🐙 GitHub Deployment Guide - SOaC Framework

Complete guide for pushing Phase 2 code to the GitHub repository.

📋 Table of Contents

🎯 Prerequisites

Before deploying to GitHub, ensure you have:

- ✅ Git installed on your system
- ✅ GitHub account with access to the repository
- ✅ Git configured with your credentials
- ✅ SSH key or HTTPS authentication set up

Verify Git Installation

```
git --version
# Should output: git version 2.x.x or higher
```

Configure Git (if not already done)

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

Test GitHub Authentication

```
# For SSH
ssh -T git@github.com

# For HTTPS
git ls-remote https://github.com/ge0mant1s/soac-framework.git
```

📦 Repository Information

- **Repository URL**: https://github.com/ge0mant1s/soac-framework
- **Clone URL (HTTPS)**: `https://github.com/ge0mant1s/soac-framework.git`
- **Clone URL (SSH)**: `git@github.com:ge0mant1s/soac-framework.git`
- **Default Branch**: `main`
- **Current Phase**: Phase 2 - Full-stack Application

✅ Pre-Deployment Checklist

Before pushing to GitHub, verify the following:

1. Code Quality

```
# Check for syntax errors in backend
cd backend
npm install
npm run dev  # Should start without errors
# Press Ctrl+C to stop

# Check for syntax errors in frontend
cd ../frontend
npm install
npm run dev  # Should start without errors
# Press Ctrl+C to stop
```

2. Environment Files

```
# Verify .env files are NOT being committed
git status | grep ".env"
# Should return nothing (or only .env.example files)

# Ensure .gitignore is working
cat .gitignore | grep ".env"
# Should show .env patterns
```

3. Documentation

```
# Check that all documentation is present
ls -la | grep -E "README|DEPLOYMENT|CHANGELOG"
ls -la docs/
```

4. Clean Working Directory

```
# Remove unnecessary files
rm -rf backend/node_modules frontend/node_modules
rm -rf backend/dist frontend/dist
rm -rf */package-lock.json

# Verify .gitignore is working
git status
# Should not show node_modules or .env files
```

🚀 Deployment Steps

Step 1: Navigate to Project Directory

```
cd /path/to/soac-framework
```

Step 2: Check Current Status

```
# View all changes
git status

# View modified files
git diff

# View staged files
git diff --staged
```

Step 3: Review Changes

```
# List all changed files
git status --short

# Review specific file changes
git diff backend/src/server.js
git diff frontend/src/App.jsx
```

Step 4: Stage Changes

Option A: Stage All Changes

```
git add .
```

Option B: Stage Specific Files/Directories

```
# Stage backend changes
git add backend/

# Stage frontend changes
git add frontend/

# Stage documentation
git add docs/ *.md

# Stage scripts
git add scripts/
```

Option C: Stage Interactively

```
# Review and stage changes one by one
git add -p
```

Step 5: Verify Staged Changes

```
# See what will be committed
git status

# See detailed changes to be committed
git diff --staged
```

Step 6: Commit Changes

Use Conventional Commit Messages:

```
# For Phase 2 complete implementation
git commit -m "feat: Phase 2 - Full-stack application with React frontend and Express
backend

- Implemented Express.js backend API with JWT authentication
- Created React frontend with Material-UI
- Added device management and rule management features
- Implemented dashboard with statistics and alerts
- Added authentication and protected routes
- Created comprehensive deployment documentation
- Added automated setup scripts for Mac"

# Or for smaller changes
git commit -m "docs: Add comprehensive deployment guides"
git commit -m "feat: Add automated Mac setup script"
git commit -m "fix: Correct CORS configuration in backend"
```

Commit Message Format:
- `feat:` - New features
- `fix:` - Bug fixes
- `docs:` - Documentation changes
- `style:` - Code style changes (formatting, etc.)
- `refactor:` - Code refactoring
- `test:` - Adding or updating tests
- `chore:` - Maintenance tasks

Step 7: Push to GitHub

```
# Push to main branch
git push origin main

# Or push with upstream tracking
git push -u origin main

# Force push (use with caution!)
# git push -f origin main  # Only if absolutely necessary!
```

Step 8: Verify on GitHub

1. Visit: https://github.com/ge0mant1s/soac-framework
2. Check that your commits appear in the commit history
3. Verify all files are present
4. Check that README.md displays correctly
5. Review the commit message and changes

🌿 Branch Management

Creating a Feature Branch

For larger changes or experimental features, use branches:

```
# Create and switch to a new branch
git checkout -b feature/phase2-enhancements

# Make changes...
git add .
git commit -m "feat: Add Phase 2 enhancements"

# Push branch to GitHub
git push -u origin feature/phase2-enhancements
```

Creating a Pull Request

1. Go to: https://github.com/ge0mant1s/soac-framework/pulls
2. Click “New pull request”
3. Select your feature branch
4. Add title and description
5. Click “Create pull request”

Merging Branches

```
# Switch to main branch
git checkout main

# Pull latest changes
git pull origin main

# Merge feature branch
git merge feature/phase2-enhancements

# Push merged changes
git push origin main

# Delete feature branch (optional)
git branch -d feature/phase2-enhancements
git push origin --delete feature/phase2-enhancements
```

Recommended Branch Strategy

```
main (production-ready code)
  └── develop (integration branch)
        ├── feature/phase2-frontend
        ├── feature/phase2-backend
        └── feature/deployment-guides
```

✅ Verification

Verify Commit on GitHub

```
# Get commit hash
git log --oneline -1

# View commit on GitHub
# https://github.com/ge0mant1s/soac-framework/commit/COMMIT_HASH
```

Verify All Files Are Pushed

```
# List all tracked files
git ls-tree -r main --name-only

# Compare local with remote
git diff main origin/main
# Should return nothing if everything is pushed
```

Clone and Test (Optional)

```
# Clone in a new directory to verify
cd /tmp
git clone https://github.com/ge0mant1s/soac-framework.git test-clone
cd test-clone

# Verify structure
ls -la
ls -la backend/ frontend/

# Cleanup
cd ..
rm -rf test-clone
```

🔧 Troubleshooting

Issue: “Permission denied (publickey)”

Solution: Set up SSH key or use HTTPS

```
# Use HTTPS instead
git remote set-url origin https://github.com/ge0mant1s/soac-framework.git

# Or add SSH key to GitHub
ssh-keygen -t ed25519 -C "your.email@example.com"
# Add ~/.ssh/id_ed25519.pub to GitHub settings
```

Issue: “Repository not found”

Solution: Verify repository access

```
# Check remote URL
git remote -v

# Update remote URL if incorrect
git remote set-url origin https://github.com/ge0mant1s/soac-framework.git
```

Issue: “Failed to push some refs”

Solution: Pull latest changes first

```
# Pull with rebase
git pull --rebase origin main

# Resolve conflicts if any
# Then push
git push origin main
```

Issue: “Your branch is behind ‘origin/main’“

Solution: Pull latest changes

```
# Pull latest changes
git pull origin main

# Or pull with merge
git pull --no-rebase origin main
```

Issue: “Accidentally committed .env file”

Solution: Remove from Git and commit again

```
# Remove .env from Git (keeps local file)
git rm --cached backend/.env
git rm --cached frontend/.env

# Commit the removal
git commit -m "chore: Remove .env files from Git"

# Push changes
git push origin main

# Verify .gitignore includes .env
cat .gitignore | grep ".env"
```

Issue: “Large files causing push to fail”

Solution: Remove large files and use .gitignore

```
# Remove large files from staging
git reset HEAD large-file.zip

# Add to .gitignore
echo "*.zip" >> .gitignore
echo "*.pdf" >> .gitignore  # if needed

# Commit .gitignore change
git add .gitignore
git commit -m "chore: Update .gitignore to exclude large files"
```

🎯 Quick Reference Commands

Daily Workflow

```
# 1. Start work - Pull latest
git pull origin main

# 2. Make changes
# ... edit files ...

# 3. Check what changed
git status
git diff

# 4. Stage and commit
git add .
git commit -m "feat: Your feature description"

# 5. Push changes
git push origin main
```

Undo Changes

```
# Discard local changes (before staging)
git checkout -- filename

# Unstage file (after git add)
git reset HEAD filename

# Undo last commit (keep changes)
git reset --soft HEAD~1

# Undo last commit (discard changes)
git reset --hard HEAD~1
```

View History

```
# View commit history
git log --oneline --graph --all

# View changes in last commit
git show

# View specific file history
git log --follow filename
```

📚 Additional Resources

- Git Documentation (https://git-scm.com/doc)
- GitHub Guides (https://guides.github.com/)
- Conventional Commits (https://www.conventionalcommits.org/)
- GitHub Flow (https://guides.github.com/introduction/flow/)

✨ Best Practices

1. **Commit Often**: Make small, frequent commits
2. **Descriptive Messages**: Write clear commit messages
3. **Review Changes**: Always review before committing
4. **Pull Before Push**: Always pull before pushing
5. **Use Branches**: Use feature branches for larger changes
6. **Never Commit Secrets**: Keep .env files out of Git
7. **Test Before Push**: Ensure code works before pushing

Next Steps: After pushing to GitHub, proceed with Local Mac Deployment (./MAC_DEPLOYMENT.md) or One.com Hosting (./ONECOM_DEPLOYMENT.md).

Last Updated: November 13, 2024 - Phase 2