Git

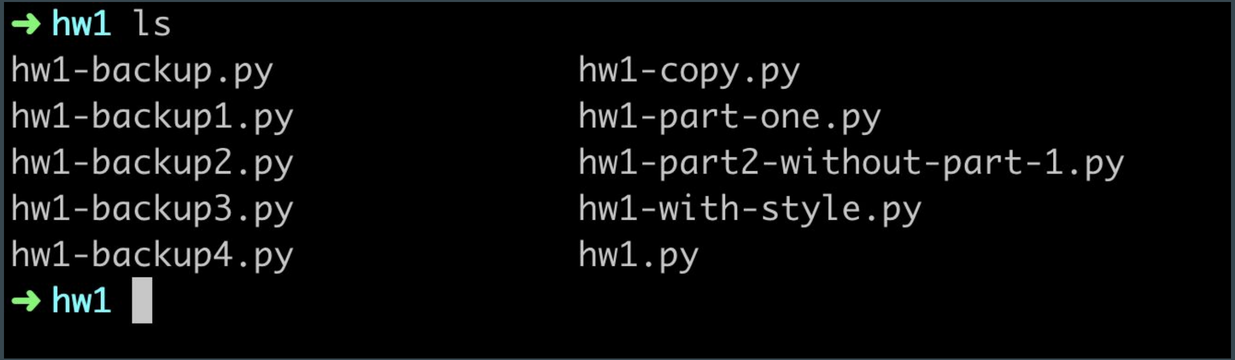
•••

07-131 Great Practical Ideas in CS

Instructors: Eduardo Feo-Flushing & Giselle Reis

Course website: https://web2.qatar.cmu.edu/cs/07131/

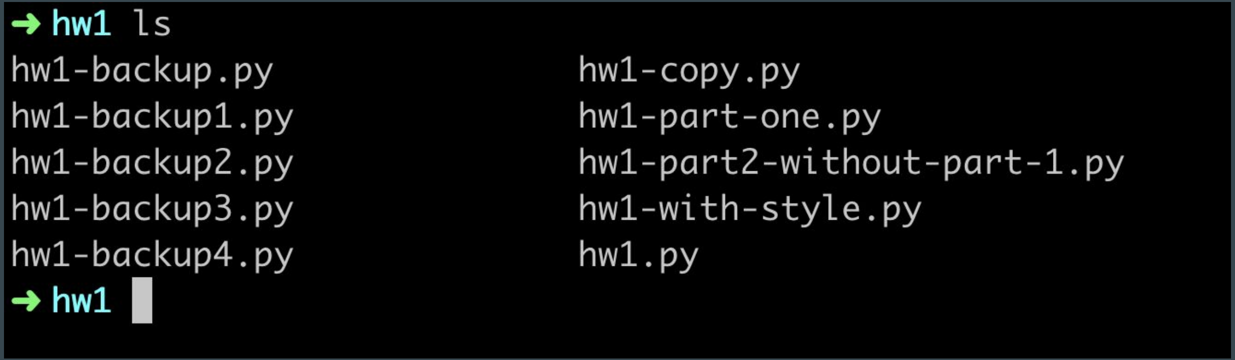
What is wrong with this?



```
➜ hw1 ls
hw1-backup.py                hw1-copy.py
hw1-backup1.py               hw1-part-one.py
hw1-backup2.py               hw1-part2-without-part-1.py
hw1-backup3.py               hw1-with-style.py
hw1-backup4.py               hw1.py
➜ hw1 
```

- Disorganized.
- Easy to get lost.
- Lots of copy & paste.
- Relies on your personal memory.
- Does not work when collaborating with others (specially for large projects).
- ...

How can we make it better?



```
➜ hw1 ls
hw1-backup.py                hw1-copy.py
hw1-backup1.py               hw1-part-one.py
hw1-backup2.py               hw1-part2-without-part-1.py
hw1-backup3.py               hw1-with-style.py
hw1-backup4.py               hw1.py
➜ hw1 ▌
```

Version Control Systems!!!

git mercurial SUBVERSION darcs

Do I really need to?

YES

- Collaborating without version control system is complicated.
- Versioning your files allows:
  - Time travel
  - Collection of statistics
  - Finding out who was responsible for what
  - Having a cool history of how a project has evolved
- Organized backup
- If you ever plan to work with other people, you will need to learn a version control system.

Git

```
git - the stupid content tracker
```

- Developed by Linus Torvalds in 2005 for the development of the Linux kernel
- Long story short:
  - Linux kernel development used BitKeeper
  - BitKeeper became paid
  - No existing version control system satisfied Linus' requirements
  - So he created his own, and git was born
- Developed in 3 days

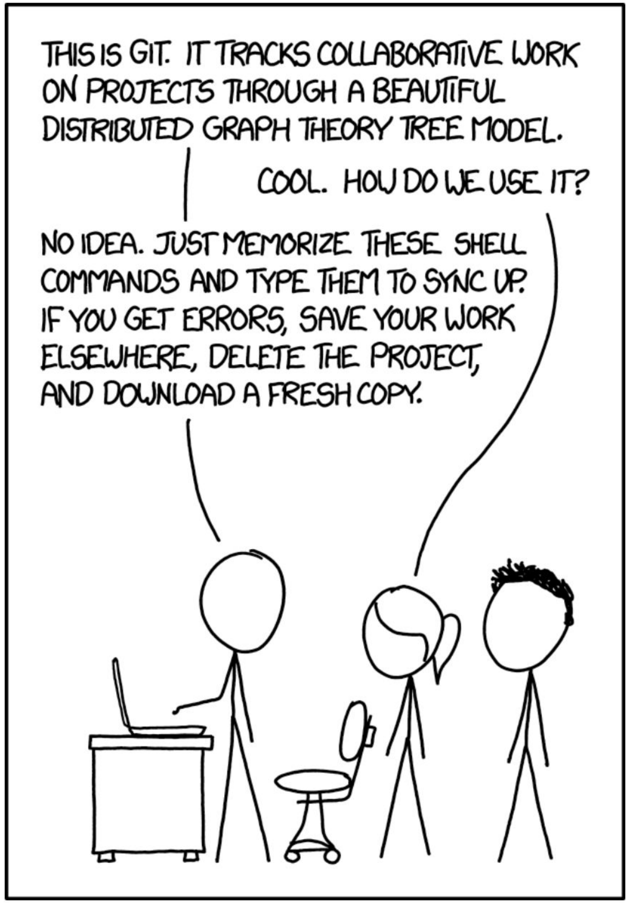
  "This is a stupid (but extremely fast) directory content manager. It doesn't do a whole lot, but what it _does_ do is track directory contents efficiently."
  (git's README file)
- Free and open source software \o/

Git repositories

- "Every Git directory on every computer is a full-fledged repository with complete history and full version-tracking abilities, independent of network access or a central server."
- Initialize repositories by writing `git init`
- Now you can add and commit stuff. Everything is local.
- Alternatively: `git clone`
  - Creates a local copy of a repo that exists elsewhere.
  - From this moment on, you have your own independent repository that is only loosely connected to the one that was cloned.
- When using git, most of the work is done *locally*, and the connection with other repositories can happen only sporadically (when you are ready!)

git commands

If that doesn't fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of "It's really pretty simple, just think of branches as..." and eventually you'll learn the commands that will fix everything.


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Local git – most used operations

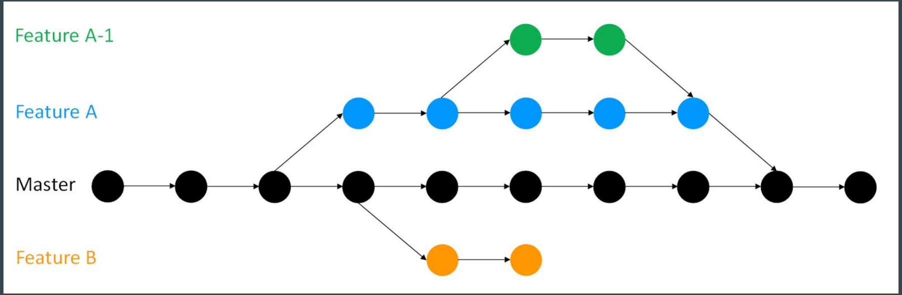
- `git status`
  - Use it often to find out what is happening!
  - branch
  - commits
  - untracked files
- `git add <path to file or folder>`
  - Empty folders are not added
  - Ignored files (in .gitignore) are not added
- `git commit`
  - Creates a "patch" – collection of changes (checkpoints you can go back to)
  - Ideally they are self-contained
  - Includes a message (describe what these changes are about)
  - `git config --global core.editor "vim"` (to configure where you will write the message)

Local git – other useful operations

- `git diff`
  - Shows changes to files that are not staged for commit (were not added)
  - Use `--cached` to see changes that are staged for commit
- `git log`
  - Shows all commits
  - Each commit has a hash that uniquely identifies it (a1d398fa82088bef5a6955acdc48c7259acf545a)
  - Press q to quit
  - use `--graph` to show a nice tree-like structure
- `git revert <commit hash>`
  - Creates a new commit which reverts the changes
  - Does not change history!

Git branches

- So far git history is a line of commits, but it can be a graph if we create **branches**!
- You work on one branch at a time.
- Adding files and commits will change only the *current branch*.
- Usual practice:
  - One **master** branch (main branch, stable)
  - One branch to develop each new feature
  - Once the changes in a branch are stable, they are **merged back** to the main branch.
  - Attention: for git, all branches are equal!



Git branch commands

- `git branch`
  - Lists all branches
  - Use `--vv` for more details
- `git branch <name>`
  - Creates a new branch from the current commit called name
- `git checkout <name>`
  - Switches to a branch called name
  - name can be a commit hash (you will be in a *detached head* state...)
- `git merge <name>`
  - Merges all the changes from branch name into the current branch
  - May create a "merge commit" if the current branch has commits that are not in name

Some git lifesaver commands

- When in a detached head state, use `git checkout -b <branch name>`to create a new branch at the commit you are at.
- If conflicts arise, use `git mergetool` to fix them (look for the <<< === >>> lines which surround the conflicted parts).
- To add stuff to your last commit, add the changes and `git commit --amend`
- To add only partial changes in a file: `git add -p <file>`
    - Git will prompt you asking if you want to add each change.

- If you have not noticed, we did not talk about github. **Github is not git.**

It's Romance Lab time!