GitHub Integration Guide for APBeeper Bot

This guide covers setting up GitHub integration for automated deployment, version control, and release management of your APBeeper Discord bot.

Table of Contents

Repository Setup

1. Initialize Git Repository

```
cd /path/to/apbeeper_bot
git init
git add .
git commit -m "Initial commit: APBeeper Discord bot"
```

2. Create GitHub Repository

1. Go to GitHub (https://github.com) and create a new repository
2. Name it `apbeeper-bot` or similar
3. Set it to private (recommended for bot tokens)
4. Don't initialize with README (we already have files)

3. Connect Local Repository to GitHub

```
git remote add origin https://github.com/yourusername/apbeeper-bot.git
git branch -M main
git push -u origin main
```

4. Repository Structure

Ensure your repository has this structure:

```
apbeeper_bot/
├── .github/
│   └── workflows/
│       └── deploy.yml
├── src/
│   ├── commands/
│   ├── events/
│   ├── utils/
│   └── index.js
├── docs/
├── scripts/
├── logs/
├── data/
├── .env.example
├── .gitignore
├── package.json
├── ecosystem.config.js
└── README.md
```

5. Update .gitignore

Ensure your `.gitignore` includes:

```
# Dependencies
node_modules/
npm-debug.log*

# Environment variables
.env
.env.local
.env.production

# Logs
logs/
*.log

# Database
data/*.db
data/*.sqlite

# PM2
.pm2/

# OS generated files
.DS_Store
Thumbs.db

# IDE files
.vscode/
.idea/

# Temporary files
tmp/
temp/
```

GitHub Actions Configuration

1. Create Deployment Workflow

The workflow file is already created at `.github/workflows/deploy.yml`. Here's the complete configuration:

```yaml
name: Deploy APBeeper Bot

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]
  workflow_dispatch:

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v4

    - name: Setup Node.js
      uses: actions/setup-node@v4
      with:
        node-version: '20'
        cache: 'npm'

    - name: Install dependencies
      run: npm ci

    - name: Run tests
      run: npm test
      continue-on-error: true

  deploy:
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main' && github.event_name == 'push'

    steps:
    - uses: actions/checkout@v4

    - name: Setup SSH
      uses: webfactory/ssh-agent@v0.9.0
      with:
        ssh-private-key: ${{ secrets.SSH_PRIVATE_KEY }}

    - name: Add VPS to known hosts
      run: |
        ssh-keyscan -p ${{ secrets.SSH_PORT || 22 }} ${{ secrets.HOST }} >> ~/.ssh/
known_hosts

    - name: Deploy to VPS
      run: |

ssh ${{ secrets.USER }}@${{ secrets.HOST }} -p ${{ secrets.SSH_PORT || 22 }} <<
'ENDSSH'
          cd ${{ secrets.APP_PATH }}

          # Pull latest changes
          git pull origin main

          # Install/update dependencies
```

```
     npm ci --production

     # Run database migrations if needed
     if [ -f "scripts/migrate_db.sh" ]; then
       ./scripts/migrate_db.sh
     fi

     # Restart the bot with PM2
     pm2 restart apbeeper-bot || pm2 start ecosystem.config.js

     # Save PM2 configuration
     pm2 save

     echo "Deployment completed successfully!"
    ENDSSH

  - name: Deployment Status
    if: success()
    run: echo "  Deployment successful!"

  - name: Deployment Failed
    if: failure()
    run: echo "  Deployment failed!"
```

2. Alternative Workflow with Docker

If you prefer using Docker:

```yaml
name: Deploy with Docker

on:
  push:
    branches: [ main ]

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v4

    - name: Setup SSH
      uses: webfactory/ssh-agent@v0.9.0
      with:
        ssh-private-key: ${{ secrets.SSH_PRIVATE_KEY }}

    - name: Deploy with Docker
      run: |
        ssh ${{ secrets.USER }}@${{ secrets.HOST }} << 'ENDSSH'
          cd ${{ secrets.APP_PATH }}
          git pull origin main

          # Build and restart containers
          docker-compose down
          docker-compose build --no-cache
          docker-compose up -d

          # Clean up unused images
          docker image prune -f
        ENDSSH
```

Environment Variables Management

1. GitHub Secrets Setup

Go to your repository → Settings → Secrets and variables → Actions

Add these secrets:

Required Secrets

- `SSH_PRIVATE_KEY` : Your VPS SSH private key
- `HOST` : Your VPS IP address or domain
- `USER` : SSH username (e.g., `apbeeper`)
- `APP_PATH` : Path to your bot on VPS (e.g., `/home/apbeeper/apbeeper_bot`)

Optional Secrets

- `SSH_PORT` : SSH port (default: 22)
- `DISCORD_TOKEN` : Bot token (if needed for testing)
- `DATABASE_URL` : Database connection string

2. Generate SSH Key for Deployment

On your local machine:

```
# Generate deployment key
ssh-keygen -t rsa -b 4096 -C "github-actions-deploy" -f ~/.ssh/github_deploy_key

# Copy public key to VPS
ssh-copy-id -i ~/.ssh/github_deploy_key.pub apbeeper@your_vps_ip

# Copy private key content to GitHub Secrets
cat ~/.ssh/github_deploy_key
```

3. Environment Variables on VPS

Create production environment file:

```
# On your VPS
cd /home/apbeeper/apbeeper_bot
cp .env.example .env.production
nano .env.production
```

Update with production values:

```
NODE_ENV=production
DISCORD_TOKEN=your_production_bot_token
DATABASE_URL=your_production_database_url
LOG_LEVEL=info
PORT=3000
```

4. Environment-Specific Configurations

Update your `ecosystem.config.js`:

```
module.exports = {
  apps: [{
    name: 'apbeeper-bot',
    script: 'src/index.js',
    instances: 1,
    autorestart: true,
    watch: false,
    max_memory_restart: '1G',
    env: {
      NODE_ENV: 'development'
    },
    env_production: {
      NODE_ENV: 'production',
      PORT: 3000
    },
    error_file: './logs/err.log',
    out_file: './logs/out.log',
    log_file: './logs/combined.log',
    time: true
  }]
};
```

Deployment Workflow

1. Development Workflow

```
# Create feature branch
git checkout -b feature/new-command

# Make changes and commit
git add .
git commit -m "Add new moderation command"

# Push to GitHub
git push origin feature/new-command

# Create Pull Request on GitHub
# After review and approval, merge to main
```

2. Automatic Deployment Process

1. Push to `main` branch triggers GitHub Actions
2. Tests run automatically
3. If tests pass, deployment begins
4. Code is pulled on VPS
5. Dependencies are updated
6. Database migrations run (if any)
7. PM2 restarts the bot
8. Deployment status is reported

3. Manual Deployment Trigger

You can manually trigger deployment:
1. Go to Actions tab in your repository
2. Select “Deploy APBeeper Bot” workflow
3. Click “Run workflow”
4. Choose branch and click “Run workflow”

Release Management

1. Semantic Versioning

Use semantic versioning (MAJOR.MINOR.PATCH):
- **MAJOR**: Breaking changes
- **MINOR**: New features (backward compatible)
- **PATCH**: Bug fixes

2. Creating Releases

```
# Tag a release
git tag -a v1.0.0 -m "Release version 1.0.0"
git push origin v1.0.0
```

3. Automated Release Workflow

Add to `.github/workflows/release.yml`:

```
name: Create Release

on:
  push:
    tags:
      - 'v*'

jobs:
  release:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v4

    - name: Create Release
      uses: actions/create-release@v1
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
      with:
        tag_name: ${{ github.ref }}
        release_name: Release ${{ github.ref }}
        draft: false
        prerelease: false
        body: |
          ## Changes in this Release
          - Feature updates
          - Bug fixes
          - Performance improvements

          ## Installation
          See [deployment documentation](./docs/hosting.md) for installation instruc-
tions.
```

4. Changelog Management

Maintain a `CHANGELOG.md` file:

```
# Changelog

## [1.0.0] - 2025-06-23
### Added
- Initial release of APBeeper bot
- Multi-server support
- Role management commands
- Channel management features
- Moderation tools

### Changed
- Improved error handling

### Fixed
- Database connection issues
```

Security Best Practices

1. Repository Security

- Keep repository private if it contains sensitive information
- Use `.env.example` instead of committing actual environment files
- Regularly audit dependencies with `npm audit`
- Enable branch protection rules

2. SSH Key Management

- Use dedicated SSH keys for deployment
- Rotate keys regularly
- Limit key permissions on the server
- Monitor SSH access logs

3. Secrets Management

- Never commit secrets to the repository
- Use GitHub Secrets for sensitive data
- Regularly rotate tokens and passwords
- Limit secret access to necessary workflows

4. Branch Protection

Configure branch protection rules:
1. Go to Settings → Branches
2. Add rule for `main` branch
3. Enable:
- Require pull request reviews
- Require status checks to pass
- Require branches to be up to date
- Include administrators

Troubleshooting

Common Issues and Solutions

1. SSH Connection Failed

```
# Test SSH connection manually
ssh -T git@github.com

# Check SSH key format in GitHub Secrets
# Ensure private key includes headers:
-----BEGIN OPENSSH PRIVATE KEY-----
...key content...
-----END OPENSSH PRIVATE KEY-----
```

2. Deployment Fails

```
# Check GitHub Actions logs
# Common issues:
# - Wrong file paths
# - Permission issues
# - Missing dependencies

# Debug on VPS
ssh apbeeper@your_vps_ip
cd /home/apbeeper/apbeeper_bot
git status
npm list
pm2 status
```

3. Environment Variables Not Loading

```
# Check .env file exists and has correct permissions
ls -la .env*
cat .env.production

# Verify PM2 is using correct environment
pm2 show apbeeper-bot
```

4. Database Migration Issues

```
# Check migration script permissions
chmod +x scripts/migrate_db.sh

# Run migration manually
./scripts/migrate_db.sh

# Check database connection
npm run db:test
```

5. PM2 Process Issues

```
# Check PM2 status
pm2 status
pm2 logs apbeeper-bot

# Restart PM2
pm2 restart apbeeper-bot

# Reset PM2 if needed
pm2 delete apbeeper-bot
pm2 start ecosystem.config.js
pm2 save
```

Debugging GitHub Actions

1. Check the Actions tab in your repository
2. Click on the failed workflow run
3. Expand the failed step to see error details

4. Common fixes:
   - Update secrets with correct values
   - Fix file paths in workflow
   - Ensure VPS has required dependencies

Monitoring Deployments

```
# On VPS, monitor deployment
tail -f /home/apbeeper/apbeeper_bot/logs/combined.log

# Check PM2 status
pm2 monit

# View system resources
htop
```

Advanced Configuration

1. Multi-Environment Setup

Create separate workflows for staging and production:

```
# .github/workflows/deploy-staging.yml
name: Deploy to Staging
on:
  push:
    branches: [ develop ]
# ... staging-specific configuration

# .github/workflows/deploy-production.yml
name: Deploy to Production
on:
  push:
    branches: [ main ]
# ... production-specific configuration
```

2. Rollback Strategy

Add rollback capability:

```
- name: Create backup before deployment
  run: |
    ssh ${{ secrets.USER }}@${{ secrets.HOST }} << 'ENDSSH'
      cd ${{ secrets.APP_PATH }}
      git tag backup-$(date +%Y%m%d-%H%M%S)
      git push origin --tags
    ENDSSH
```

3. Health Checks

Add health check after deployment:

```
- name: Health Check
  run: |
    sleep 30
    ssh ${{ secrets.USER }}@${{ secrets.HOST }} << 'ENDSSH'
      if pm2 list | grep -q "online.*apbeeper-bot"; then
        echo "  Bot is running"
      else
        echo "  Bot failed to start"
        exit 1
      fi
    ENDSSH
```

Next Steps

After setting up GitHub integration:
1. Follow the Hosting Documentation (./hosting.md) for deployment procedures
2. Configure monitoring and logging
3. Set up the Bot Distribution (./bot_distribution.md) system

---

Note: Always test your deployment workflow in a staging environment before deploying to production.