Principles of Software Construction: Objects, Design, and Concurrency

Git workflows (and maybe concurrency primitives)

Michael Hilton  Bogdan Vasilescu
Administrivia

- HW 5a presentations in Recitation in front of your classmates
  - Goal: illustrate how you achieve reuse in a domain
  - Describe domain, examples of plugins, decisions regarding generality vs specificity, overall project structure (e.g., how are plugins loaded), plugin interfaces
  - Similar to design review sessions

- Compete for “best framework”? 
Administrivia (2)

- Commit messages are (one of) your primary means of communication with the rest of the team.
  - This will become more obvious in HW5.

HW4b; Oops forgot to save. (Also bus is here)

Woke up and dreamt of some bugs. They were there.

HW 4b update (...kill me)

dropped my laptop, then I banged it on a table. Was reminded of impor...
Last week Tuesday
Three ways to move work around between branches

1) `git merge bugFix (into master)`
Move work from bugFix directly onto master

2) git rebase master
Copy a series of commits below current location

3) git cherry-pick C2 C4
Ways to undo work (1)

```
git reset HEAD~1
```

HEAD is the symbolic name for the currently checked out commit
Ways to undo work (2)

`git revert HEAD`
SYNCING LOCAL <---> REMOTE
Git

Every computer is a server and version control happens locally.
Git

How do you share code with collaborators if commits are *local*?

git commit
Git

You *push* your commits into their repositories / They *pull* your commits into their repositories

... But requires host names / IP addresses
GitHub typical workflow

Public repository where you make your changes public
GitHub typical workflow

git commit
GitHub typical workflow

git commit
GitHub typical workflow

push your local changes into a remote repository.
GitHub typical workflow

Collaborators can push too if they have access rights.
git push <remote> <branch>: upload local repository content to a remote repository

https://www.atlassian.com/git/tutorials-syncing/git-push
GitHub typical workflow

Without access rights, “don’t call us, we’ll call you” (pull from trusted sources) ... But again requires host names / IP addresses.
git pull <remote>: Fetch the specified remote’s copy of the current branch and immediately merge it into the local copy.

Equivalent to:

```
git fetch origin HEAD + git merge HEAD
```

Also possible: `git pull --rebase origin`
GitHub typical workflow

Instead, people maintain public remote “forks” of “main” repository on GitHub and push local changes.
GitHub typical workflow

Availability of new changes is signaled via "Pull Request".
GitHub typical workflow

Changes are pulled into main if PR accepted.
BRANCH WORKFLOWS

https://www.atlassian.com/git/tutorials/comparing-workflows
1. Centralized workflow

- Central repository to serve as the single point-of-entry for all changes to the project
- Default development branch is called master
  - all changes are committed into master
  - doesn’t require any other branches
Example

John works on his feature
Example

Mary works on her feature
Example

John publishes his feature
Example

John publishes his feature

git push origin master
Example

Mary tries to publish her feature

git push origin master
error: failed to push some refs to '/path/to/repo.git'
hint: Updates were rejected because the tip of your current branch is behind its remote counterpart. Merge the remote changes (e.g. 'git pull') before pushing again. See the 'Note about fast-forwards' in 'git push --help' for details.

Mary tries to publish her feature

git push origin master
Example

Mary rebases on top of John’s commit(s)

```
$ git pull --rebase origin master
```
Mary’s Repository

Origin/Master

Master

Master
Example

Mary resolves a merge conflict
Example

```
git rebase --continue
```
Example

Mary successfully publishes her feature
2. Git Feature Branch Workflow

- *All* feature development should take place in a dedicated branch instead of the master branch
- Multiple developers can work on a particular feature without disturbing the main codebase
  - master branch will never contain broken code (enables CI)
  - Enables pull requests (code review)
Example

Mary begins a new feature

```bash
git checkout -b marys-feature master

git status

git add <some-file>

git commit
```
Example

```
Example

Mary goes to lunch

git push -u origin marys-feature
```
Example

Mary finishes her feature

![Diagram showing a database with three users connected to it.]

git push
Example

Bill receives the pull request
Example

Mary makes the changes
Example - Merge pull request

Mary publishes her feature

- git checkout master
- git pull
- git pull origin marys-feature
- git push
3. Gitflow Workflow

- Strict branching model designed around the project release
  - Suitable for projects that have a scheduled release cycle
- Branches have specific roles and interactions
- Uses two branches
  - master stores the official release history; tag all commits in the master branch with a version number
  - develop serves as an integration branch for features
GitFlow feature branches (from develop)
GitFlow release branches (eventually into master)

- no new features after this point—only bug fixes, docs, and other release tasks
GitFlow hotfix branches used to quickly patch production releases
Summary

• Version control has many advantages
  – History, traceability, versioning
  – Collaborative and parallel development

• Collaboration with branches
  – Different workflows