GPG/GnuPG Cheatsheet

Complete Guide for macOS & Fish Terminal

Encryption • Signing • Key Management

Security Reference Guide

October 25, 2025

What is GPG?

GNU Privacy Guard (GPG) is a free implementation of the OpenPGP standard that enables you to encrypt and sign data and communications. GPG provides:

- **End-to-end encryption** for files and messages
- **Digital signatures** to verify authenticity
- **Key management** for public/private key pairs
- **Symmetric encryption** using passwords
- **Integration** with email clients, Git, and other tools

macOS Silicon Compatibility

This guide is specifically optimized for:

- **Apple Silicon** (M1/M2/M3/M4) processors running native ARM64 binaries
- **Fish shell** version 3.x with Fish-specific syntax
- **Homebrew** package manager on macOS
- **Pinentry-mac** for native macOS password prompts
- **Keychain integration** for seamless password management

Security Fundamentals

Critical security practices when using GPG:

1. **Never share private keys** – Only distribute public keys
2. **Use strong passphrases** – Minimum 20 characters with complexity
3. **Regular key rotation** – Generate new keys yearly for high-security needs
4. **Verify fingerprints** – Always verify through separate channels
5. **Backup securely** – Store encrypted backups in multiple locations
6. **Set expiration dates** – Force regular security review
7. **Generate revocation certificates** – Prepare for key compromise scenarios

1 Fish Shell vs Bash Syntax

Key Syntax Differences

When using GPG in Fish shell, note these important syntax differences:

Operation	Bash	Fish
Command substitution	`$(command)`	`(command)`
Heredoc alternative	`« EOF`	`echo` or `begin/end`
String concatenation	`"$var1$var2"`	`"$var1$var2"`
Export variable	`export VAR=value`	`set -x VAR value`

2 Installation & Setup

2.1 Install GPG on macOS Silicon

Install the native ARM64 version of GnuPG optimized for Apple Silicon:

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

# Verify installation
gpg --version

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

Installation Breakdown

- `brew install gnupg` – Installs the native ARM64 version of GnuPG
- `gpg -version` – Verifies successful installation and displays version
- `pinentry-mac` – Provides native macOS dialog boxes for password entry
- Homebrew automatically handles dependencies and ARM64 optimization
- Installation includes GPG agent for password caching

2.2 Initial Configuration

Configure GPG for optimal macOS integration:

```
# 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
```

Configuration Breakdown

- `mkdir -p /.gnupg` – Creates GPG configuration directory
- `chmod 700` – Sets strict permissions (owner-only access)
- `pinentry-program` – Configures native macOS password prompts
- `gpgconf -kill` – Terminates current GPG agent process
- `gpgconf -launch` – Starts fresh GPG agent with new configuration

Fish Shell Configuration

Add this to `/.config/fish/config.fish` for permanent GPG TTY configuration:

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

This fixes common “inappropriate ioctl” errors in Fish shell.

3 Key Management

3.1 Generate Keys

Creating your public/private key pair is the foundation of GPG encryption:

```
# 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
```

Key Generation Options

- `-full-generate-key` – Interactive mode with all options
- `-quick-generate-key` – Fast generation with sensible defaults (RSA 3072-bit)
- `rsa4096` – Specifies 4096-bit RSA keys for maximum security
- `sign,encr` – Creates keys capable of both signing and encryption
- Interactive prompts ask for: name, email, comment, and passphrase
- Key generation uses system entropy for randomness

Key Algorithm Considerations

RSA key size recommendations:

RSA 2048	Minimum for basic security (faster generation)
RSA 3072	Recommended default (good balance)
RSA 4096	Maximum security (slower operations)
ECC (Curve25519)	Modern alternative, shorter keys

Consider computational overhead on mobile devices when choosing key size.

3.2 List & Export Keys

View and share your keys:

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

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

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

# Export secret key (be 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
```

Export Commands

- `-list-keys (-k)` – Displays all public keys in your keyring
- `-list-secret-keys (-K)` – Shows private keys you own
- `-armor` – Outputs in ASCII format instead of binary (.asc extension)
- `-export` – Exports public key for sharing with others
- `-export-secret-keys` – Exports private key (requires passphrase)
- `-send-keys` – Uploads public key to OpenPGP keyserver

Private Key Export Warning

Exporting private keys is **extremely dangerous**:

1. Only export for secure backup purposes
2. Store exported key in encrypted storage
3. Never send private keys via email or unencrypted channels
4. Delete exported file after secure transfer
5. Consider using `paperkey` for physical backup instead

3.3 Import Keys

Add keys to your keyring:

```
# 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
```

Trust Levels

GPG uses a web of trust model with these levels:

Level	Value	Meaning
Unknown	0	Not yet evaluated
Never	1	Key is not trusted
Marginal	2	Some trust
Full	3	Fully trusted
Ultimate	4	Your own keys

4 Encryption & Decryption

4.1 Encrypting Messages

Encrypt data for secure transmission:

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

# Encrypt multiline content (Fish syntax)
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 specific key to use
echo "message" | gpg --armor --encrypt --local-user "your@email.com"
```

Encryption Options

- `-armor (-a)` – Produces ASCII-armored output (text-based)
- `-encrypt (-e)` – Performs encryption operation
- `-r recipient@email.com` – Specifies recipient's email/key ID
- `-local-user` – Selects which of your keys to use
- Multiple recipients can be specified with multiple `-r` flags

4.2 Decrypting Messages

Decrypt received data:

```
# Decrypt from file
gpg --decrypt encrypted.asc

# Decrypt and save to file
gpg --decrypt encrypted.asc > decrypted.txt

# Decrypt with specific output
gpg --output decrypted.pdf --decrypt encrypted.pdf.gpg

# Decrypt directly in terminal
echo "-----BEGIN PGP MESSAGE-----
...encrypted content...
-----END PGP MESSAGE-----" | gpg --decrypt
```

4.3 Symmetric Encryption

Encrypt with a passphrase (no key pair needed):

```
# Symmetric encryption
gpg --symmetric --armor file.txt

# Decrypt symmetric encryption
gpg --decrypt file.txt.asc

# Specify cipher algorithm
gpg --symmetric --cipher-algo AES256 --armor sensitive.zip
```

Symmetric vs Asymmetric

Symmetric encryption uses the same passphrase for encryption and decryption. It's simpler but requires secure passphrase sharing.

Asymmetric encryption uses public/private key pairs. More secure for multiple recipients, but requires key management.

5 Digital Signatures

Digital signatures prove authenticity and integrity of data.

5.1 Creating Signatures

```
# Create detached signature
gpg --detach-sig document.pdf

# Create clearsigned text file
gpg --clearsign message.txt

# Sign and encrypt simultaneously
gpg --sign --encrypt -r recipient@email.com contract.docx

# Create inline signature
gpg --sign important-file.zip
```

Signature Types

1. **Detached Signature** – Separate .sig file, most common for software
2. **Clearsign** – Human-readable text with signature wrapper
3. **Inline Signature** – Signature embedded in encrypted file
4. **Sign & Encrypt** – Combined operation for authenticated encryption

5.2 Verifying Signatures

```
# Verify detached signature
gpg --verify document.pdf.sig

# Verify clearsigned file
gpg --verify message.txt.asc

# Verify and extract content
gpg --output original.txt --decrypt signed.txt.gpg
```

Signature Verification

A **"Good signature"** message indicates:

- File has not been modified
- Signature matches the claimed identity
- You have the signer's public key in your keyring

A **"BAD signature"** means:

- File was tampered with after signing
- Signature is invalid or corrupted
- **Do not trust the file!**

6 Advanced Key Operations

6.1 Edit Key Properties

Enter interactive key editing mode:

```
# Edit key interactively
gpg --edit-key your@email.com
```

Common Edit Commands

Interactive commands within `-edit-key`:

`expire`	Change expiration date
`adduid`	Add additional user ID (email)
`addkey`	Add subkey
`revkey`	Revoke a key
`trust`	Change trust level
`passwd`	Change passphrase
`save`	Save changes and exit
`quit`	Exit without saving

6.2 Create Revocation Certificate

Create this immediately after generating your key!

```
# Generate revocation certificate
gpg --output revocation.crt --gen-revoke your@email.com

# Import revocation certificate (when needed)
gpg --import revocation.crt

# Publish revoked key to keyserver
gpg --keyserver hkps://keys.openpgp.org --send-keys YOUR_KEY_ID
```

Revocation Certificate Safety

- Store revocation certificate in a **separate, secure location**
- Keep multiple copies (encrypted USB, password manager, safe deposit box)
- Anyone with this certificate can revoke your key
- Use it only if your private key is compromised or lost
- After revocation, generate a new key pair

6.3 Delete Keys

```
# Delete public key from keyring
gpg --delete-key "user@example.com"
```

```
# Delete private key (DANGEROUS!)
gpg --delete-secret-key "your@email.com"

# Delete both (requires two confirmations)
gpg --delete-secret-and-public-key "your@email.com"
```

7 Git Integration

Sign your Git commits and tags with GPG:

7.1 Configure Git for GPG

```
# Get your GPG key ID
gpg --list-secret-keys --keyid-format=long

# Configure Git to use GPG
git config --global user.signingkey YOUR_KEY_ID
git config --global commit.gpgsign true
git config --global tag.gpgsign true

# Set GPG program (if needed)
git config --global gpg.program gpg
```

7.2 Signing Commits & Tags

```
# Sign a commit (with auto-sign enabled)
git commit -m "Your commit message"

# Sign a commit (manual)
git commit -S -m "Signed commit message"

# Create signed tag
git tag -s v1.0.0 -m "Version 1.0.0"

# Verify signed commit
git log --show-signature

# Verify signed tag
git tag -v v1.0.0
```

Why Sign Commits?

Signing Git commits provides:

- **Authenticity** – Proves you authored the commit
- **Integrity** – Ensures code hasn't been modified
- **Non-repudiation** – Cannot deny creating the commit
- **Trust** – GitHub shows verified badge for signed commits

8 Security Best Practices

8.1 Key Management Best Practices

Essential Security Measures

1. **Strong Passphrases**
   - Minimum 20 characters
   - Mix of uppercase, lowercase, numbers, symbols
   - Use passphrases (multiple words) rather than passwords
   - Consider using a password manager
2. **Key Rotation Schedule**
   - High-security: Yearly rotation
   - Standard use: Every 2-3 years
   - Set expiration dates on keys
   - Generate new keys before old ones expire
3. **Secure Backups**
   - Export private keys to encrypted storage
   - Store in multiple physical locations
   - Test backup restoration periodically
   - Use `paperkey` for paper backups
4. **Fingerprint Verification**
   - Always verify key fingerprints before trusting
   - Use separate communication channel (phone, in-person)
   - Compare full 40-character fingerprint
   - Never trust keys solely based on email

8.2 Common Security Pitfalls

Avoid These Mistakes

1. **Never** share your private key with anyone
2. **Don't** use weak or dictionary-based passphrases
3. **Don't** store unencrypted private keys on cloud services
4. **Don't** use keys without expiration dates
5. **Don't** forget to create revocation certificates
6. **Don't** trust keys without verifying fingerprints
7. **Don't** reuse the same key for everything (consider subkeys)

9 Quick Reference

Essential Commands

Key Management:

- `gpg -gen-key`
- `gpg -k`
- `gpg -K`
- `gpg -export`
- `gpg -import`

Encryption:

- `gpg -e -r email`
- `gpg -d file.gpg`
- `gpg -c file`
- `gpg -armor`

Signatures:

- `gpg -sign`
- `gpg -verify`
- `gpg -clearsign`
- `gpg -detach-sig`

Advanced:

- `gpg -edit-key`
- `gpg -gen-revoke`
- `gpg -refresh-keys`
- `gpgconf -kill`

9.1 File Extension Guide

Extension	Description
`.gpg`	Binary encrypted/signed file
`.asc`	ASCII-armored encrypted/signed file (text-based)
`.sig`	Detached signature file
`.key`	Exported key file (public or private)
`.crt`	Revocation certificate

10 Troubleshooting

10.1 Common Issues

"Inappropriate ioctl for device" Error

Solution for Fish shell:

```
# Add to ~/.config/fish/config.fish
set -x GPG_TTY (tty)

# Reload configuration
source ~/.config/fish/config.fish
```

This error occurs when GPG cannot determine the terminal for passphrase entry.

"No secret key" Error

Possible causes:

- You don't have the private key for decryption
- Key was not properly imported
- Key has expired
- Wrong key selected

Solution:

```
# Verify you have the secret key
gpg -K

# Check key expiration
gpg --list-keys your@email.com
```

GPG Agent Not Responding

Solution:

```
# Kill and restart GPG agent
gpgconf --kill gpg-agent
gpgconf --launch gpg-agent

# Check agent status
gpg-connect-agent /bye
```

10.2 Useful Diagnostic Commands

```
# Show GPG version and capabilities
gpg --version
```

```
# Display detailed information about a key
gpg --list-keys --with-fingerprint your@email.com

# Show GPG configuration
gpgconf --list-components

# Test GPG functionality
echo "test" | gpg -e -r your@email.com | gpg -d

# Check keyserver connectivity
gpg --keyserver hkps://keys.openpgp.org --search-keys test@example.com
```

11 Additional Resources

11.1 Official Documentation

- **GNU Privacy Guard** – `https://gnupg.org/`
- **OpenPGP Standard** – `https://www.openpgp.org/`
- **Fish Shell Documentation** – `https://fishshell.com/docs/`
- **Homebrew** – `https://brew.sh/`

11.2 Keyservers

- **keys.openpgp.org** (Recommended) – `https://keys.openpgp.org/`
- **MIT PGP Keyserver** – `https://pgp.mit.edu/`
- **Ubuntu Keyserver** – `https://keyserver.ubuntu.com/`

11.3 Security Standards

- **AES (Advanced Encryption Standard)**
- **RSA (Rivest–Shamir–Adleman)**
- **SHA-2 Hash Functions**
- **ECC (Elliptic Curve Cryptography)**

Mathematical representation of encryption:

$$E_k(M) = C \quad \text{and} \quad D_k(C) = M$$

Where E is encryption, D is decryption, k is the key,
M is plaintext message, and C is ciphertext

End of GPG Cheatsheet

Stay secure, verify everything, backup your keys