). `--attach pfad` fügt normale, herunterladbare Anhänge auf dieselbe Weise hinzu. Der Aufruf `cs-send mail` ohne weitere Argumente zeigt die vollständige Optionsliste.

3.4 Logdatei

`--logfile <pfad>` (bei `mail` und `chat`) hängt pro Aufruf eine Ergebniszeile mit Zeitstempel an. Das ist nützlich, wenn `cs-send` aus cron oder aus einem napp-it CS Job gestartet wird, wo stdout und stderr nicht aufbewahrt werden.

Das Log enthält nie Zugangsdaten: SMTP-Passwörter (`--key`), Webhook-URLs und Bot-Tokens werden unkenntlich gemacht. Eine Webhook-URL erlaubt jedem, der sie kennt, das Senden - sie wird deshalb wie ein Passwort behandelt.

4. Chat-Benachrichtigungen senden

Eine Chat-Benachrichtigung ist je Provider ein ausgehender HTTPS-Request - kein OAuth, kein App-Review, nur eine Webhook-URL oder ein Bot-Token, das man schon hat.

```
cs-send chat --provider discord --url <webhook-url> --text "Deploy finished"
cs-send chat --provider slack --url <webhook-url> --text "Deploy finished"
cs-send chat --provider ntfy --url https://ntfy.sh/my-topic --text "Disk 90% full" --priority 5
cs-send chat --provider gotify --url https://gotify.example.com --token <app-token> --text "Scrub done"
cs-send chat --provider telegram --token <bot-token> --chat-id <id> --text "Backup failed"
```

Provider	Was man braucht	Einrichtung
Discord	eine Webhook-URL	Servereinstellungen > Integrationen > Webhooks > Neuer Webhook > URL kopieren
Slack	eine Webhook-URL	klassischer Incoming Webhook (funktioniert für bestehende Workspaces weiter)
ntfy	eine Topic-URL	beliebigen Topic-Namen auf ntfy.sh wählen oder selbst hosten; für ein öffentliches Topic ist kein Konto nötig
Gotify	eine Server-URL und ein App-Token	selbst gehostet; in der Gotify-Weboberfläche eine App anlegen, um das Token zu erhalten
Telegram	ein Bot-Token und eine Chat-ID	@BotFather anschreiben, <code>/newbot</code> , Token kopieren; über @userinfobot die eigene Chat-ID erfahren

`--title` und `--priority` werden von ntfy und Gotify ausgewertet, die beides kennen. Die anderen drei Provider haben kein eigenes Titelfeld, deshalb wird ein `--title` als fette Zeile vor `--text` gestellt.

4.1 Nicht enthalten

Bewusst (vorerst) nicht enthalten: WhatsApp (die Business-API hängt hinter einer Meta-Verifizierung - für ein Alarm-Tool unverhältnismäßig), Signal (keine offizielle Webhook-API, es wäre ein eigener `signal-cli`-Daemon nötig), Microsoft Teams (klassische Incoming Webhooks werden zugunsten von Power-Automate-Workflows abgeschafft - deutlich mehr Aufwand) und Matrix (machbar mit Access-Token und einem REST-PUT, aber mehr Einrichtung als bei den fünf genannten Providern).

5. Einsatz in napp-it CS

Die Report-Jobs von napp-it CS rufen cs-send auf, um Status- und Fehlermeldungen per Mail oder Chat zu verschicken. SMTP- und Chat-Einstellungen werden in den Job-Menüs eingetragen; cs-send selbst speichert weder Zustand noch Konfiguration.

Für langlebige Zugangsdaten sind Umgebungsvariablen oder ein Wrapper-Skript besser als eine Option direkt auf der Kommandozeile - so landen Geheimnisse nicht in der Shell-History eines gemeinsam genutzten Rechners.

6. Aus dem Quellcode bauen

```
git clone https://github.com/guenther-alka/cs-send.git
cd cs-send
go build -o cs-send .
GOOS=linux GOARCH=amd64 go build -o cs-send .
```

7. Tests

```
go build ./...
go vet ./...
go test ./...
```

Die Unit-Tests decken den MIME-Nachrichtenbau ab (nur Text, Reihenfolge Text plus HTML-Alternative, **Content-ID** bei eingebetteten Bildern, normale Anhänge und die Regel, dass Bcc nie in den Headern steht - geprüft, indem die gebaute Nachricht mit **net/mail** und **mime/multipart** wieder eingelesen wird) sowie jeden Chat-Provider gegen einen lokalen **httptest.Server**, einschließlich des Fehlertexts bei einer Antwort außerhalb von 2xx. Die echte SMTP-Konversation von **mail.Send** ist nicht automatisch getestet (sie braucht einen echten Server); sie wurde manuell gegen Gmail und ein lokales Relay geprüft.

8. Lizenz und Gewährleistung

BSD 2-Clause - siehe Datei LICENSE im Repository.

Es gibt keine Gewährleistung. Empfänger und Anhänge sollten vor dem Versand wichtiger Nachrichten geprüft werden. SMTP-Zugangsdaten und Chat-Tokens sind wie jedes Geheimnis zu behandeln: nicht einchecken und auf gemeinsam genutzten Rechnern möglichst nicht auf der Kommandozeile übergeben.