Git Version Control

Module 1 of 7

Presenter: Vương Cường — Research Assistant, BAI Lab

Introduction

What is Git?

Git = **Distributed Version Control System** enabling collaborative research development

Why Use Git for Research?

Core Features

- **Complete project history tracking** - Every change is recorded
- **Branch-based experimentation** - Safe parallel development
- **Offline capability with remote sync** - Work anywhere, sync later
- **Merge conflict resolution** - Handle collaborative conflicts systematically

Why Essential for Research?

- **Reproducible experiments** through version control
- **Collaborative development** without file conflicts
- **Experiment tracking** with dedicated branches
- **Rollback capability** to any previous state

Comparison with Traditional Methods

Criteria	Traditional Methods	Git + GitHub
Change History	Manual versioning (v1_final_FINAL.zip)	Complete commit history with diffs
Experimentation	Copy entire folders → messy structure	Cheap branches, clean merging
Collaboration	File overwriting conflicts	Systematic merge conflict resolution
Code Review	Scattered email/chat comments	Standardized Pull Request process
Reproducibility	Environment dependency issues	Fixed commits + CI/CD automation

Essential Commands (80/20 Principle)

Daily Workflow (Covers 80% of Usage)

```
# 1. Basic workflow loop
git add .                              # Stage all changes
git commit -m "feat: add preprocessing" # Commit with message
git push origin feature/preprocessing  # Push to remote branch

# 2. Branch management
git checkout -b feature/new-experiment  # Create and switch to new branch
git checkout develop                   # Switch to develop branch
git merge feature/new-experiment       # Merge feature into current branch

# 3. Status and history
git status                             # Check current state
git log --oneline                      # View commit history
git diff                               # See unstaged changes
```

Essential Branch Operations

```
# Clone repository and setup
git clone <repository-url>
cd <repository-name>

# Always work from develop branch
git checkout develop
git pull origin develop                # Get latest changes
```

Lab Demo with base-research-repo

Step-by-Step Workflow

```
# 1. Setup repository
git clone https://github.com/bailab/base-research-repo
cd base-research-repo

# 2. Create feature branch from develop
git checkout develop
git pull origin develop                  # Ensure latest version
git checkout -b feature/improve-config  # Create feature branch

# 3. Make meaningful changes
# Edit config/data_config.yaml
echo "
dataset:
  name: 'bailab_custom_dataset'
  preprocessing:
    normalize: true
```

Commit and Push Changes

```
# 4. Stage and commit changes
git add config/data_config.yaml
git status                                     # Verify changes

# 5. Commit with conventional format
git commit -m "feat(config): enhance data preprocessing options

- Add normalization and augmentation flags
- Configure batch size for training
- Prepare for advanced preprocessing pipeline

Closes #42"

# 6. Push feature branch
git push -u origin feature/improve-config
```

Important Considerations

Critical Git Rules for Lab

Branch Management Rules

- **Never work directly on `main` branch** - main is for stable releases only
- **All work happens on `develop` branch** - main working branch
- **Feature branches from develop** - `feature/descriptive-name`
- **All changes via Pull Requests** - no direct pushes to develop/main

Commit Message Standards

```
# Conventional Commits format
feat(scope): add new feature
fix(scope): fix bug in module
docs(scope): update documentation
style(scope): formatting changes
refactor(scope): code refactoring
test(scope): add or update tests
chore(scope): maintenance tasks
```

File Size Limitations

- **GitHub limit: 100MB per file** - use Git LFS for larger files
- **Repository size: <1GB ideal** - larger repos get warnings
- **Never commit large datasets** - use Hugging Face Hub instead

Security Best Practices

- **Never commit API keys or passwords**
- **Use environment variables** for sensitive data
- **Add `.env` files to `.gitignore`**
- **Review changes before committing** with `git diff`

Summary: Git Version Control

What We Covered

✅ **Git fundamentals** - Distributed version control system

✅ **Advantages over traditional methods** - Professional development workflow

✅ **Essential commands** - 80/20 principle for daily usage

✅ **Lab-specific workflow** - Feature branches and Pull Requests

✅ **Critical considerations** - Security, file limits, and best practices

Key Takeaways

1. **Git eliminates version control chaos** in research projects
2. **Branch-based workflow** enables safe experimentation
3. **Pull Requests** provide quality gates and code review
4. **Conventional commits** create readable project history
5. **Proper setup** prevents common pitfalls and conflicts

Next Steps

➡️ **Module 2: UV Package Manager** - Fast Python environment management
➡️ Setup your first feature branch in base-research-repo
➡️ Practice the commit → push → PR workflow