Git for Version Control

These slides are heavily based on slides created by Ruth Anderson.
images taken from http://git-scm.com/book/en/
The second part is based on Mike Zastre's slides.

About Git

- Created by Linus Torvalds, creator of Linux, in 2005
  - Came out of Linux development community
  - Designed to do version control on Linux kernel

- Goals of Git:
  - Speed
  - Support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects efficiently

  - *(A "git" is a cranky old man. Linus meant himself.)*

Installing/learning Git

- Git website: http://git-scm.com/
  - Free on-line book: http://git-scm.com/book
  - Reference page for Git: http://gitref.org/index.html
  - Git tutorial: http://schacon.github.com/git/gittutorial.html
  - Git for Computer Scientists:
    - http://eagain.net/articles/git-for-computer-scientists/

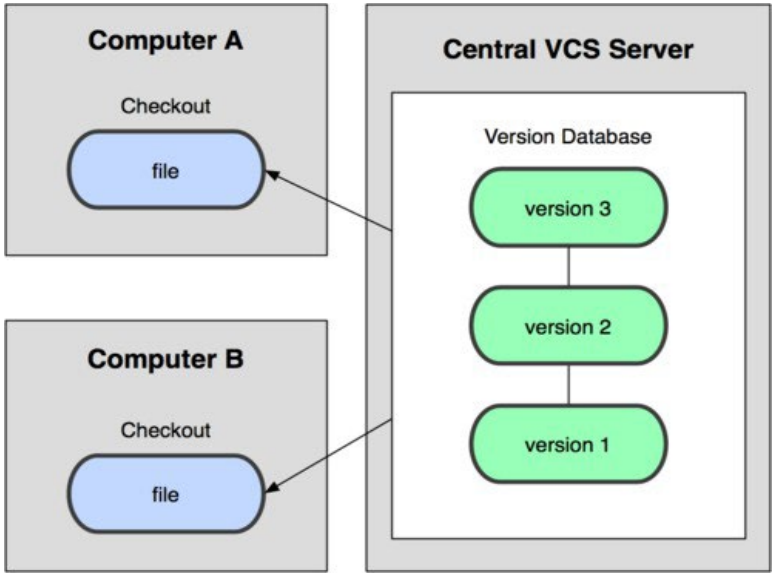
- At command line: *(where verb = config, add, commit, etc.)*
  - `git help verb`

Centralized VCS

- In Subversion, CVS, Perforce, etc.
  A central server repository (repo)
  holds the "official copy" of the code
  - the server maintains the sole
    version history of the repo

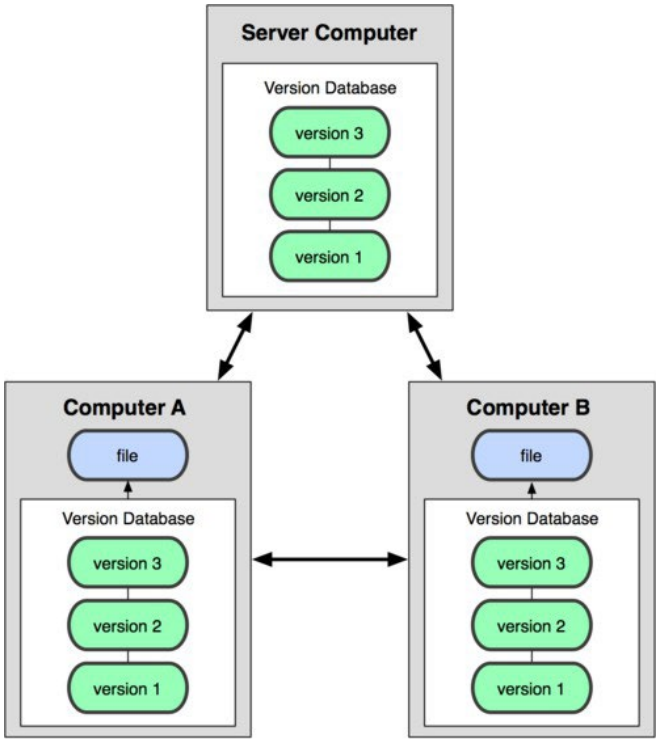
- You make "checkouts" of it
  to your local copy
  - you make local modifications
  - your changes are not versioned

- When you're done, you
  "check in" back to the server
  - your checkin increments the repo's version



Distributed VCS (Git)

- In git, mercurial, etc., you don't "checkout" from a central repo
  - you "clone" it and "pull" changes from it
- Your local repo is a complete copy of everything on the remote server
  - yours is "just as good" as theirs
- Many operations are local:
  - check in/out from *local* repo
  - commit changes to *local* repo
  - local repo keeps version history

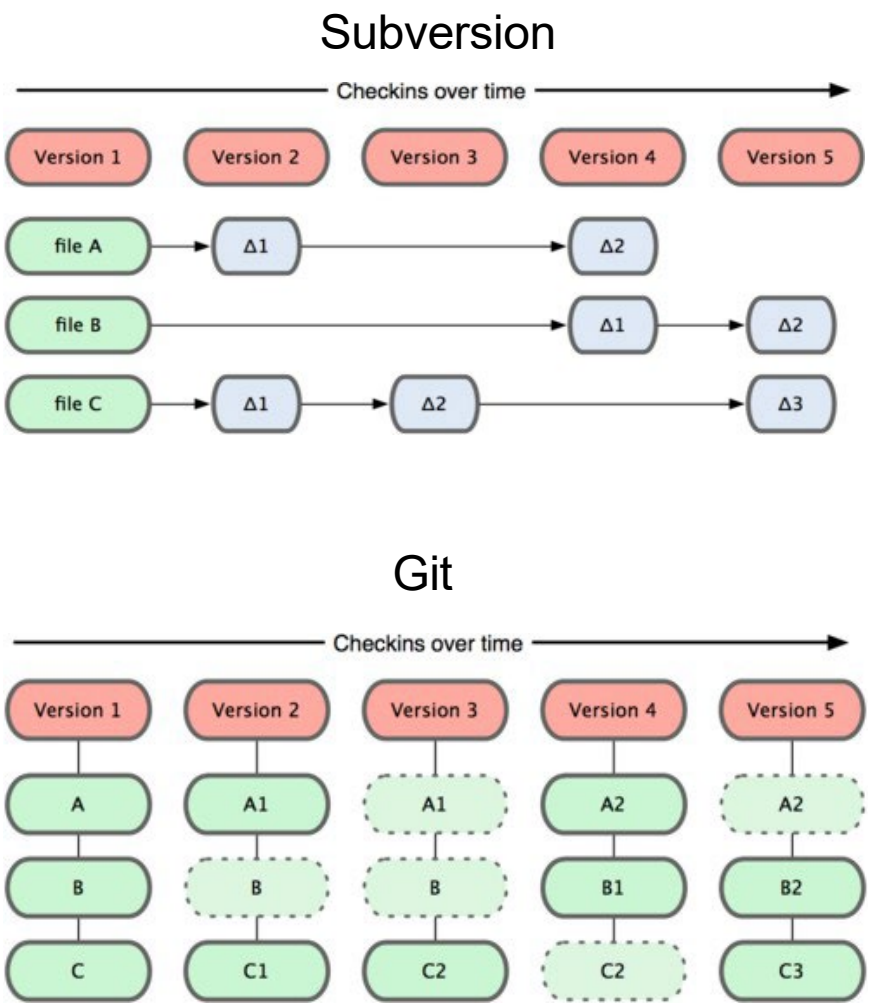


- When you're ready, you can "push" changes back to server

Git snapshots

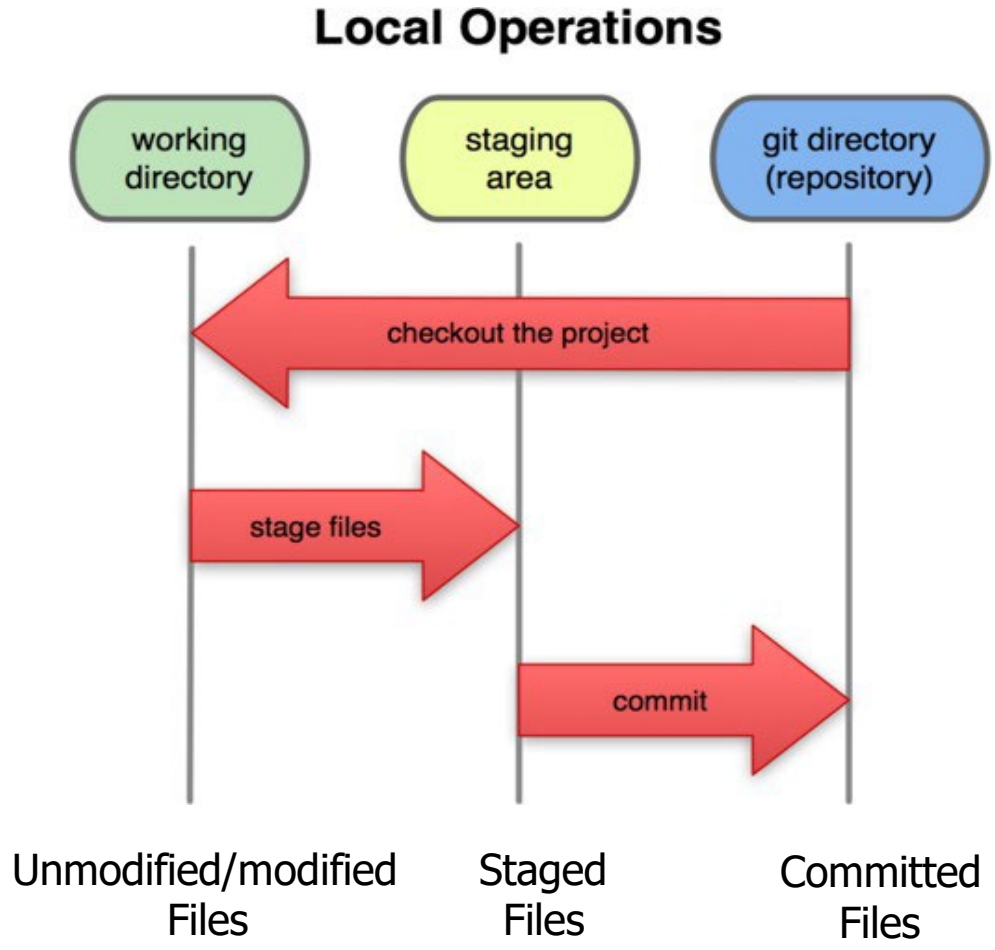
- Centralized VCS like Subversion track version data on each individual file.

- Git keeps "snapshots" of the entire state of the project.
  - Each checkin version of the overall code has a copy of each file in it.
  - Some files change on a given checkin, some do not.
  - More redundancy, but faster.



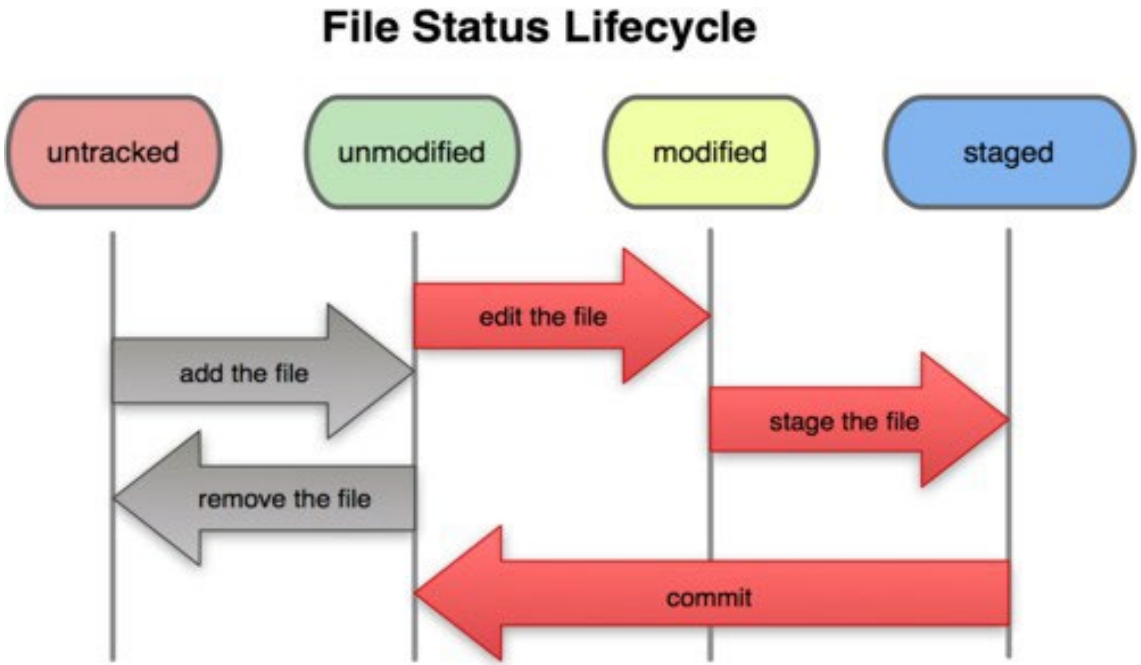
Local git areas

- In your local copy on git, files can be:
  - In your local repo
    - (committed)
  - Checked out and modified, but not yet committed
    - (working copy)
  - Or, in-between, in a **"staging" area**
    - Staged files are ready to be committed.
    - A commit saves a snapshot of all staged state.



Basic Git workflow

- **Modify** files in your working directory.
- **Stage** files, adding snapshots of them to your staging area.
- **Commit**, which takes the files in the staging area and stores that snapshot permanently to your Git directory.



Git commit checksums

- In Subversion each modification to the central repo increments the version # of the overall repo.
  - In Git, each user has their own copy of the repo, and commits changes to their local copy of the repo before pushing to the central server.
  - So Git generates a unique **SHA-1 hash** (40 character string of hex digits) for every commit.
  - Refers to commits by this ID rather than a version number.
  - Often we only see the first 7 characters:
    - `1677b2d Edited first line of readme`
    - `258efa7 Added line to readme`
    - `0e52da7 Initial commit`

Initial Git configuration

- Set the name and email for Git to use when you commit:
  - `git config --global user.name "Bugs Bunny"`
  - `git config --global user.email bugs@gmail.com`
  - You can call `git config -list` to verify these are set.

- Set the editor that is used for writing commit messages:
  - `git config --global core.editor nano`
    - (it is vim by default)

Creating a Git repo

Two common scenarios: (only do one of these)

- To create a new **local Git repo** in your current directory:
  - `git init`
    - This will create a `.git` directory in your current directory.
    - Then you can commit files in that directory into the repo.
  - `git add` *`filename`*
  - `git commit -m "`*`commit message`*`"`

- To **clone a remote repo** to your current directory:
  - `git clone` *`url localDirectoryName`*
    - This will create the given local directory, containing a working copy of the files from the repo, and a `.git` directory (used to hold the staging area and your actual local repo)

Git commands

command	description
`git clone` ***`url [dir]`***	copy a Git repository so you can add to it
`git add` ***`file`***	adds file contents to the staging area
`git commit`	records a snapshot of the staging area
`git status`	view the status of your files in the working directory and staging area
`git diff`	shows diff of what is staged and what is modified but unstaged
`git help` ***`[command]`***	get help info about a particular command
`git pull`	fetch from a remote repo and try to merge into the current branch
`git push`	push your new branches and data to a remote repository
others: `init`, `reset`, `branch`, `checkout`, `merge`, `log`, `tag`	

Add and commit a file

- The first time we ask a file to be tracked, *and every time before we commit a file*, we must add it to the staging area:
  - `git add Hello.java Goodbye.java`
    - Takes a snapshot of these files, adds them to the staging area.
    - In older VCS, "add" means "start tracking this file." In Git, "add" means "add to staging area" so it will be part of the next commit.
- To move staged changes into the repo, we commit:
  - `git commit -m "Fixing bug #22"`
  - All these commands are acting on your local version of repo.

Viewing/undoing changes

- To view status of files in working directory and staging area:
  - `git status` or `git status -s` (short version)
- To see what is modified but unstaged:
  - `git diff`
- To see a list of staged changes:
  - `git diff --cached`
- To see a log of all changes in your local repo:
  - `git log` or `git log --oneline` (shorter version)

```
1677b2d Edited first line of readme
258efa7 Added line to readme
0e52da7 Initial commit
```

  - `git log -5` (to show only the 5 most recent updates), etc.

Interaction w/ remote repo

- **Push** your local changes to the remote repo.
- **Pull** from remote repo to get most recent changes.
  - (fix conflicts if necessary, add/commit them to your local repo)

- To fetch the most recent updates from the remote repo into your local repo, and put them into your working directory:
  - `git pull origin master`

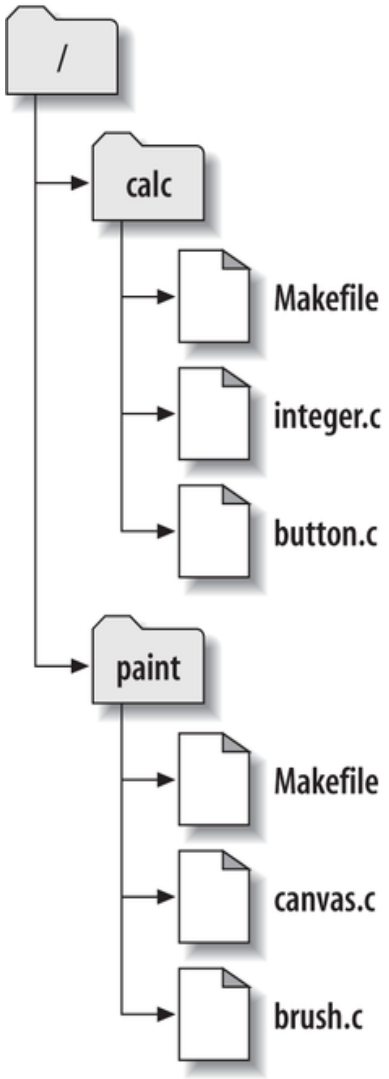
- To put your changes from your local repo in the remote repo:
  - `git push origin master`

GitHub and GitLab

- GitHub.com is a site for online storage of Git repositories.
  - You can create a **remote repo** there and push code to it.
  - Many open source projects use it, such as the Linux kernel.
  - You can get free space for open source projects, or you can pay for private projects.
    - Free private repos for educational use
- GitLab.com is also a site for online storage of Git repositories.
  - But it also allows educational institutions to create their GitLab sites.
  - Example: gitlab.csc.uvic.ca
- *Question:* Do I always have to use GitHub or GitLab to use Git?
  - *Answer:* No! You can use Git locally for your own purposes.
  - Or you or someone else could set up a server to share files.
  - Or you could share a repo with users on the same file system, as long everyone has the needed file permissions).

Basic Concepts: Working Copy

- A **working copy / local repository** is an ordinary directory on your local system
- These are the files you edit
- When changes are at some suitable stage, add & commit your changes in your local repo
  - In practice, this usually means metadata files in the .git directory within the working copy are added or changed (.git directory not shown)
- You may even decide to push committed changes to the remote repo



Basic Concepts: Working Copy

- Obtaining a working copy means either **cloning** an existing repository or performing **init** in some existing directory
  - This is normally done **only once** per working copy
  - All work is done in a (you guessed it!) working copy
  - Working copy is simply a set of directories & files
- Repository access methods differ:
  - direct access via local disk (file:///) **(ugh!)**
  - via ssh:// (when security is very important)
  - http:// or https:// (as used by GitHub and GitLab)
  - original access method for working copy / local repo is stored as metadata in .git subdirectories in that working copy)

Please do not use **git init** for this course!

Basic Concept: Working Copy

- Example: get working copy of the "calc" project from a (made up!) git.example.com

```
$ git clone ssh://stewie@git.example.com/repos/calc
Cloning into 'calc'...
<password for stewie>
remote: Counting objects: 37, done.
remote: Compressing objects: 100% (31/31), done.
remote: Total 37 (delta 5), reused 0 (delta 0)
Receiving objects: 100% (37/37), done.
Resolving deltas: 100% (5/5), done.
Checking connectivity... done.

$ cd calc
$ git ls-files  # could even use `tree` here...
Makefile
button.c
integer.c

$
```

Basic Concepts: Commit

- Suppose you wish to keep track of some changes to **button.c**
  - You edit the file using your normal workflow
  - Time and date on edited file will be more recent than time and date of file in local repo
  - Changes are recorded by committing your changed file to the repository
- We first stage our changes (via **add**)…
- … and then make a "permanent" record of the change (**commit**)

```
$ pwd
calc

$ git add button.c
$ git commit –m "Fixed the geometry of button for v3 of library"
[master 12788ce] Fixed the geometry of button for v3 of library
 1 file changed, 1 deletion(-)
$
```

Basic Concepts: Commit

- Note that commits are **always** to our local repo
  - i.e., take place within our working/local copy
- We can have a sequence of **many** such commits as we work through sets of changes
- **To have the results of the commits available to others...**
  - **... or to ourselves on a different machine ...**
  - **... we must push them to a remote repository**
- Observations:
  - Commits can be quite frequent (if needed by our workflow)
  - **Pushes are much less frequent** (with such a push perhaps reflecting our work has reached some suitable state and is ready for other team members to pull).
  - Example: writing and completing each assignment will require several commits
  - Example: submitting an assignment will require at least one push

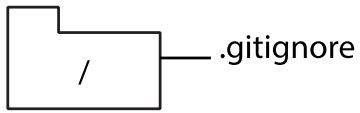
Basic Concepts: Commit

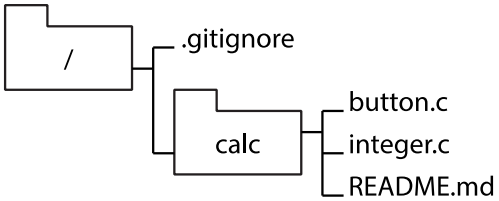
- Each **commit** results **in the creation of a new snapshot of the contents** in our working copy / local repository
- Snapshots are kept in chronological order
  - **git log** produces a list of the snapshots
  - Default log output order is reverse chronological (i.e., most recent commit/snapshot is listed first)
- **git status** reports the relationship amongst files in our working directory with what within git's local repository
  - More precisely, "status" tells us what has changed in our working directory…
  - … and therefore what may need to be "add"ed and "commit"ed to the local repository

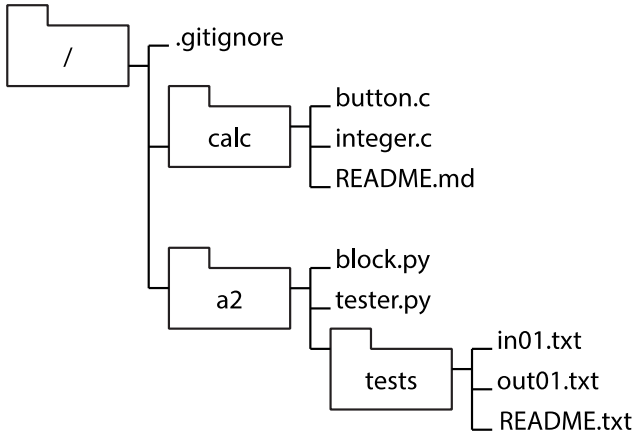
Basic Concepts: Commit

time

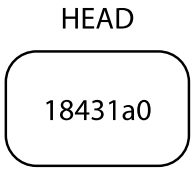
files

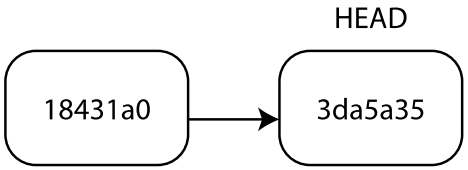






snapshots





note

initial commit

commit after adding "calc" with its files

commit after adding "a2" with its files and subdirectory

(steps leading to third commit)

```
$ pwd
/home/stewie/project

$ git add *  # Let git know what files are to be committed; recursive on directories

$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   a2/block.py
    new file:   a2/tester.py
    new file:   a2/tests/README.txt
    new file:   a2/tests/in01.txt
    new file:   a2/tests/out01.txt

$ git commit -m "Now the A#2 directory (a2) is in place"  # Note message
[master 85687aa] Now the A#2 directory (a2) is in place
 5 files changed, 6 insertions(+)
 create mode 100644 a2/block.py
 create mode 100644 a2/tester.py
 create mode 100644 a2/tests/README.txt
 create mode 100644 a2/tests/in01.txt
 create mode 100644 a2/tests/out01.txt
```

Basic Concepts: Commit

```
$ git log
commit 85687aa056e299897153a3125c1826f64581bdc5
Author: Stewie Griffin <stewie@uvic.ca>
Date:   Thu Apr 15 10:05:54 2018 -0700

    Now the A#2 directory (a2) is in place

commit 3da5a353956c320fbe8e585cd692b173e44b06c1
Author: Stewie Griffin <stewie@uvic.ca>
Date:   Thu Apr 15 10:01:57 2018 -0700

    Added calc and some files

commit 18431a0b85f0645c98e4cceb311074594a19a38d
Author: Stewie Griffin <stewie@uvic.ca>
Date:   Thu Apr 15 09:38:34 2018 -0700

    Initial commit (just .gitignore for now)
$
```

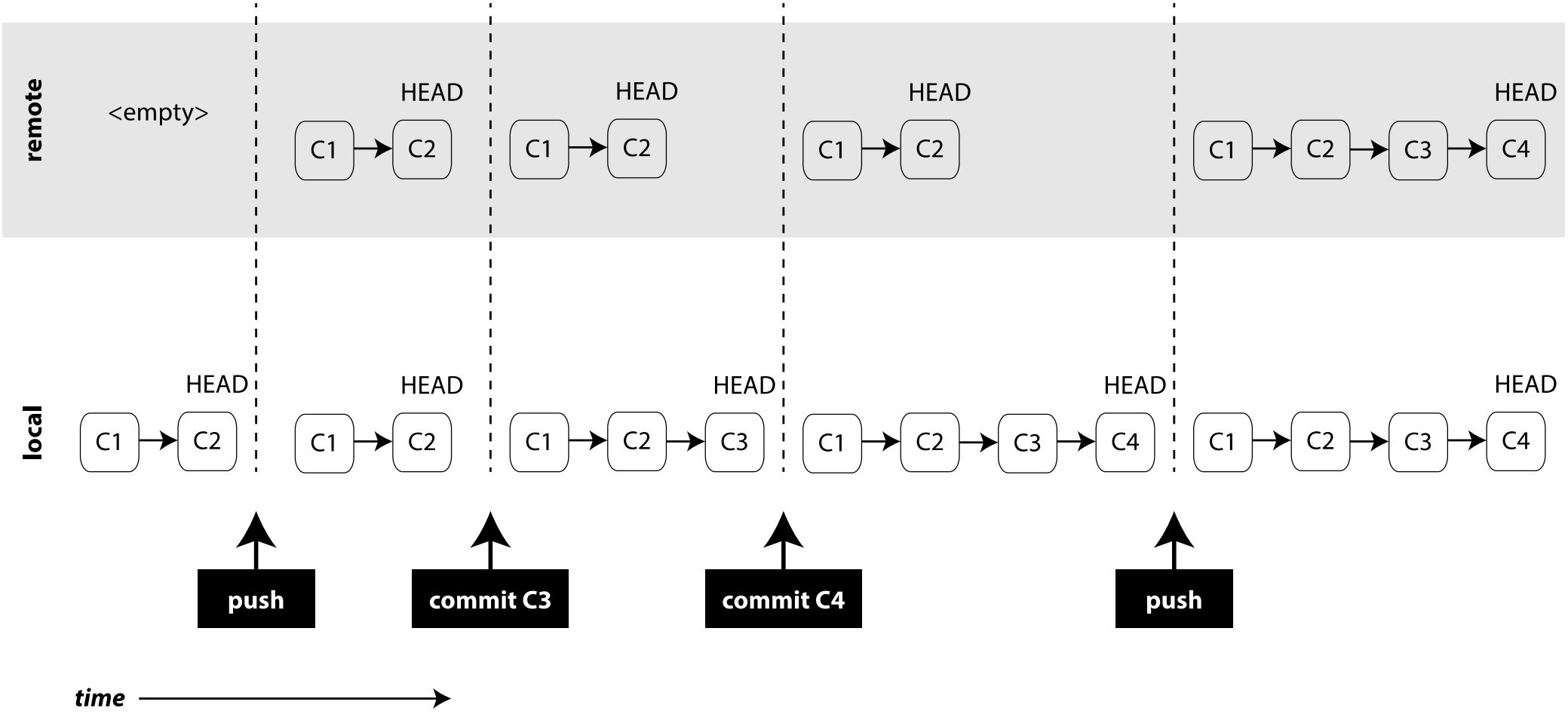
Basic Concepts: Commit

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working directory clean

$ git push
<verbiage>
<password for stewie>
Counting objects: 13, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (10/10), done.
Writing objects: 100% (12/12), 1.11 KiB | 0 bytes/s, done.
Total 12 (delta 0), reused 0 (delta 0)
To ssh://stewie@git.example.com/project
   18431a0..85687aa  master -> master

$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean

$
```

commit != push

- `git is different from many other kinds of VCSes`
  - commits are made to the local repo, **not the remote**
  - making these changes available to others means transferring data from the local repo to the remote repo (i.e., a push)
- `git separates` **the tracking of file/directory changes** `from` **their storage on remote servers/repositories**
  - this is very different from Subversion, Perforce, etc.
  - … and can seem a bit confusing on first encounter

commit != push


remote
local
<empty>
HEAD
C1
C2
C3
C4
push
commit C3
commit C4
push
time

Basic Concepts: Update

- What if Meg starts working on the project after someone else's commit?
  - Assume she made a working copy of Stewie's repo some time ago
  - (Also assume here she has read & write privileges on Stewie's remote.)
  - Let's also assume she was not working on button.c in calc
- She can ask git to bring her working copy "up to date"
  - git will only update files for which there are changes on the remote
  - Principle: Make sure to update often if working with a group on a project that uses a repository!

```
$ pwd
/home/meg/calc

$ git pull
<password for meg>
remote: Counting objects: 7, done.
remote: Compressing objects: 100% (4/4), done.
<... snip ...>
Fast-forward
 calc/button.c | 1 +
 1 file changed, 1 insertion(+)

$
```

Basic Concepts: Update

- "git pull" is actually two commands together:
  - "git fetch" followed by "git merge"

```
$ pwd
/home/meg/calc

$ git fetch
<password for meg>
remote: Counting objects: 7, done.
<snip>

$ git log --name-status
commit eb6c8a6ffb2e2a70a89e4a89db8d62b5a22eccd4
Author: Stewie Griffin <stewie@uvic.ca>
Date:   Thu Apr 15 10:27:23 2018 -0700
    Added headers so buttons can be beveled

M       calc/button.c
```

Basic Concepts: Update

```
$ pwd
/home/meg/calc

$ git merge origin/master
Updating 85687aa..eb6c8a6
Fast-forward
 calc/button.c | 3 +++
 1 file changed, 3 insertions(+)
$
```

- Therefore "git pull" is the same as the following two commands in succession
  - git fetch
  - git merge origin/master
- We used "git log --name-status" to obtain the names of files that are different from our working copy and the remote repo