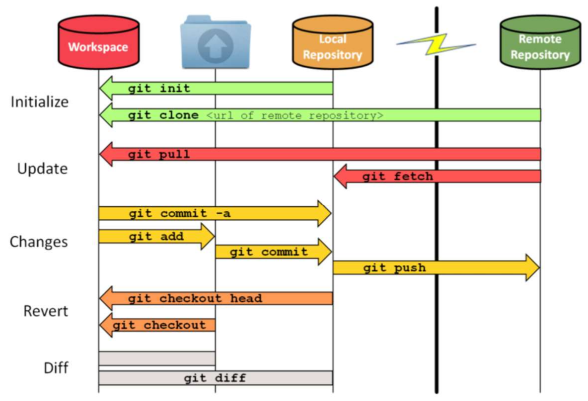
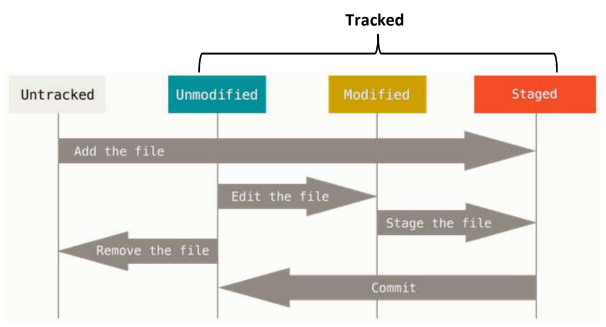
CS35L – 5
Week 9 Lec 1

Terms used

- **Repository**
  - Files and folder related to the software code
  - Full History of the software
- **Working copy**
  - Copy of software's files in the repository
- **Check-out**
  - To create a working copy of the repository
- **Check-in / Commit**
  - Write the changes made in the working copy to the repository
  - Commits are recorded by the VCS



Git File Status Lifecycle



Terms used

- Head
  - Refers to a commit object
  - There can be many heads in a repository
- HEAD
  - Refers to the currently active head
- Detached HEAD
  - If a commit is not pointed to by a branch
  - This is okay if you want to just take a look at the code and if you don't commit any new changes
  - If the new commits have to be preserved then a new branch has to be created
    - git checkout v3.0 -b BranchVersion3.1
- Branch
  - Refers to a head and its entire set of ancestor commits
- Master
  - Default branch

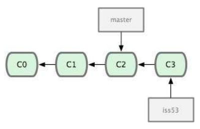


Image Source: git-scm.com

What Is a Branch?

- A pointer to one of the commits in the repo (head) + all ancestor commits
- When you first create a repo, are there any branches?
  - Default branch named ‘master’
- The default master branch
  - points to last commit made
  - moves forward automatically, every time you commit

First Git Repository

```
$ mkdir gitroot

$ cd gitroot

$ git init
```
creates an empty git repo (.git directory with all necessary subdirectories)

```
$ echo “Hello World” > hello.txt

$ git add .
```
Adds content to the index
Must be run prior to a commit

```
$ git commit –m ‘Check in number one’
```

Git Example

- Project
  - games: pacman.c, pacman.h, README
- Create repository to track new project
  - `$ git init` (creates .git dir w/ all necessary repo files)
- Is the project tracked?
  - No, need to add files and do an initial commit
    - `$ git add` pacman.c pacman.h README
    - `$ git commit –m` “initial commit of my project”

Working With Git

```
$ echo "I love Git" >> hello.txt
```

```
$ git status
```
Shows list of modified files
hello.txt

```
$ git diff
```
Shows changes we made compared to index

```
$ git add hello.txt
```

```
$ git diff
```
No changes shown as diff compares to the index

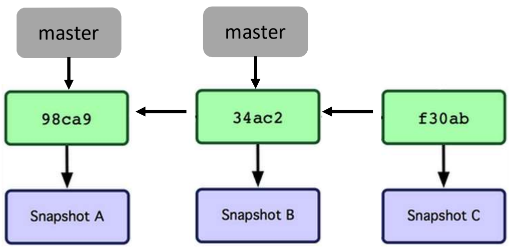
```
$ git diff HEAD
```
Now we can see changes in working version

```
$ git commit -m "Second commit"
```

Git commands

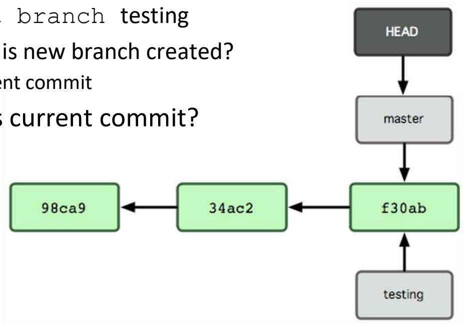
- Repository creation
  - $ git init (Start a new repository)
  - $ git clone (Create a copy of an exisiting repository)
- Branching
  - $ git checkout <tag/commit> -b <new_branch_name> (creates a new branch)
- Commits
  - $ git add (Stage modified/new files)
  - $ git commit (check-in the changes to the repository)
- Getting info
  - $ git status (Shows modified files, new files, etc)
  - $ git diff (compares working copy with staged files)
  - $ git log (Shows history of commits)
  - $ git show (Show a certain object in the repository)
- Getting help
  - $ git help

Where Is Master?



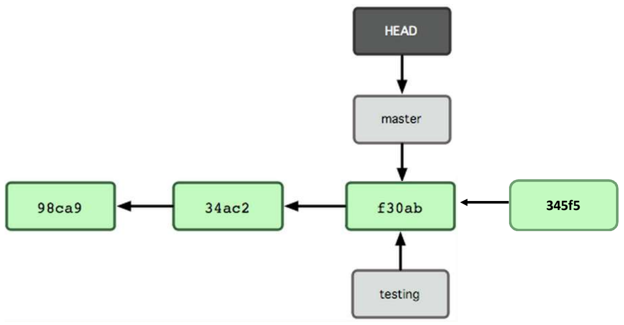
New Branch

- Creating a new branch = creating new pointer
  - $ `git branch` testing
  - Where is new branch created?
    - Current commit
- Where is current commit?
  - HEAD



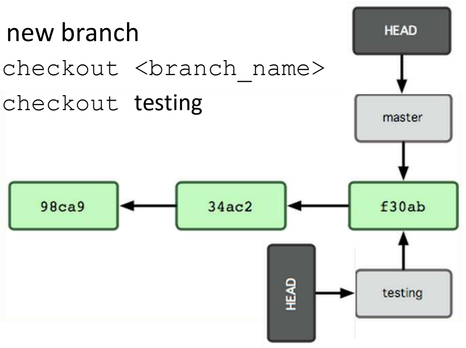
New Commit

- What happens if we make another commit?

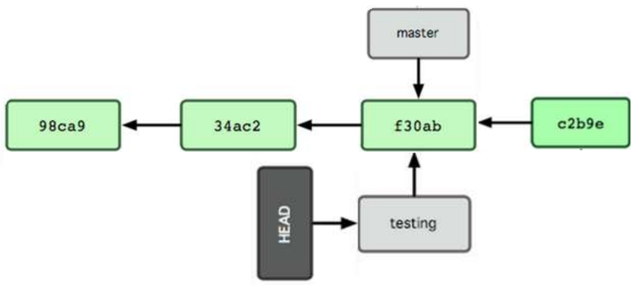


Switching to New Branch

- Check out new branch
  - `$ git checkout <branch_name>`
  - `$ git checkout testing`



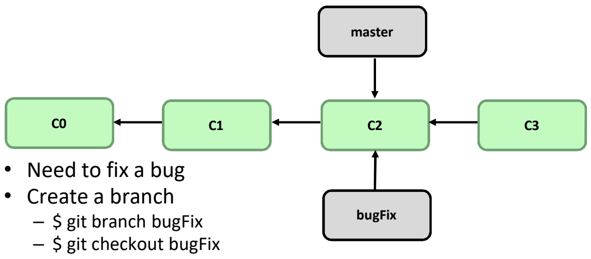
Commit After Switch



Why Branching?

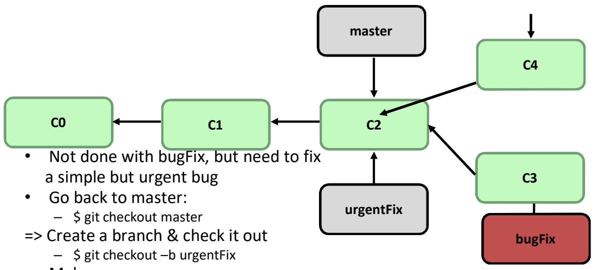
- Experiment with code without affecting main branch
- Separate projects that once had a common code base
- 2 versions of the project

Merging



- Need to fix a bug
- Create a branch
  - $ git branch bugFix
  - $ git checkout bugFix
- Make some progress
  - Make a commit

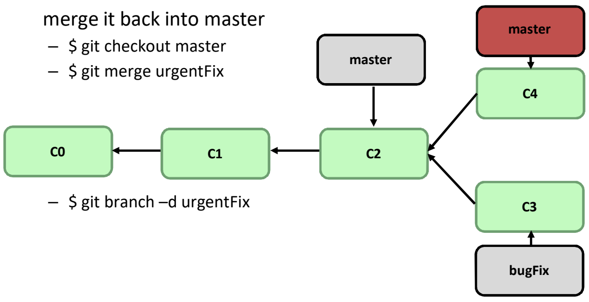
Merging



- Not done with bugFix, but need to fix a simple but urgent bug
- Go back to master:
  - $ git checkout master

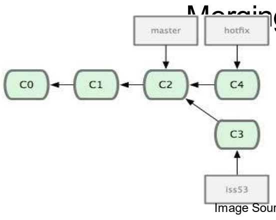
=> Create a branch & check it out
  - $ git checkout –b urgentFix
- Make some progress
  - Make a commit

Merging



- When confident about fix, we can merge it back into master
  - $ git checkout master
  - $ git merge urgentFix
  - $ git branch –d urgentFix

Merging



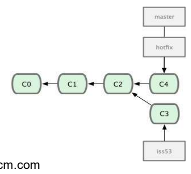


Image Source: git-scm.com

- Merging hotfix branch into master
  - $ git checkout master
  - $ git merge hotfix
  - Git tries to merge automatically
    - Simple if its a forward merge
    - Otherwise, you have to manually resolve conflicts

Merging

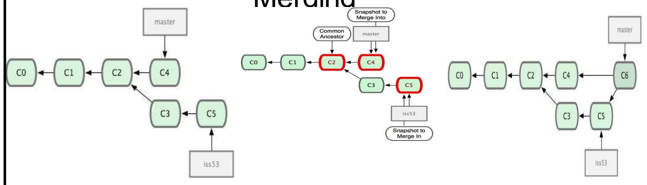


Image Source: git-scm.com

- Merge iss53 into master
- Git tries to merge automatically by looking at the changes since the common ancestor commit
- Manually merge using 3-way merge or 2-way merge
  - Merge conflicts - Same part of the file was changed differently

Merging

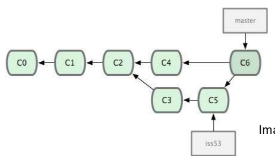
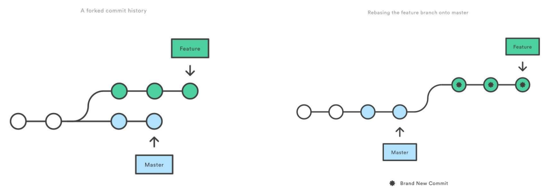


Image Source: git-scm.com

- Refer to multiple parents
  - $ git show hash
  - $ git show hash^2 (shows second parent)
- HEAD^^ == HEAD~2

Git Rebase

- Rewrites commit history.
- Loses context
- Never use this on public branches!
- How to rebase?
  - $ git checkout feature
  - $ git rebase master



Source - https://www.atlassian.com/git/tutorials

More Git Commands

- Reverting
  - $ git checkout HEAD main.cpp
    - Gets the HEAD revision for the working copy
  - $ git checkout -- main.cpp
    - Reverts changes in the working directory
  - $ git revert
    - Reverting commits (this creates new commits)
- Cleaning up untracked files
  - $ git clean
- Tagging
  - Human readable pointers to specific commits
  - $ git tag -a v1.0 –m 'Version 1.0'
    - This will name the HEAD commit as v1.0

Assignment 9

- Installing Git
  - Ubuntu: $ sudo apt-get install git
  - SEASnet
    - Git is installed in /usr/local/cs/bin
    - Add it to PATH variable or use whole path
      - $ export PATH=/usr/local/cs/bin:$PATH
- Make a directory 'gitroot' and get a copy of the Diffutils Git repository
  - $ mkdir gitroot
  - $ cd gitroot
  - $ git clone git://git.savannah.gnu.org/diffutils.git
- Follow steps in lab and use man git to find commands