GPG Cheat Sheet for Fish Terminal on macOS (Rosé Pine)

Quick, secure workflows with Fish shell, Homebrew, and pinentry-mac.

1 GPG Cheat Sheet for Fish terminal

Info

This document includes clickable links and a nested structure mirroring your original Markdown. Headings below are fully navigable in most PDF viewers.

2 GPG Cheat Sheet for Fish Terminal on macOS Silicon

Info

What is GPG? GNU Privacy Guard (GPG) implements the OpenPGP standard for:

1. End-to-end encryption for files and messages
2. Digital signatures to verify authenticity
3. Key management for public/private key pairs
4. Symmetric (password) encryption
5. Integration with email clients, Git, and other tools

Important

macOS Silicon Compatibility
Optimized for Apple Silicon (M1/M2/M3), Fish 3.x, Homebrew, `pinentry-mac`, and Keychain integration.

Warning

Security Fundamentals

1. Never share private keys (only public keys)
2. Use strong passphrases ($\geq$ 20 chars)
3. Rotate keys regularly for high-security needs
4. Verify fingerprints via an independent channel
5. Keep encrypted backups in multiple locations
6. Set expiration dates for periodic reviews
7. Generate revocation certificates in advance

Tip

Fish vs Bash Syntax

Operation	Bash	Fish
Command substitution	`$(command)`	`(command)`
Heredoc alternative	`<<EOF`	`echo/begin...end`
String concatenation	`"$v1$v2"`	`"$v1$v2"` or `{v1}{v2}`
Process substitution	`<(command)`	`(command | psub)`
Export variable	`export VAR=val`	`set -x VAR val`

2.1 Installation & Setup

2.1.1 Install GPG on macOS Silicon

```
# Install via Homebrew (Apple Silicon native)
brew install gnupg

# Verify installation
gpg --version

# Install additional tools
brew install pinentry-mac
```

Info

Breakdown
`gnupg` (ARM64), version check, and `pinentry-mac` for native password prompts. Homebrew handles deps and ships gpg-agent.

2.1.2 Initial Configuration

```
# Create GPG directory if not exists
mkdir -p ~/.gnupg
chmod 700 ~/.gnupg

# Configure GPG for macOS
echo "pinentry-program /opt/homebrew/bin/pinentry-mac" >> ~/.gnupg/gpg-agent.conf

# Restart GPG agent
gpgconf --kill gpg-agent
gpgconf --launch gpg-agent
```

Important

Fish Shell Config (GPG TTY)
Add this to `~/.config/fish/config.fish`:

```
set -x GPG_TTY (tty)
```

Fixes "inappropriate ioctl" errors in Fish.

2.2 Key Management

2.2.1 Generate Keys

```
# Generate a new key pair (interactive)
gpg --full-generate-key

# Quick generate with defaults
gpg --quick-generate-key "Your Name <email@example.com>"

# Generate with specific algorithm
gpg --quick-generate-key "Your Name <email@example.com>" rsa4096 sign,encr
```

Warning

Algorithm Notes
RSA 2048 (min), 3072 (recommended), 4096 (max security). ECC (Curve25519) is a modern alternative with shorter keys.

2.2.2 List & Export Keys

```
# List public keys
gpg --list-keys
gpg -k

# List secret keys
gpg --list-secret-keys
gpg -K

# Export public key
gpg --armor --export your@email.com > public_key.asc

# Export secret key (careful!)
gpg --armor --export-secret-keys your@email.com > private_key.asc

# Export to keyserver
gpg --keyserver hkps://keys.openpgp.org --send-keys YOUR_KEY_ID
```

Danger

Private Key Export
Only for secure backup, store encrypted, never send over email/unencrypted channels, delete after transfer. Consider paperkey.

2.2.3 Import Keys

```
# Import public key
gpg --import public_key.asc

# Import from keyserver
gpg --keyserver hkps://keys.openpgp.org --receive-keys KEY_ID

# Import and trust automatically
echo "FINGERPRINT:6:" | gpg --import-ownertrust
```

Tip

Trust Levels
Unknown (0), Never (1), Marginal (2), Full (3), Ultimate (4: your keys).

2.3 Encryption & Decryption

2.3.1 Encrypting Messages

```
# Encrypt for recipient using echo
echo "Secret message" | gpg --armor --encrypt -r recipient@email.com

# Encrypt multiline content (Fish block)
begin
    echo "Line 1"
    echo "Line 2"
    echo "Line 3"
end | gpg --armor --encrypt -r recipient@email.com

# Using fish variable
set message "This is my secret message"
echo $message | gpg --armor --encrypt -r recipient@email.com > encrypted.asc

# Encrypt file
gpg --armor --encrypt -r recipient@email.com document.pdf

# Select a specific local key for encryption
echo "message" | gpg --armor --encrypt --local-user "leifbruce1996@pm.me"

# (Mislabelled in source) To sign a message in a .txt file:
gpg --armor --encrypt message.txt   # (encrypts; for signing see Section 2.4)
```

Tip

Multiple Recipients

```
echo "Shared secret" | gpg --armor --encrypt \
  -r person1@example.com \
  -r person2@example.com \
  -r person3@example.com
```

2.3.2 Decrypting Messages

```
# Decrypt from inline block
echo "-----BEGIN PGP MESSAGE-----
[encrypted content here]
-----END PGP MESSAGE-----" | gpg --decrypt

# Using printf for multiline
printf "%s\n" \
"-----BEGIN PGP MESSAGE-----" \
"[encrypted content]" \
"-----END PGP MESSAGE-----" | gpg --decrypt

# Decrypt from file
gpg --decrypt message.asc

# Decrypt and save output
gpg --decrypt message.asc > decrypted.txt
```

Warning

Heredoc Alternatives in Fish
Use `echo`, `printf`, `begin...end`, save to a file, or process substitution with `psub`.

2.4 Signing & Verification

2.4.1 Signing Messages

```
# Clear sign (readable signature)
echo "This is my statement" | gpg --clear-sign

# Detached signature
gpg --detach-sign document.pdf

# Sign and encrypt together
echo "Secret signed message" | gpg --armor --sign --encrypt -r recipient@email.com

# Select a key when signing (Fish + local-user)
echo "message" | gpg --clear-sign --local-user "leifbruce1996@pm.me"

# Sign a message in a .txt file
gpg --clear-sign message.txt
```

Tip

Signature Types
Clear-sign (readable), Detached (`.sig`), Inline (binary), Sign+Encrypt (private, authenticated).

2.4.2 Verifying Signatures

```
# Verify a detached signature
gpg --verify signature.sig file.txt

# Verify inline signed message
echo "-----BEGIN PGP SIGNED MESSAGE-----
[message content]
-----BEGIN PGP SIGNATURE-----
[signature]
-----END PGP SIGNATURE-----" | gpg --verify

# Verify and extract signed content
gpg --decrypt signed_message.asc
```

Warning

Trust Warnings
"Good signature" can still be untrusted if you haven't certified the key. Verify fingerprints; adjust trust via `gpg --edit-key EMAIL trust`.

2.5 Symmetric Encryption

```
# Encrypt with password
echo "Secret data" | gpg --armor --symmetric
```

```
# Decrypt symmetric encryption
echo "[encrypted content]" | gpg --decrypt

# Encrypt file with AES256
gpg --cipher-algo AES256 --symmetric document.pdf
```

Info

Algorithms
AES256 (strongest), AES192, AES128, Twofish, Camellia256.
List available: `gpg --version` (Cipher section).

2.6 Fish-Specific Techniques

2.6.1 Process Substitution with `psub`

```
# Using psub for temporary files
gpg --decrypt (echo "encrypted content" | psub)

# Compare two encrypted messages
diff (gpg --decrypt file1.gpg | psub) (gpg --decrypt file2.gpg | psub)
```

2.6.2 Multi-line Strings in Fish

```
set encrypted_msg "-----BEGIN PGP MESSAGE-----
Version: GnuPG v2

hQEMA...
-----END PGP MESSAGE-----"

echo $encrypted_msg | gpg --decrypt
```

2.6.3 Using `begin/end` Blocks

```
begin
  echo "-----BEGIN PGP MESSAGE-----"
  echo "[encrypted content]"
  echo "-----END PGP MESSAGE-----"
end | gpg --decrypt
```

2.7 Advanced Operations

2.7.1 Key Trust & Signing

```
# Sign someone's key
gpg --sign-key their@email.com

# Set trust level interactively
gpg --edit-key their@email.com trust

# List signatures on a key
gpg --list-sigs their@email.com
```

2.7.2 Batch Operations

```
# Encrypt multiple files
for file in *.pdf
  gpg --armor --encrypt -r recipient@email.com $file
end

# Decrypt all .asc files
for file in *.asc
  gpg --decrypt $file > (basename $file .asc).txt
end
```

2.7.3 Key Maintenance

```
# Refresh keys from keyserver
gpg --refresh-keys

# Clean up keyring
gpg --delete-key OLD_KEY_ID
gpg --delete-secret-key OLD_KEY_ID

# Backup GPG directory
tar -czf gpg-backup-(date +%Y%m%d).tar.gz ~/.gnupg

# Change key passphrase
gpg --edit-key YOUR_KEY_ID passwd
```

Danger

Deletion is Permanent
Backup first; deleted secret keys cannot decrypt/sign; consider revocation certs.

2.8 OpenSSL Integration

2.8.1 Quick Crypto

```
# AES encryption with OpenSSL
echo "secret" | openssl enc -aes-256-cbc -a -pbkdf2

# Decrypt
echo "U2FsdGVkX1+..." | openssl enc -aes-256-cbc -d -a -pbkdf2

# Generate random password
openssl rand -base64 32
```

2.8.2 Hashing & Verification

```
# Create hash
echo "message" | sha256sum > message.hash

# Verify hash
echo "message" | sha256sum --check (echo "hash_value  -" | psub)

# Multiple algorithms
echo "data" | openssl dgst -sha512
```

2.9 macOS Silicon Tips

2.9.1 Performance

```
# Ensure native ARM64
file (which gpg)  # should show "arm64"

# Agent caching
echo "default-cache-ttl 3600" >> ~/.gnupg/gpg-agent.conf
echo "max-cache-ttl 86400" >> ~/.gnupg/gpg-agent.conf
```

2.9.2 Keychain Integration

```
brew install pinentry-mac
echo "use-agent" >> ~/.gnupg/gpg.conf

# Test
echo "test" | gpg --sign | gpg --verify
```

2.10 Common Workflows

2.10.1 Secure File Transfer

```
# Sender
gpg --armor --sign --encrypt -r recipient@email.com sensitive.pdf

# Recipient
gpg --decrypt sensitive.pdf.asc > sensitive.pdf
```

2.10.2 Encrypted Backup

```
# Backup
tar -czf - ~/Documents | gpg --symmetric --cipher-algo AES256 > backup.tar.gz.gpg

# Restore
gpg --decrypt backup.tar.gz.gpg | tar -xzf -
```

2.10.3 Git Commit Signing

```
git config --global user.signingkey YOUR_KEY_ID
git config --global commit.gpgsign true

# Sign a specific commit
git commit -S -m "Signed commit"
```

2.11 Troubleshooting

2.11.1 Common Issues (Fish/macOS)

```
# Fix "inappropriate ioctl" error
set -x GPG_TTY (tty)
echo 'set -x GPG_TTY (tty)' >> ~/.config/fish/config.fish

# Reset GPG agent
gpgconf --kill all
```

```
gpgconf --launch gpg-agent

# Fix permissions
chmod 700 ~/.gnupg
chmod 600 ~/.gnupg/*

# Test agent connection
echo "test" | gpg --clearsign
```

Troubleshoot

Common Errors
Inappropriate ioctl: set `GPG_TTY`.
No pinentry: install/configure `pinentry-mac`.
Permission denied: `chmod 700 ~/.gnupg` and `chmod 600 ~/.gnupg/*`.
Agent refused operation: restart agent.

2.11.2 Debug

```
gpg --verbose --decrypt file.asc
gpg --dump-options
gpg --version | grep Home
```

2.12 Quick Reference Card

```
gpg -k                    # List public keys
gpg -K                    # List private keys
gpg --encrypt -r EMAIL    # Encrypt for recipient
gpg --decrypt             # Decrypt message
gpg --sign                # Sign message
gpg --verify              # Verify signature
gpg --clearsign           # Create clear signature
gpg --detach-sign         # Create detached signature
gpg --armor               # ASCII armor output
gpg --refresh-keys        # Update keys from server
```

Tip

Shortcuts
`-k`/`-K` list keys; `-e -r` encrypt; `-d` decrypt; `-s` sign; `-a` armor.

2.13 References

- GnuPG
- OpenPGP
- Homebrew
- Fish Shell Docs
- OpenPGP Keyserver
- Web of Trust
- AES
- SHA-2
- OpenSSL
- Git

3 GPG (GnuPG) Cheat Sheet for macOS

3.1 Installation (Homebrew)

```
brew install gnupg
brew install pinentry-mac

mkdir -p ~/.gnupg
echo "pinentry-program /opt/homebrew/bin/pinentry-mac" >> ~/.gnupg/gpg-agent.conf

# Restart the agent
gpg-connect-agent reloadagent /bye
```

3.2 Key Management Basics

3.2.1 Generate a New Key Pair

```
gpg --full-generate-key
```

Suggested: RSA+RSA, 4096 bits, set expiration (e.g., 1y), strong passphrase.

3.2.2 List Keys

```
gpg --list-keys
gpg --list-secret-keys
```

3.2.3 Export/Share Your Public Key

```
gpg --output your-public-key.gpg --export "your_email@example.com"
gpg --armor --output your-public-key.asc --export "your_email@example.com"
```

3.2.4 Export Private Key (Caution)

```
gpg --list-secret-keys --keyid-format=long
gpg --armor --export-secret-keys YOUR_KEY_ID > private.asc
```

3.2.5 Import Someone Else's Public Key

```
gpg --import someones-public-key.asc
```

3.2.6 Edit a Key

```
gpg --edit-key "your_email@example.com"
```

3.2.7 Delete Keys (Irreversible)

```
gpg --delete-key "user@example.com"
gpg --delete-secret-key "your_email@example.com"
```

3.2.8 Create a Revocation Certificate

```
gpg --output revocation.crt --gen-revoke "your_email@example.com"
```

3.3 On-the-Fly CLI Encryption & Decryption

3.3.1 Encrypt Text

```
echo "This is a top secret message." \
| gpg --encrypt --armor --recipient "recipient_email@example.com"
```

3.3.2 Decrypt Text

```
echo "-----BEGIN PGP MESSAGE-----
... (paste full block) ...
-----END PGP MESSAGE-----" | gpg --decrypt
```

3.4 File Encryption & Decryption

3.4.1 Encrypt a File for a Recipient

```
gpg --encrypt --recipient "recipient_email@example.com" document.txt
```

3.4.2 Symmetric Encryption (Passphrase)

```
gpg --symmetric --armor archive.zip
```

3.4.3 Decrypt a File

```
gpg --decrypt document.txt.gpg
gpg --output original_document.txt --decrypt document.txt.gpg
```

3.5 Digital Signatures

3.5.1 Detached Signature

```
gpg --detach-sig software-release.zip
```

3.5.2 Verify a Detached Signature

```
gpg --verify software-release.zip.sig
```

3.5.3 Clearsign a Text File

```
gpg --clearsign announcement.txt
gpg --verify announcement.txt.asc
```

3.5.4 Sign and Encrypt

```
gpg --sign --encrypt --recipient "recipient_email@example.com" important-contract.docx
```

#GPG #Encryption #macOS #AppleSilicon #FishShell #Cryptography #OpenPGP #Security #Homebrew #KeyManagement