THE UNIVERSITY OF WESTERN AUSTRALIA

git

Lecture 12

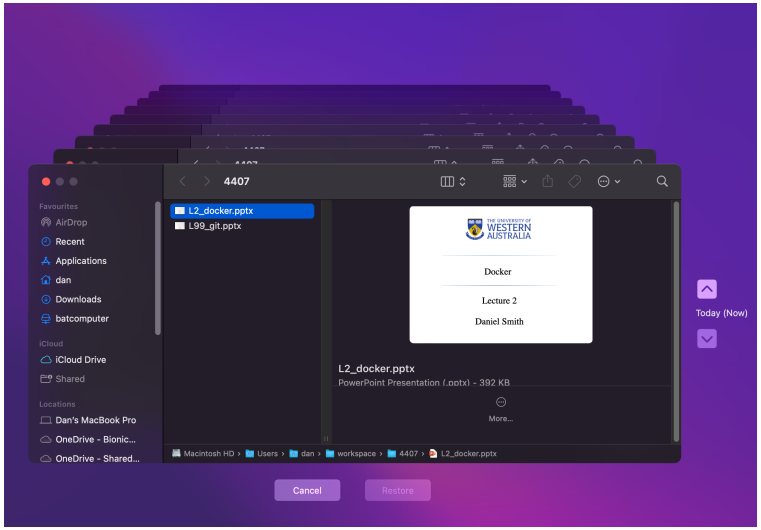
Michael Wise

Mistakes Happen

Git allows Time Travel (past only)

- Git is a free and open source distributed version control system
- “Version control” means keeping track of files at specific points in time

- Git tracks your files using snapshots called **commits**
- A directory containing files and directories tracked by git is called a **repository**
- Git allows you to roll back to an earlier version

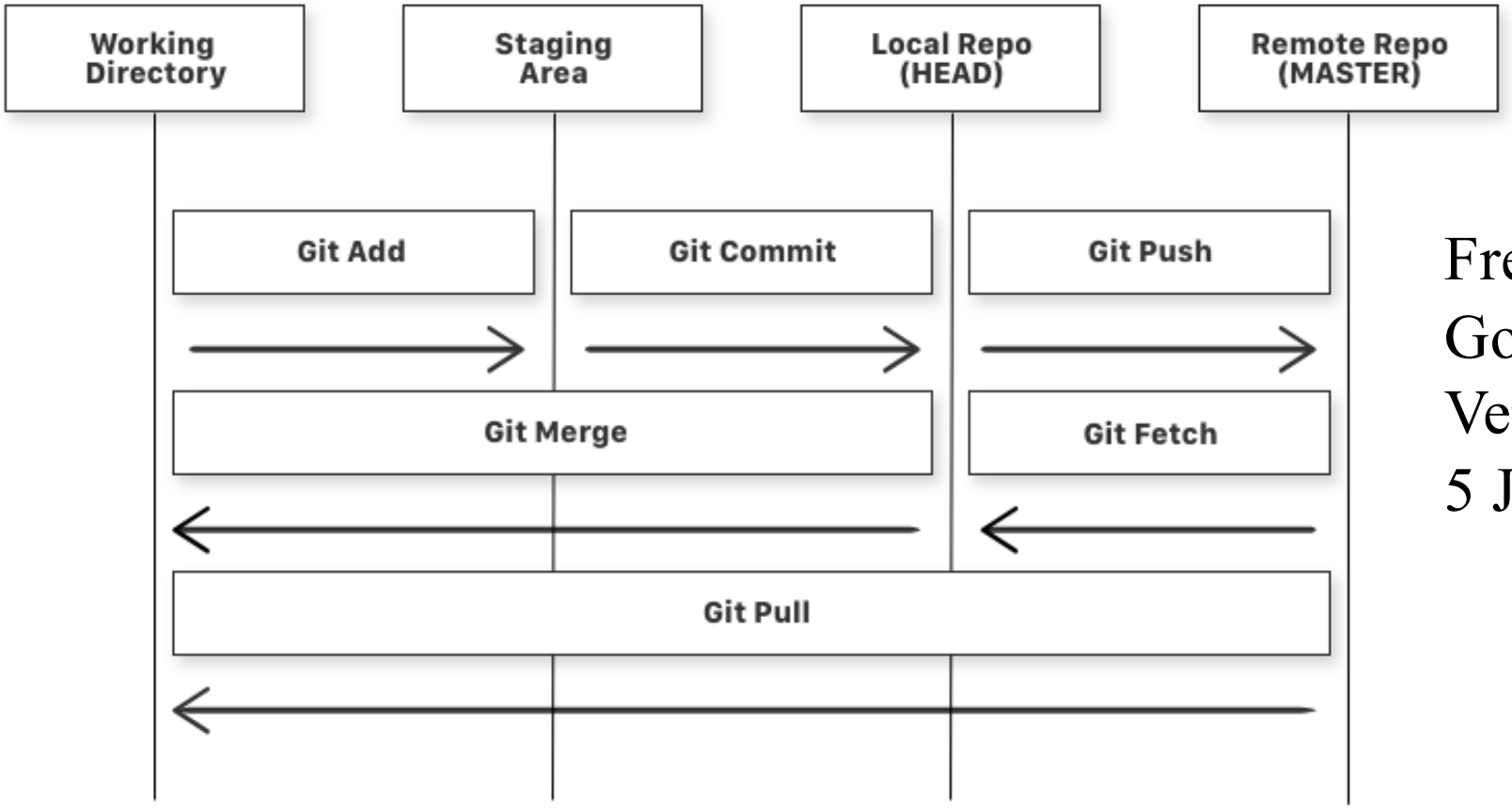


Git, Daniel Smith, 2022

Git Concepts

- A directory containing files and directories tracked by git is called a **repository**, or **repo**.
- First step is to create an empty repo using
  - *`git init`* *<name>*
- Each local repo has a *staging area* and a *head*
- The idea is that you *`git add`* files to the staging area, and then when you ready with the files to that point, you *`git commit`* them

Git Concepts



Freecodecamp,
Gowtham
Venkatesan
5 Jan 2019

git commit (local repo head); git push (remote master) for shared (global) repo.

Git Concepts

- Git merge brings files in from local repo to current directory; git fetch from global master to local repo

OR

- Git pull from master to local directory
  - *pull = fetch + merge*

git Commands Starter Pack

- `git init`

Create a new git repository in the current directory. This creates a `.git` directory containing all the information about the repo.

- `git status`

Show status of the current repo. Files shown by status can be in one of three states:

`staged` (will be included in the next commit)

`unstaged` (will not be included in the next commit)

`untracked` (not tracked by git)

Note: most git commands only work inside a repo.

git Commands Starter Pack

- `git add` *<files>*
  - *`git add .  # Contents of directory`*

Add each of the listed files to the set of files staged for inclusion in the next commit.

- `git commit`

Commit all staged files and open an editor to record an informative (!!) **commit message**. Options:

`-m` *<message>* Provide a message on the command line instead of opening an editor.

`-a` Automatically stage all modified and deleted files before committing. Does not stage untracked files. Convenient, but use with care.

When to Commit

- When you have made a noteworthy change
- When your code compiles for the first time
- Right before you make a big change that could break lots of things
- When you fix a bug
- When you add a new feature

Commit Messages

- 50 char summary header, blank line, then body paragraph(s) to explain the commit
- Commit message body should give context (**why** and **how**) to the commit. The actual changes (**what**) are already recorded by git in the diff.
- Use imperative mood ("it does this") to describe what the commit does. Don't use past tense ("I did this")
- See https://cbea.ms/git-commit/ for more on writing excellent commit messages
  - *By the way, each commit is a complete snapshot of the entire set of files (not just list of differences)*

Bad Commit Example

Add three lines to main.c

Code wasn't working so fix it. Now the thing should print for q1.

Bad Commit Example

WIP

Good Commit Example

Fix timeout bug in file parser

Add check to detect empty lines when parsing file and move loop counter increment outside of if conditions.

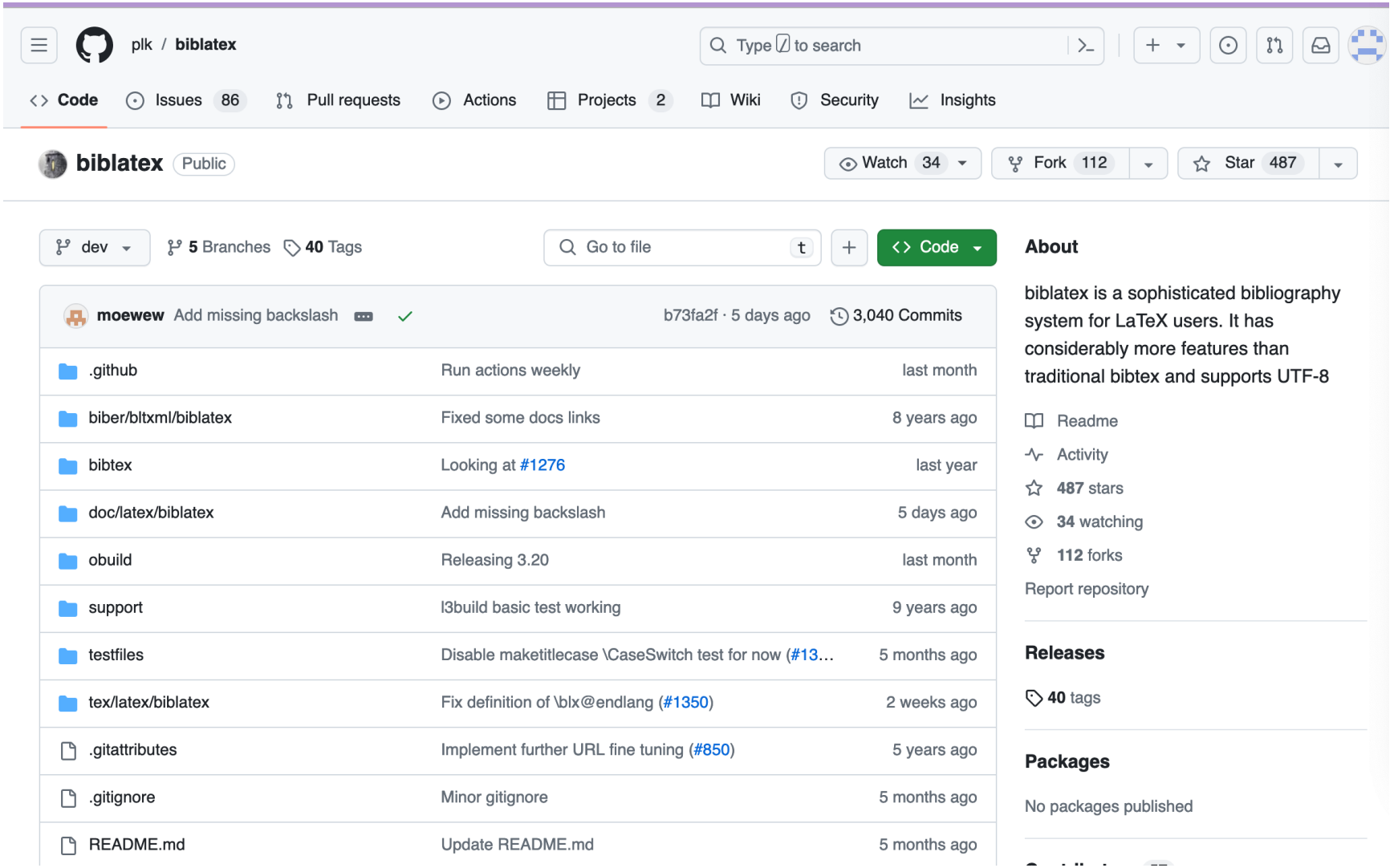
Empty lines did not match any of the parser checks and so were not recorded. Add if statement to record empty lines and add them to database.

The loop counter only incremented within if statements, risking infinite loop when conditions were true. Move increment outside if statements to correct this.

GitHub

- Git is a distributed version control system, so the same repo can exist on multiple computers and be synchronized between them
- There will often be one **remote** repo which multiple computers may sync with
- GitHub is a public repository hosting service
- GitHub keeps a **remote** copy of your repo in the cloud

GitHub Example - biblatex



git remote commands

- `git clone` *<repo>*

Download a copy of the specified repo. You do this at the start

- `git pull`

Fetch commits from the remote and merge them into the current branch

- `git push`

Push local commits on the current branch to the remote.

git Commands Continued

- `git diff`

Show changes (differences) between two data sources. By default, diff compares between all uncommitted changes and `HEAD`. Options:

`--cached` Ignore unstaged files (only compare staged files against `HEAD`)

- `git diff` *<commit>*

Compare against a specific commit, not `HEAD`

- `git diff` *<commit>* *<commit>*

Compare between two specific commits

- `git diff` *<file>*

Limit comparison to a particular file or directory

git diff examples

- `git diff --cached`

Compare all staged files with `HEAD`

- `git diff 4c8ae99`

Compare current working tree* (including all uncommitted work) with commit 4c8ae99

- `git diff 4c8ae99 36d0608`

Compare commit 4c8ae99 with commit 36d0608

- `git diff 4c8ae99 36d0608 foo/README.txt`

Compare changes to `foo/README.txt` between commits 4c8ae99 and 36d0608

*Working tree = repo directory and all subdirectories

git Commands Continued

- `git checkout` <*commit*>

Travel to a commit or branch. This command moves `HEAD` and updates all files in the working tree to match the new location. Options:

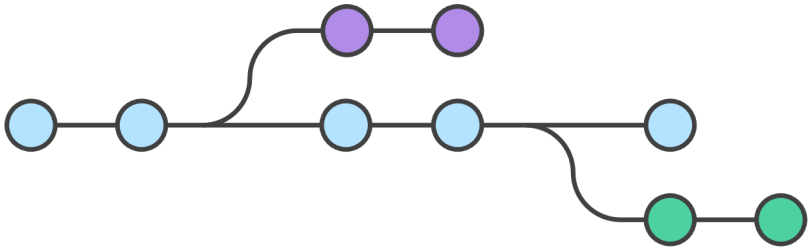
<*commit*> The commit or branch to travel to. Defaults to `HEAD`.

<*file*> If specified, do not travel (no change to `HEAD`). Instead, overwrite the specified file or directory with its contents at the specified commit. Great for restoring an old version of a file if you broke it!

Time isn't a straight line

- `git branch`

List branches in the repo.



- `git branch` *<name>*

Create a new branch at `HEAD` with the specified name.
Options:

`-m` Rename the current branch to the specified name.

`-d` Delete branch with the specified name. Branch must be fully merged

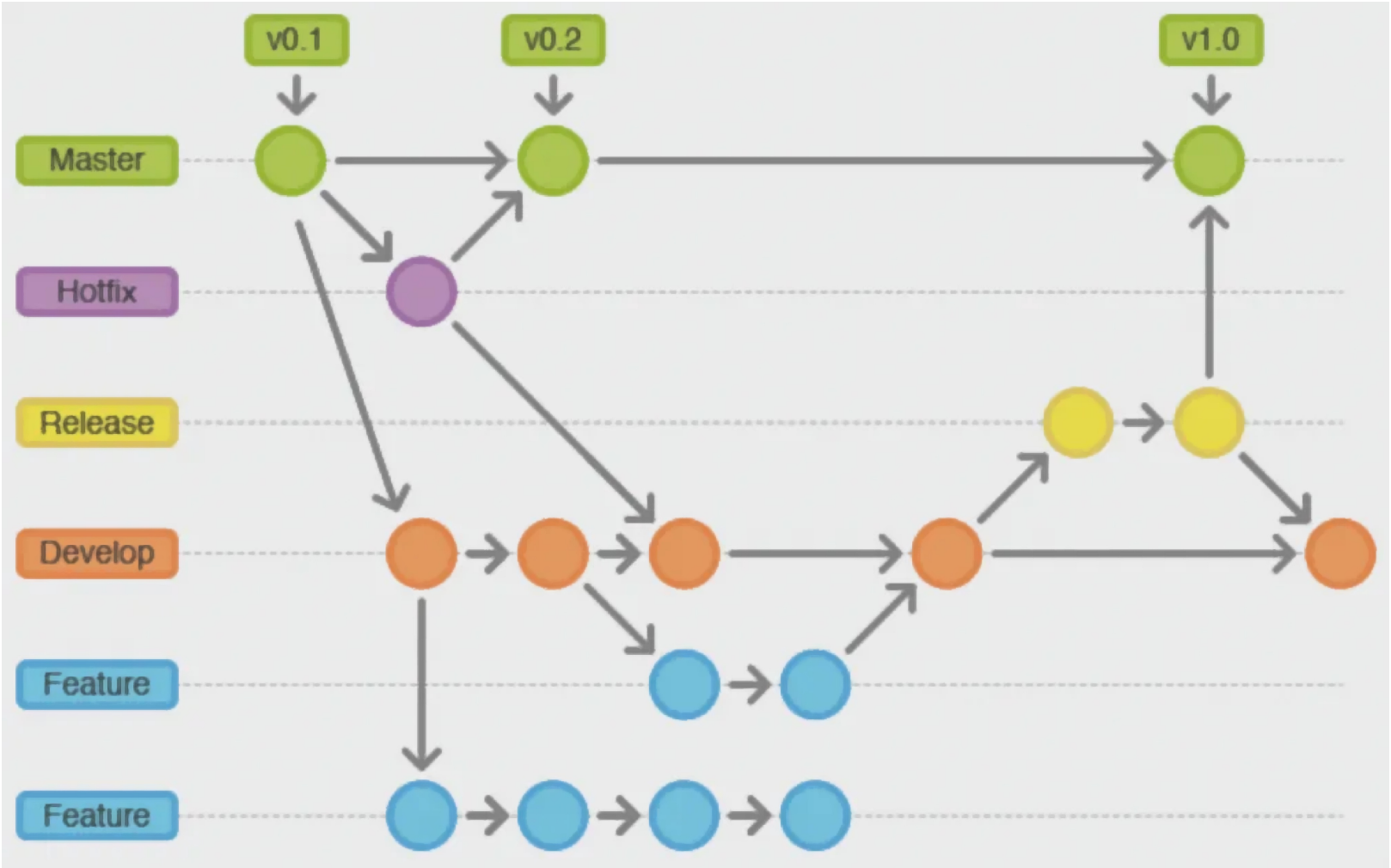
- `git checkout -b` *<name>*

Create and check out a new branch. Equivalent of:

`git branch` *<name>*

`git checkout` *<name>*

Good Branch Naming



"Best Practices for Branch Naming in Git for Successful CI/CD Implementation" Juan Tejeria, Medium, 17 May 2023

Reconnecting the timeline

- `git merge` <*branch*>

Merge the changes made in *branch* back into the current branch.

Merge is smart, but if both branches have different changes to the same line, there will be conflicts. You must resolve these before continuing the merge. You can see which files have conflicts using `git status`.

```
~/workspace/4407/git-practice merge-target > git merge complex-feature
Auto-merging crawl.txt
CONFLICT (content): Merge conflict in crawl.txt
Automatic merge failed; fix conflicts and then commit the result.
```

.gitignore

- It is often useful to store files in a git repo but not track them with git. Build files, temporary files and some config files are common examples.
- Create a file called `.gitignore` in repo root
- Each line is a search pattern for something to be ignored
- `/` is the directory separator
- Patterns containing `/` in the middle are relative to .gitignore. Otherwise, the pattern is applied at every level of the repo.
- `*` matches anything except `/`
- `?` Matches any single character except `/`
- `#` denotes a comment
- Don't forget to add `.gitignore` itself to git

.gitignore example

```
# vim temporary files
# these patterns apply everywhere
*.swp
*~
.netrwhist

# build directory and its content
# pattern applies to anything called build
build

# a particular file
# this pattern only applies to this file
foo/bar/my_config.txt
```

Workflow examples

- Single user:
  1. *Pull from remote - or simply clone and use*
  2. *Change file(s)*
  3. *Commit*
  4. *Push to remote*

- Multi user:
  1. *Pull from remote*
  2. *Create new feature branch*
  3. *Change files*
  4. *Commit*
  5. *Check out and pull main branch*
  6. *Merge feature branch onto main branch*
  7. *Push to remote*

Useful links

- Further reading:
  - *https://rogerdudler.github.io/git-guide/*
  - *https://github.com/danrs/git-practice*
  - *https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control*
  - *https://docs.github.com/en/get-started/getting-started-with-git*
  - *https://git-scm.com/docs/*
  - *https://www.atlassian.com/git/tutorials/*
  - *https://cbea.ms/git-commit/*

Acknowledgement

- This lecture was originally written by Daniel Smith, and though changed in many ways, it still makes substantial use of the original material. 🙏