THE UNIVERSITY OF WESTERN AUSTRALIA

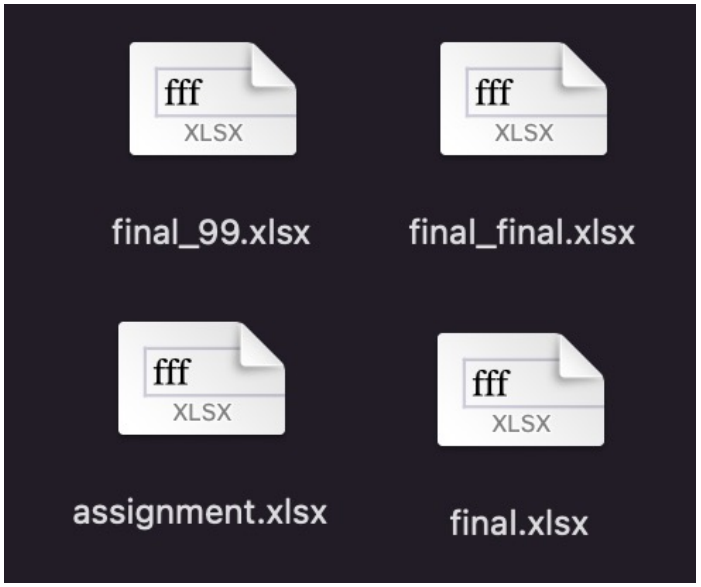
git

Lecture 12

Daniel Smith

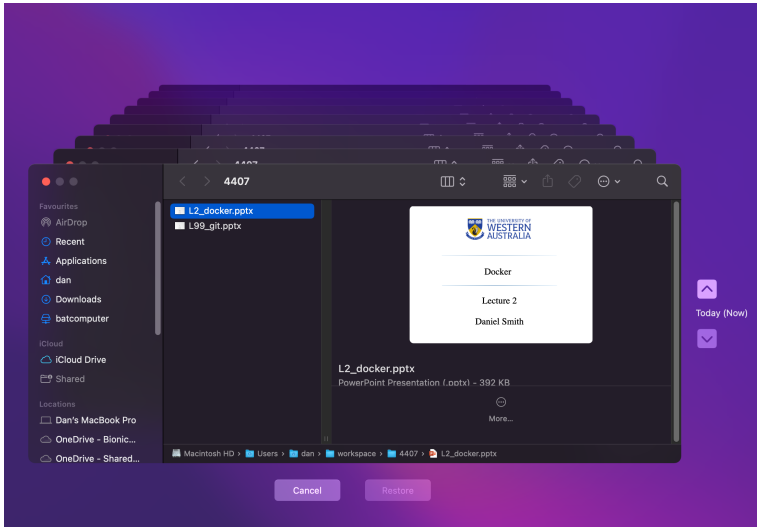
Mistakes, Time Travel and final_2.docx

```
dan:important $ rm -rf *
dan:important $ ls
dan:important $ echo noooo
noooo
```



Time Travel

- 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 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>*

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 a 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 ("do 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

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.

git log and git lola

- `git log` shows you the timeline of git commits
- `HEAD` is where you are on the timeline.

```
commit a96b64b491b0f0146460ac98b04f111ae95bb2f0 (HEAD -> master, origin/master, origin/HEAD)
Author: Dan Smith <5696173+danrs@users.noreply.github.com>
Date:   Mon Feb 10 13:11:34 2020 +0800

    Fixed typos and clarified some concepts

commit 1c6004b96a758dd0db885169d1ddbc777b0b3a56
Author: Dan Smith <5696173+danrs@users.noreply.github.com>
Date:   Thu Nov 7 15:20:48 2019 +0000

    Cleaned up formatting

commit c9d108f0cfea46a6c0e929f5a190563dd2636a37
Author: Dan Smith <5696173+danrs@users.noreply.github.com>
Date:   Thu Nov 7 15:11:54 2019 +0000

    typo fix
```

git log and git lola

- The git log isn't very concise or pretty, so I like to use `git lola`:

```
~/workspace/4407/git-practice master > git lola
* a96b64b (HEAD -> master, origin/master, origin/HEAD) Fixed typos and clarified some concepts
* 1c6004b Cleaned up formatting
* c9d108f typo fix
* 18a7706 Removed incorrect sentence
* 8b8e7e0 Brief spell check and correctness check
* 30a2004 fixed a typo
* 11bd06d Finished merge description in README
| * 915388c (origin/hoth) updated the blueprints
| | * bb8c987 (origin/yoda) wars not make one great
| |/
| | * 38b805d (origin/palp) caused some strife
| |/
| | * d81163b (origin/rebel-alliance) added a non-conflicting file
| |/
| * 36d0608 added stolen plans
|/
| * d45b5e0 (origin/merge-target) created changes that will conflict
| | * a47205d (origin/complex-feature-2) incorrected the lyrics
| |/
```

git log and git lola

Copy the following into ~/.gitconfig for your full color git lola action:

```
[alias]
    lol = log --graph --decorate --pretty=oneline --abbrev-commit
    lola = log --graph --decorate --pretty=oneline --abbrev-commit --all
[color]
    branch = auto
    diff = auto
    interactive = auto
    status = auto
```

Source: http://blog.kfish.org/2010/04/git-lola.html

git GUIs

- You can also use a GUI (graphical user interface) to view the git log. Most of these are pretty and have buttons for common git commands.
- `git-gui` and `gitk` are built-in GUI tools that come with git
- There are many third-party options as well, such as github desktop
- Graphical git interfaces will not be covered in this course

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:

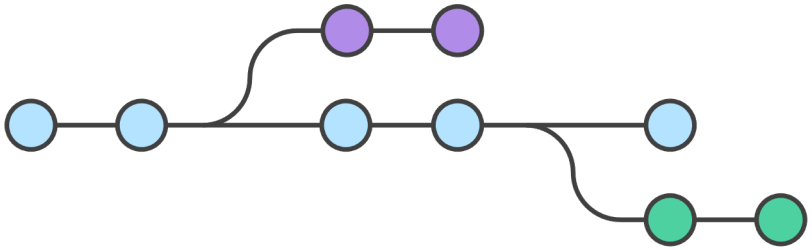
<*commit*> The commit or branch to travel to. Defaults to `HEAD`.

<*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>*

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.
```

Resolving conflicts

Below is a simple example of a conflict in `crawl.txt`:

```
STAR WARS
<<<<<<< HEAD
It is a planet of civil war.
=======
It is a period of PARTIES and maybe war.
>>>>>>> complex-feature
Rebel spaceships, striking
from a hidden base, have won
```

To resolve a conflict, first edit the conflicted file(s) and make any desired changes. Delete all the lines added by git.

Resolving conflicts

Below is our edited version of the file:

```
STAR WARS
It is a period of civil war.
Rebel spaceships, striking
from a hidden base, have won
```

When you are done editing all conflicted files, stage the changed files and continue the merge:

```
git add crawl.txt
git merge --continue
```

If desired, you can abort your merge instead:

```
git merge --abort
```

.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
```

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

git remote commands

- `git clone` *<repo>*

Download a copy of the specified repo

- `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.

Workflow examples

- Single user:
  1. *Pull from remote*
  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/*

Demo

- Create a repo and commit some files
- View the log
- Restore an old version of a file
- Pull and push with github
- Advanced: merging! (if time permits)