Git
1

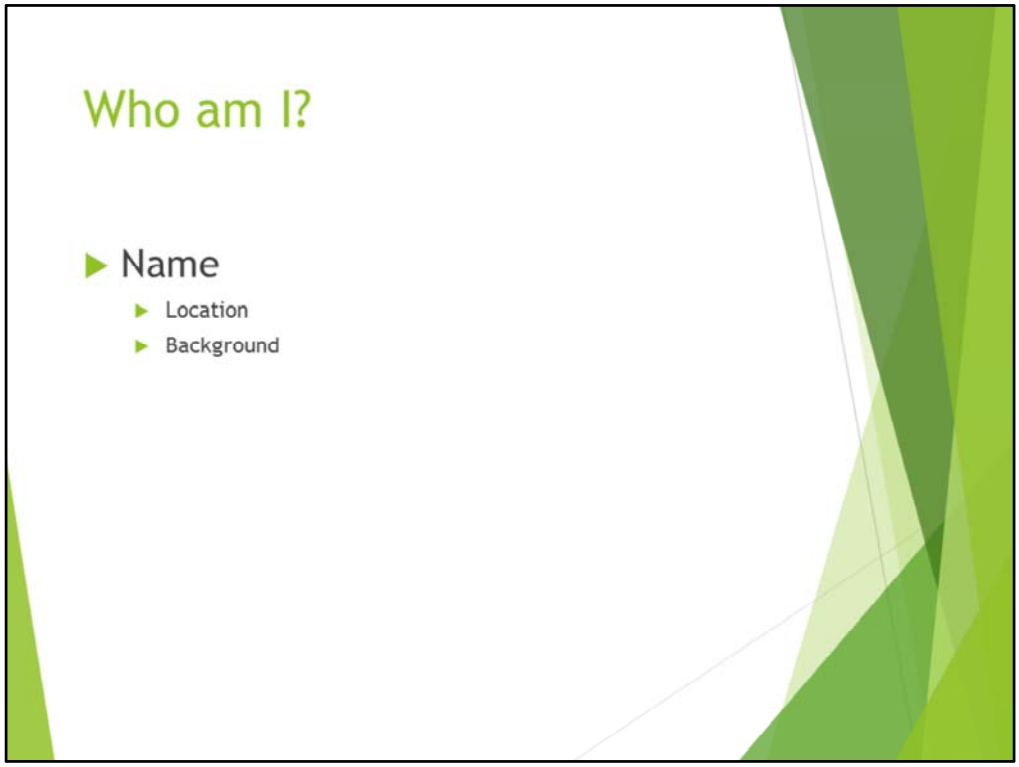
Outline

- History of Git
- Distributed V.S Centralized Version Control
- Getting started
- Branching and Merging
- Working with remote
- Summary

2

Introductions

Who are you?
Introductions
Name
What do you do for money?
How long
Favorite food
Technology Stack
Servers
Languages
Middleware
Persistence


Who am I?
Name
Location
Background

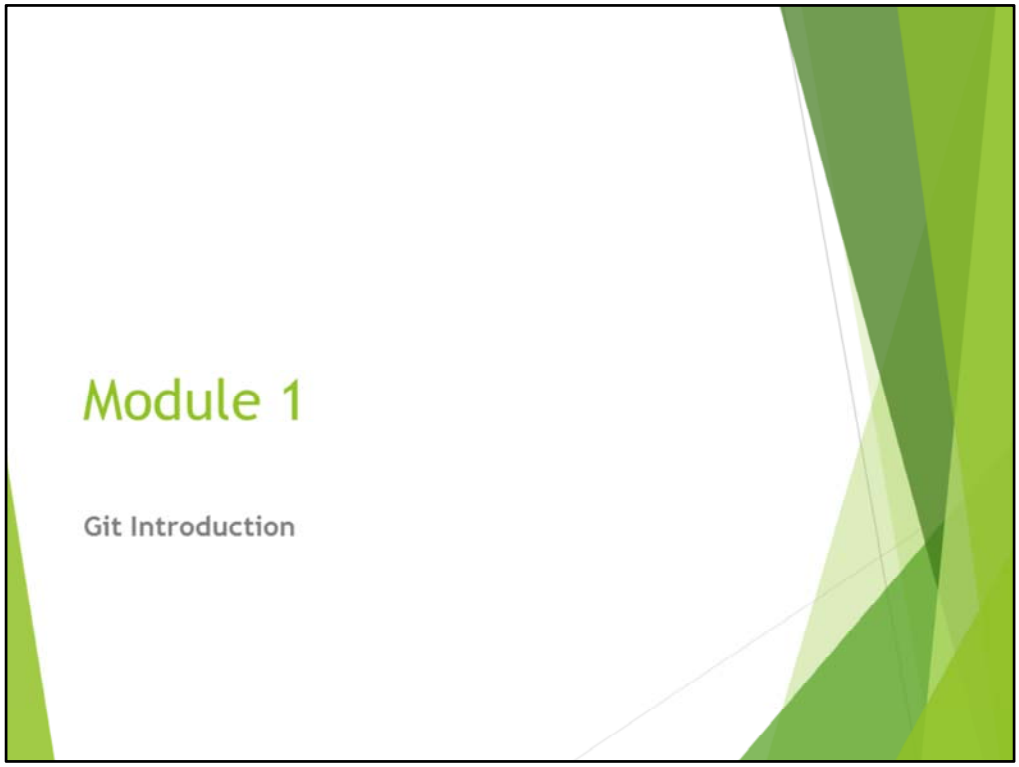
Logistics

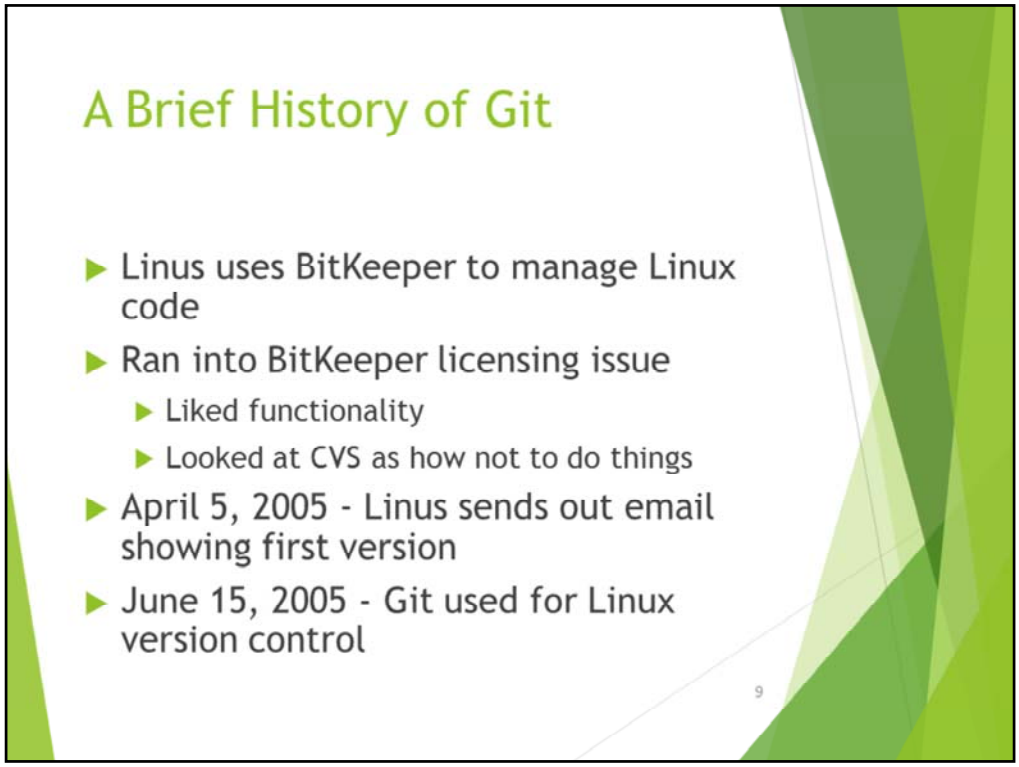
Logistics

- Stop/Start Time?

- Break every hour

- Ask questions anytime


Module 1
Git Introduction


A Brief History of Git
Linus uses BitKeeper to manage Linux code
Ran into BitKeeper licensing issue
Liked functionality
Looked at CVS as how not to do things
April 5, 2005 - Linus sends out email showing first version
June 15, 2005 - Git used for Linux version control
9

Git is Not an SCM

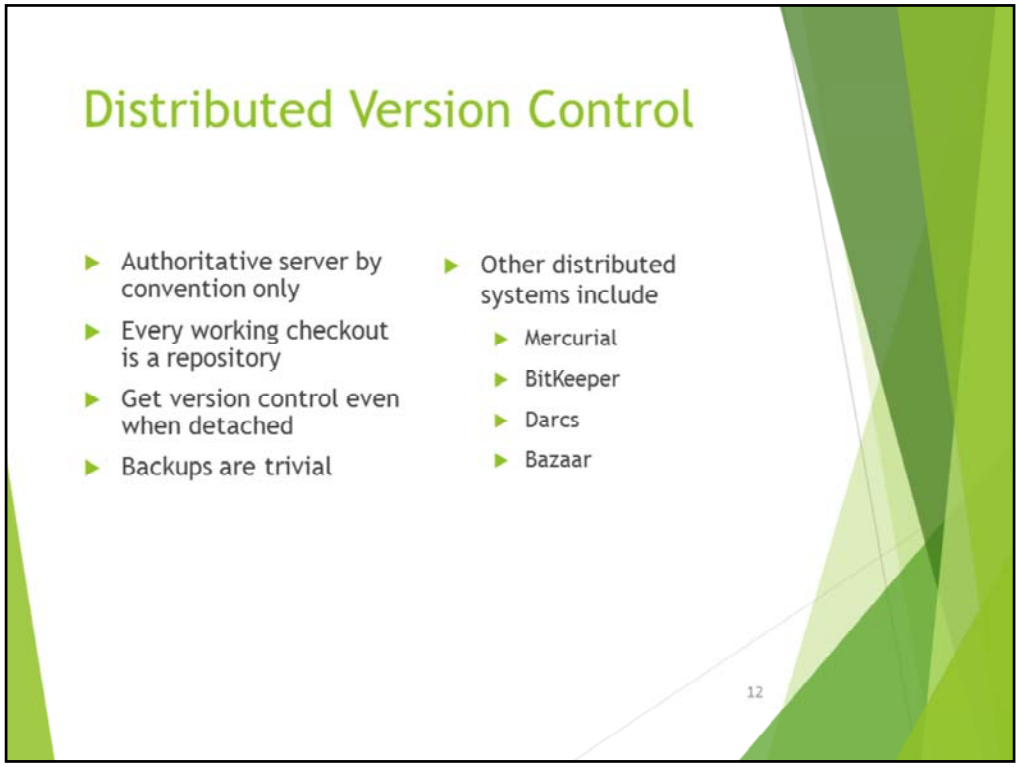
Never mind merging. It's not an SCM, it's a distribution and archival mechanism. I bet you could make a reasonable SCM on top of it, though. Another way of looking at it is to say that it's really a content-addressable filesystem, used to track directory trees.

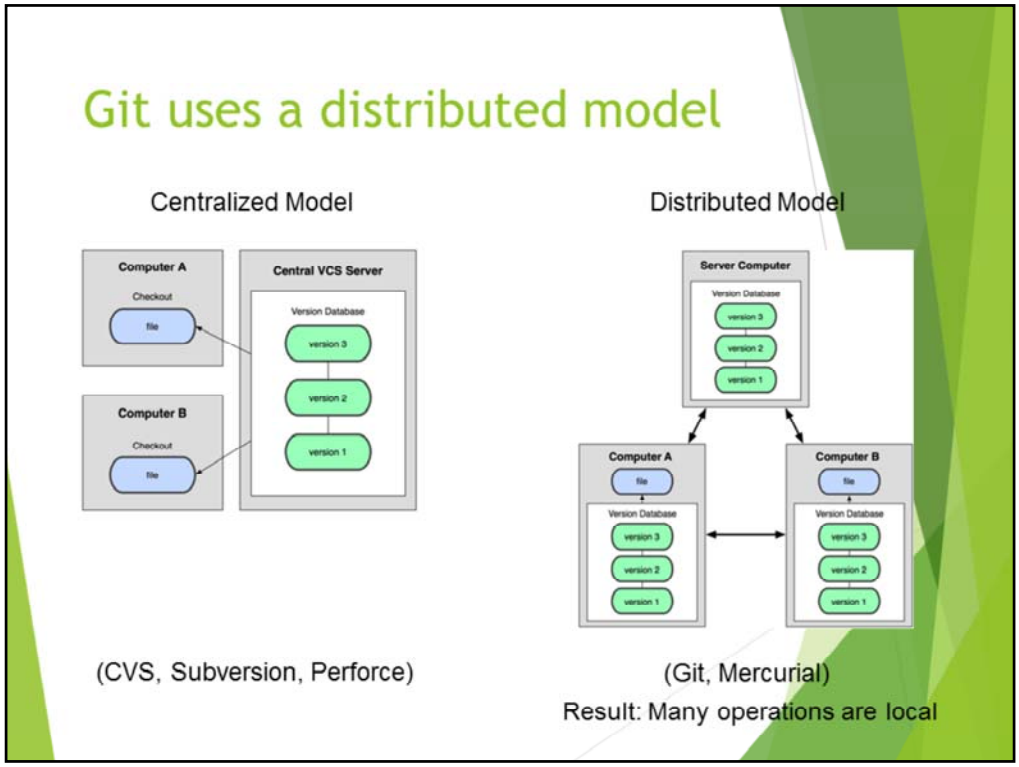
Linus Torvalds, 7 Apr 2005

http://lkml.org/lkml/2005/4/8/9

10

Centralized Version Control
Traditional version control system
Server with database
Clients have a working version
Examples
CVS
Subversion
Visual Source Safe
Challenges
Multi-developer conflicts
Client/server communication
11


Distributed Version Control
Authoritative server by convention only
Every working checkout is a repository
Get version control even when detached
Backups are trivial
Other distributed systems include
Mercurial
BitKeeper
Darcs
Bazaar
12

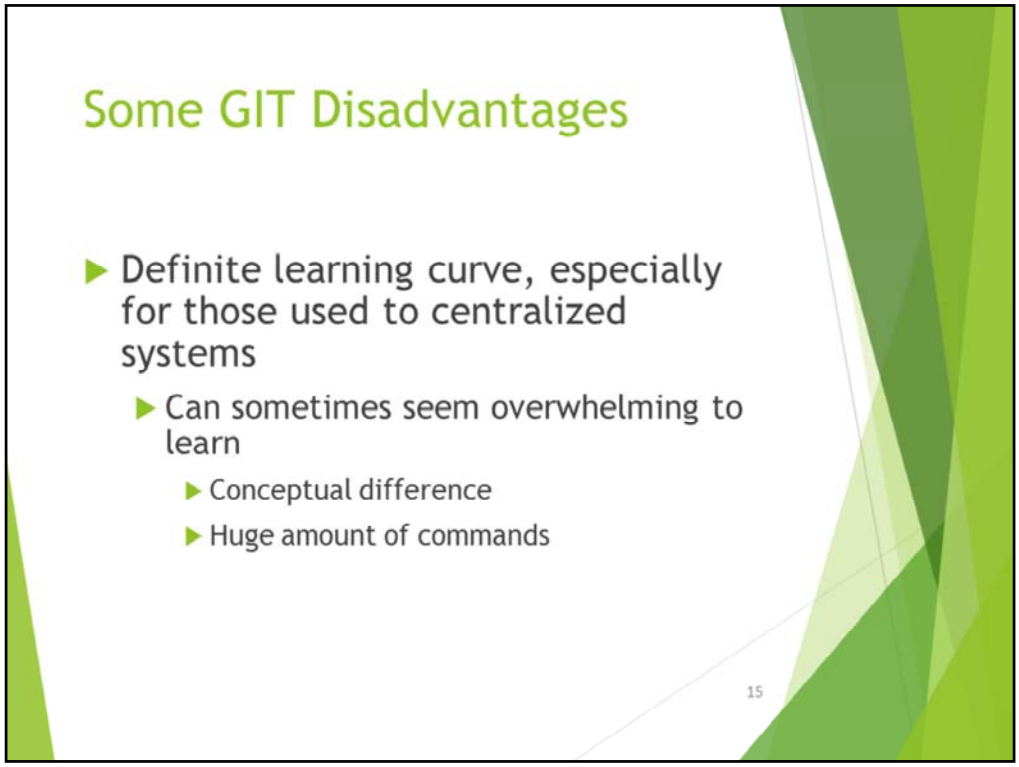


http://git-scm.com/book/en/Getting-Started-About-Version-Control

Git Advantages

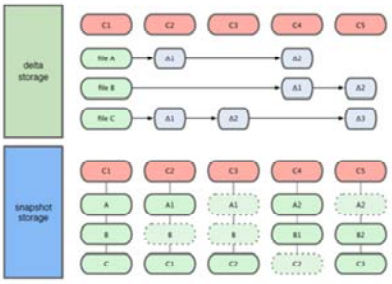
- Resilience
  - No one repository has more data than any other
- Speed
  - Very fast operations compared to other VCS (I'm looking at you CVS and Subversion)
- Space
  - Compression can be done across repository not just per file
  - Minimizes local size as well as push/pull data transfers
- Simplicity
  - Object model is very simple
- Large userbase with robust tools

14


Some GIT Disadvantages
Definite learning curve, especially for those used to centralized systems
Can sometimes seem overwhelming to learn
Conceptual difference
Huge amount of commands
15

Getting Started

- Users have their own copies of the repository
  - The full repository could get overwhelming large so the differences are maintained
- Git uses "snapshot storage"



16

Git uses checksums

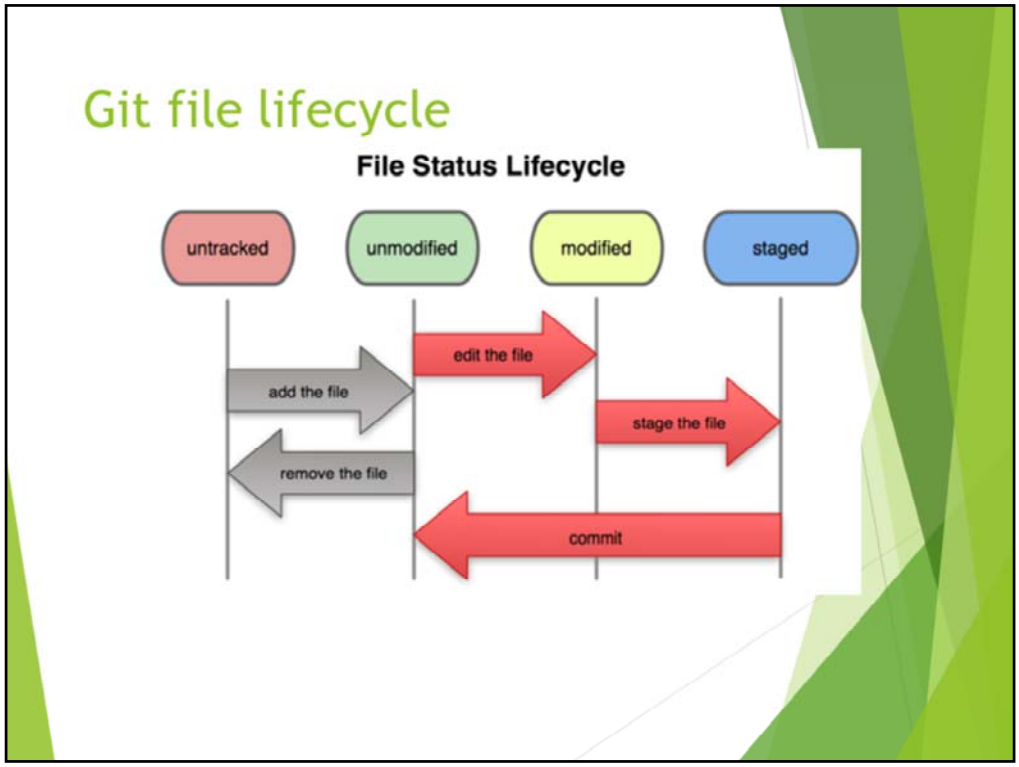
- In Subversion each modification to the central repo incremented the version # of the overall repo.
- How will this numbering scheme work **when each user has their own copy of the repo**, and commits changes to their local copy of the repo before pushing to the central server?????
- Git generates a unique SHA-1 hash - 40 character string of hex digits, for every commit. Refer to commits by this ID rather than a version number. Often we only see the first 7 characters:

  1677b2d Edited first line of readme

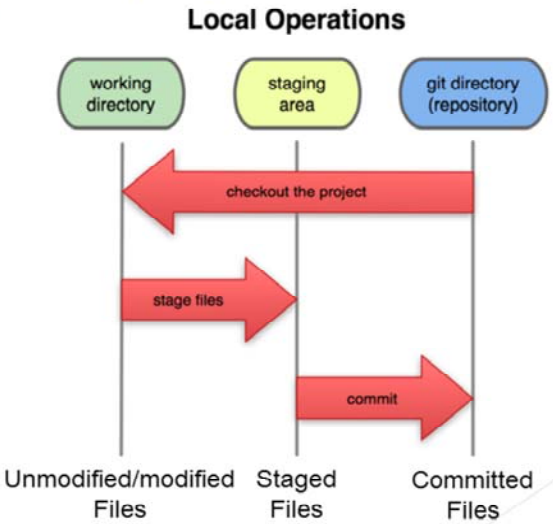
  258efa7 Added line to readme

  0e52da7 Initial commit

Git file lifecycle



A Local Git project has three areas



Note: working directory sometimes called the "working tree", staging area sometimes called the "index".

Basic Git commands

command	description
`git clone` ***`url [dir]`***	copy a git repository so you can add to it
`git add` ***`files`***	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`	

Get ready to use Git!

- 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.
- These will be set globally for all Git projects you work with.
- You can also set variables on a project-only basis by not using the **`--global`** flag.
- You can also set the editor that is used for writing commit messages:
  `$ git config --global core.editor emacs` (it is vim by default)

Create a local copy of a repo

Two common scenarios: (only do one of these)

a) To clone an already existing repo to your current directory:

```
$ git clone <url> [local dir name]
```

This will create a directory named *local dir name*, containing a working copy of the files from the repo, and a **.git** directory (used to hold the staging area and your actual repo)

Create a local copy of a repo

Two common scenarios: (only do one of these)

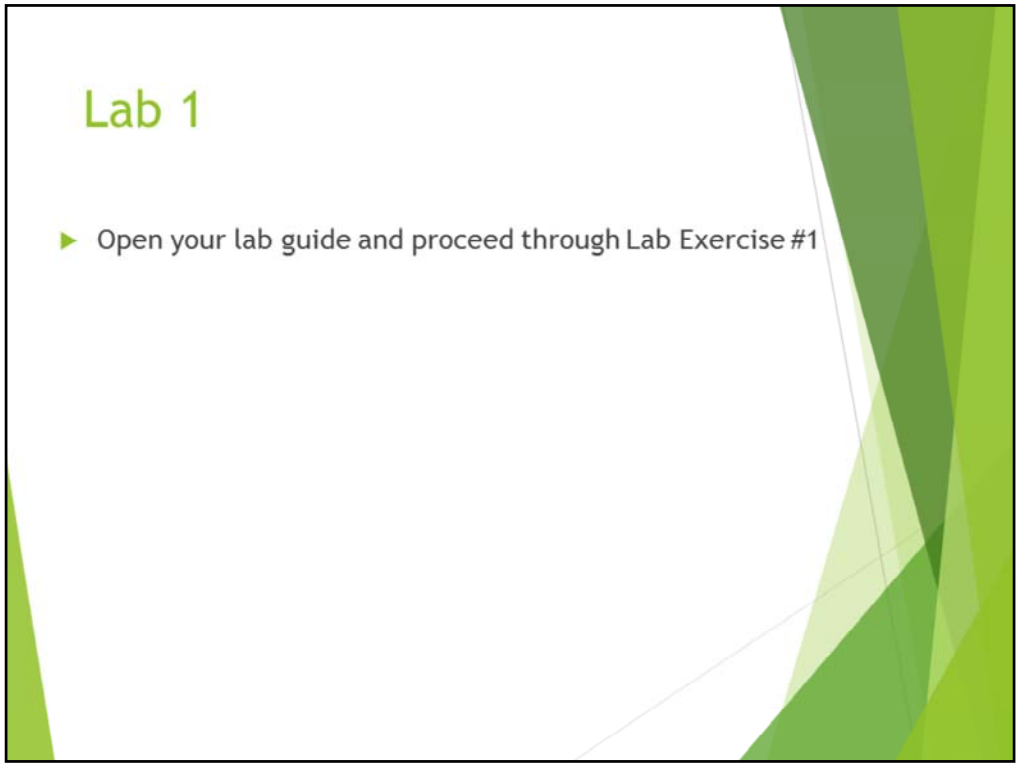
b) To create a Git repo in your current directory:

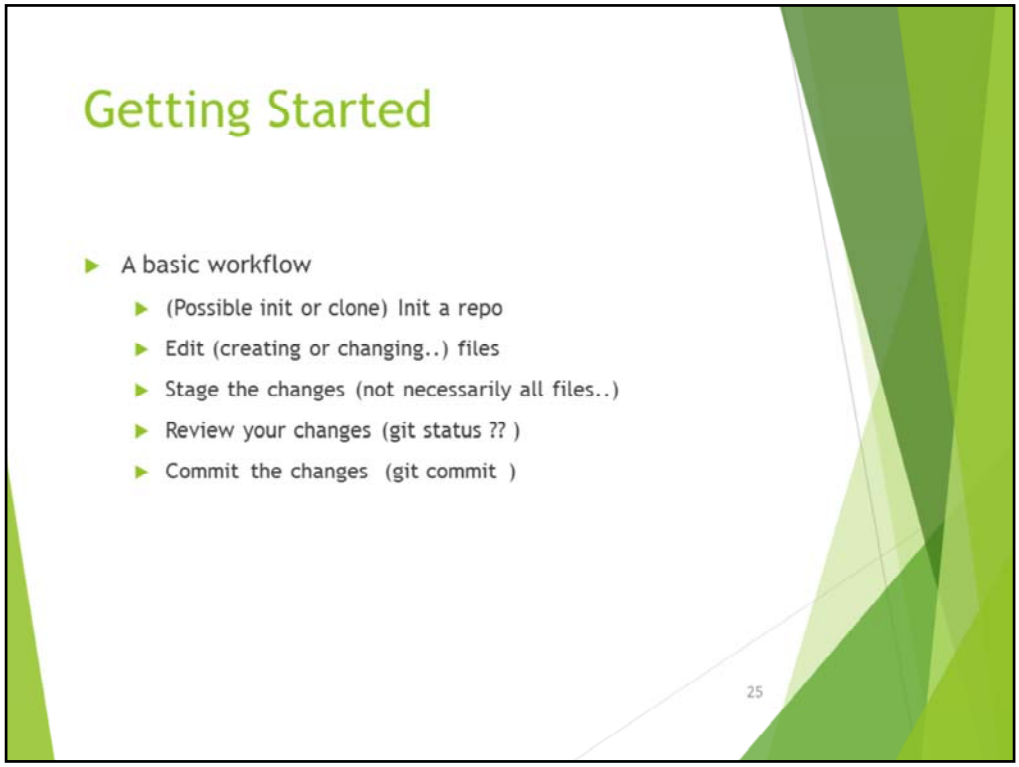
```
$ git init
```

This will create a .git directory in your current directory.

Then you can commit files in that directory into the repo:

```
$ git add file1.java
$ git commit -m "initial project version"
```


Lab 1
Open your lab guide and proceed through Lab Exercise #1


Getting Started
A basic workflow
(Possible init or clone) Init a repo
Edit (creating or changing..) files
Stage the changes (not necessarily all files..)
Review your changes (git status ??)
Commit the changes (git commit)
25

Getting Started

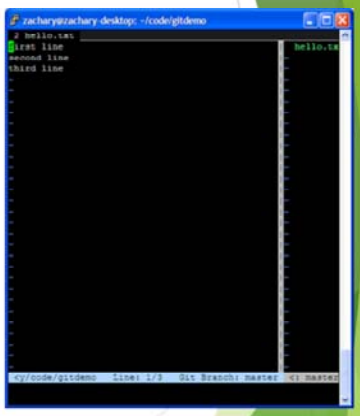
- Init a repository
- Git init

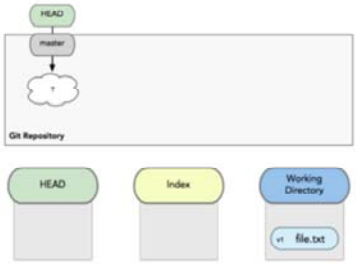
```
zachary@zachary-desktop:~/code/gitdemo$ git init
Initialized empty Git repository in /home/zachary/code/gitdemo/.git/
```

```
zachary@zachary-desktop:~/code/gitdemo$ ls -l .git/
total 32
drwxr-xr-x 2 zachary zachary 4096 2011-08-28 14:51 branches
-rw-r--r-- 1 zachary zachary   92 2011-08-28 14:51 config
-rw-r--r-- 1 zachary zachary   73 2011-08-28 14:51 description
-rw-r--r-- 1 zachary zachary   23 2011-08-28 14:51 HEAD
drwxr-xr-x 2 zachary zachary 4096 2011-08-28 14:51 hooks
drwxr-xr-x 2 zachary zachary 4096 2011-08-28 14:51 info
drwxr-xr-x 4 zachary zachary 4096 2011-08-28 14:51 objects
drwxr-xr-x 4 zachary zachary 4096 2011-08-28 14:51 refs
```

Getting Started

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes
- Use your favorite editor

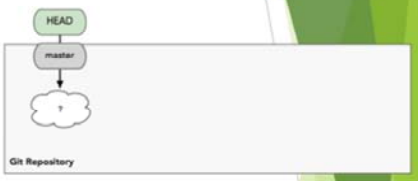




27

Getting Started

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes
- Git add filename



git add

```
zachary@zachary-desktop:~/code/gitdemo$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   hello.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```

28

Getting Started

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes
- Git status

```
zachary@zachary-desktop:~/code/gitdemo$ git add hello.txt
zachary@zachary-desktop:~/code/gitdemo$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   hello.txt
#
```

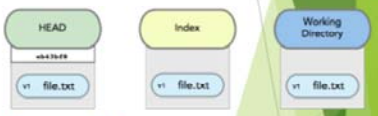
git add

29

Getting Started

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes

- git commit
- git commit 'My Message'



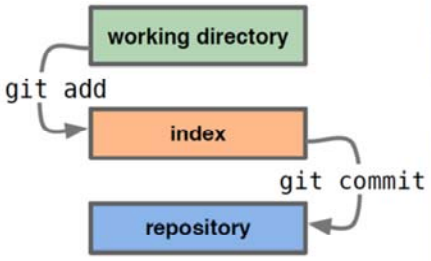
git commit

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   hello.txt
#
```

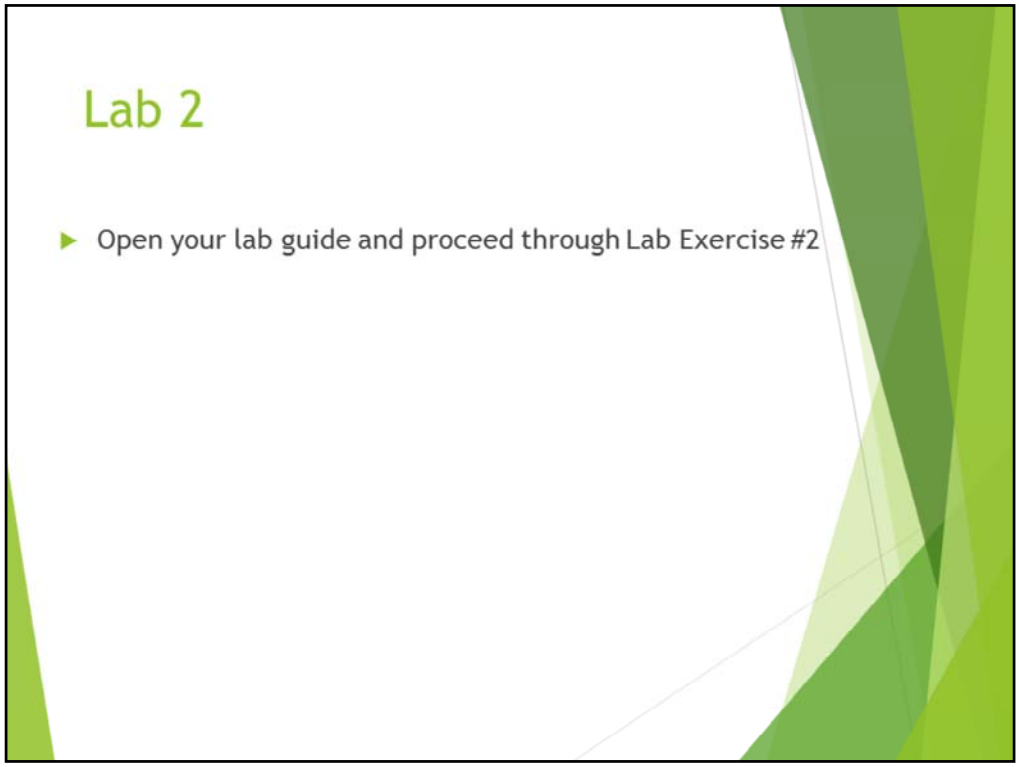
30

Getting Started

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes



31


Lab 2
Open your lab guide and proceed through Lab Exercise #2

Getting Started

- View changes
- Git diff
  - Show the difference between working directory and staged
- Git diff --cached
  - Show the difference between staged and the HEAD

- View history
- Git log

```
zachary@zachary-desktop:~/code/gitdemo$ git log
commit efb3aeae66029474e28273536a8f52969d705d04
Author: Zachary Ling <zacling@gmail.com>
Date:   Sun Aug 28 15:02:08 2011 +0800

    Add second line

commit 453914143eae3fc5a57b9504343e2595365a7357
Author: Zachary Ling <zacling@gmail.com>
Date:   Sun Aug 28 14:59:13 2011 +0800

    Initial commit
```

33

Getting Started

- Revert changes (Get back to a previous version)
  - Git checkout commit_hash

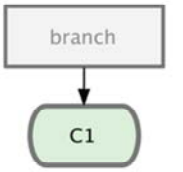
```
zachary@zachary-desktop:~/code/gitdemo$ git log
commit efb3aeae66029474e28273536a8f52969d705d04
Author: Zachary Ling <zacling@gmail.com>
Date:   Sun Aug 28 15:02:08 2011 +0800

    Add second line

commit 453914143eae3fc5a57b9504343e2595365a7357
Author: Zachary Ling <zacling@gmail.com>
Date:   Sun Aug 28 14:59:13 2011 +0800

    Initial commit
zachary@zachary-desktop:~/code/gitdemo$ git checkout 4539
Note: checking out '4539'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at 4539141... Initial commit
```

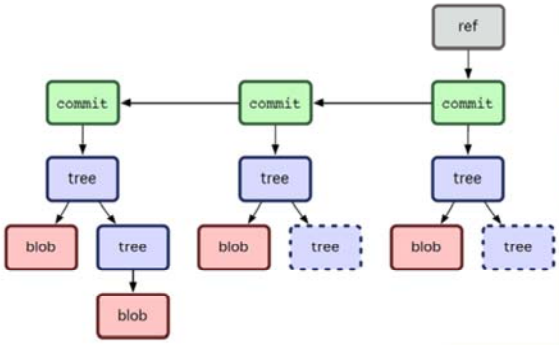
34

Branching

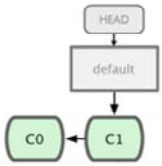
- Git sees commit this way...
- Branch annotates which commit we are working on

lightweight, movable pointers to a commit

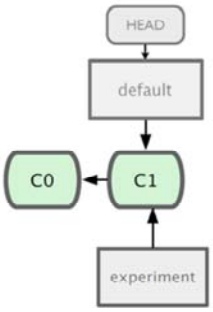




Branching

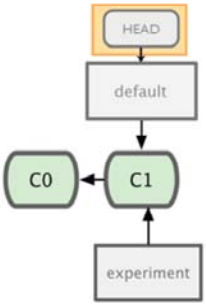


Branching



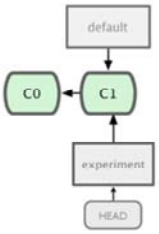
git branch experiment

Branching




```
$ git branch
* default
  experiment
```

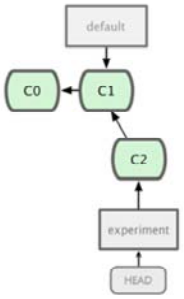
Branching



git checkout experiment

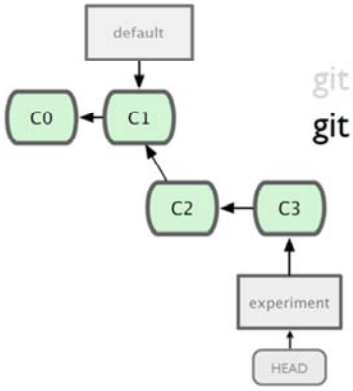
39

Branching



git commit

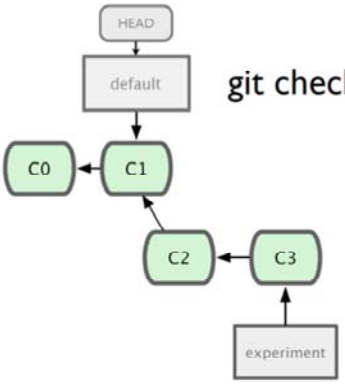
Branching



git commit

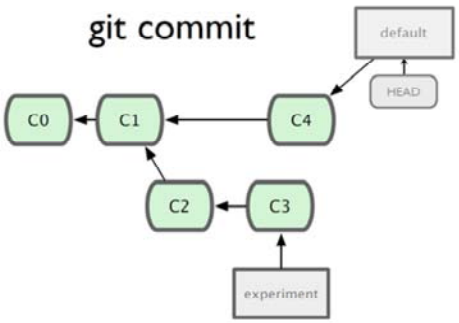
git commit

Branching



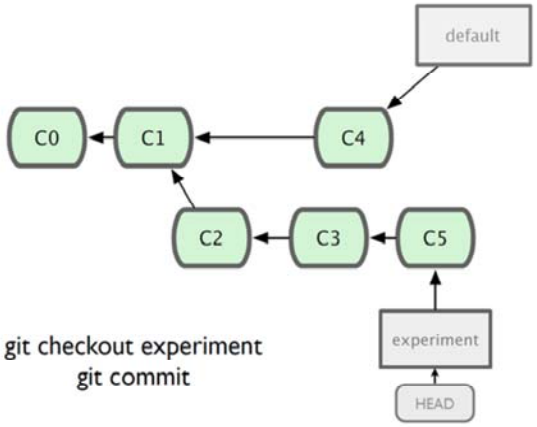
git checkout default

Branching



43

Branching

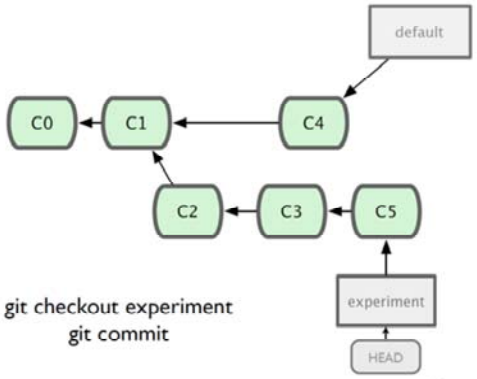


git checkout experiment
git commit

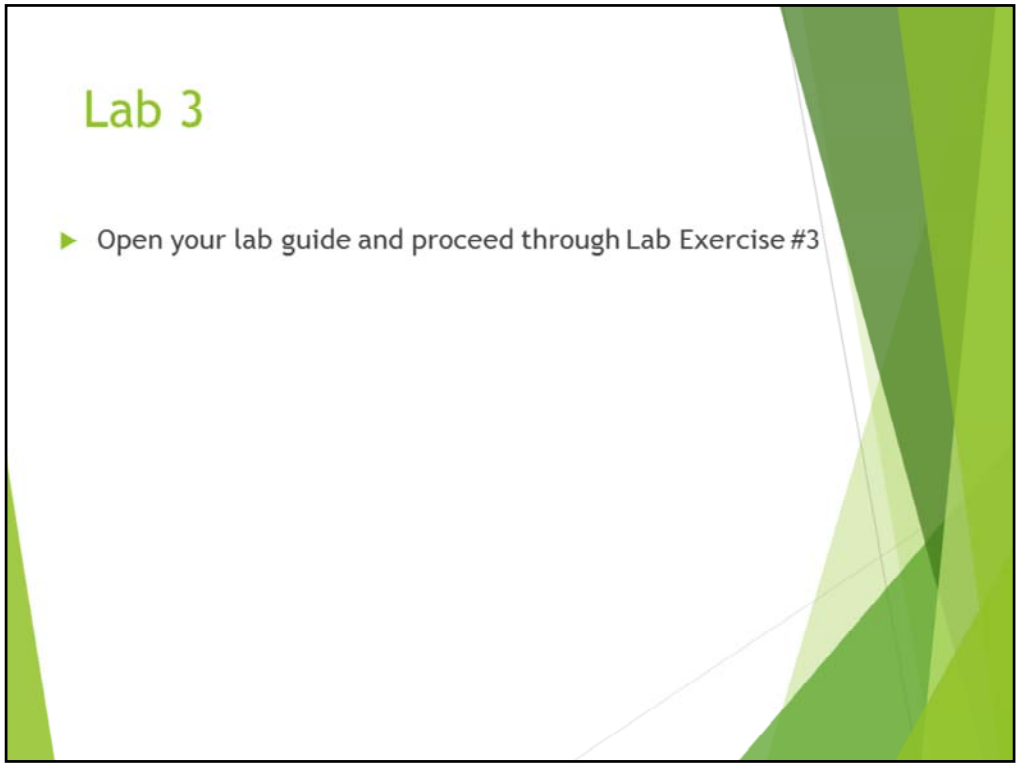
44

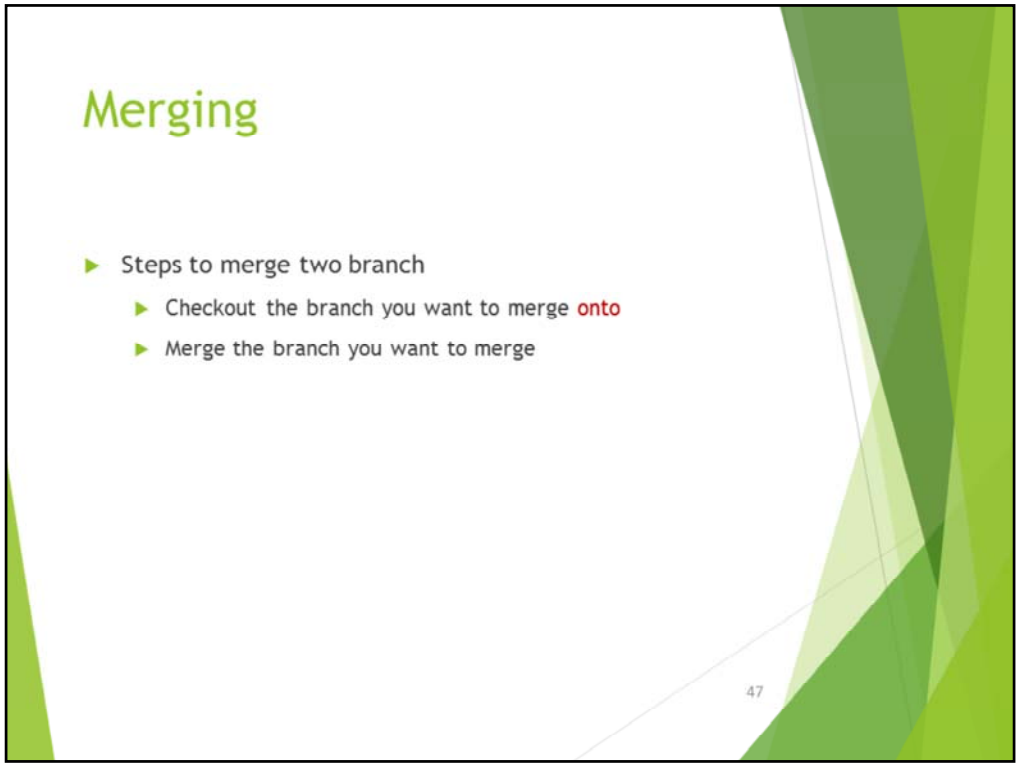
Merging

- What do we do with this mess?
  - Merge them

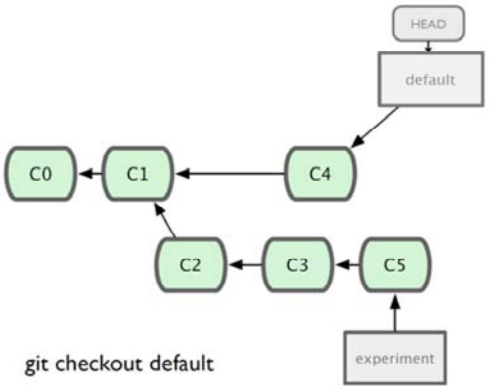


git checkout experiment
git commit


Lab 3
Open your lab guide and proceed through Lab Exercise #3


Merging
Steps to merge two branch
Checkout the branch you want to merge onto
Merge the branch you want to merge
47

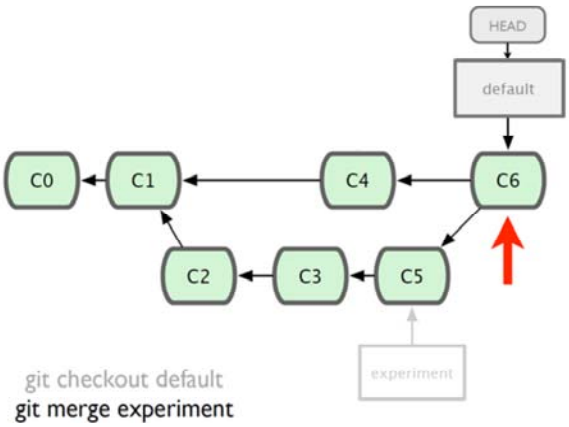
Merging



git checkout default

48

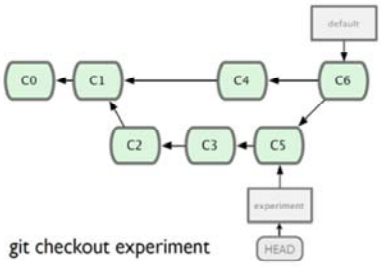
Merging



git checkout default
git merge experiment

49

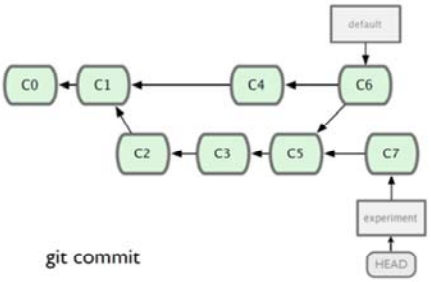
Merging



git checkout experiment

50

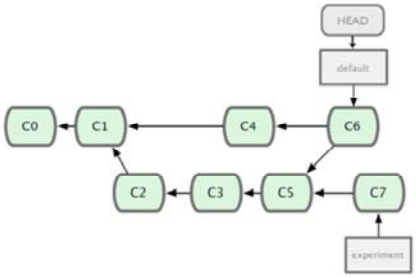
Merging



git commit

51

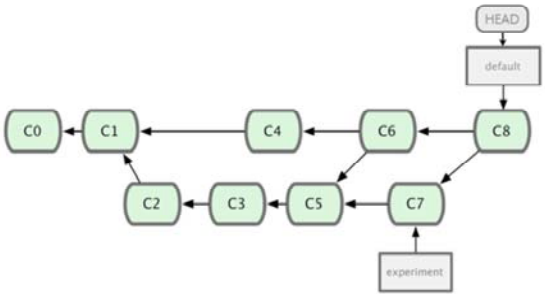
Merging



git checkout default

52

Merging



git merge experiment

53

Branching and Merging

- Why this is cool?
  - Non-linear development

```
clone the code that is in production
create a branch for issue #53 (iss53)
work for 10 minutes
someone asks for a hotfix for issue #102
checkout 'production'
create a branch (iss102)
fix the issue
checkout 'production', merge 'iss102'
push 'production'
checkout 'iss53' and keep working
```

54

Working with remote

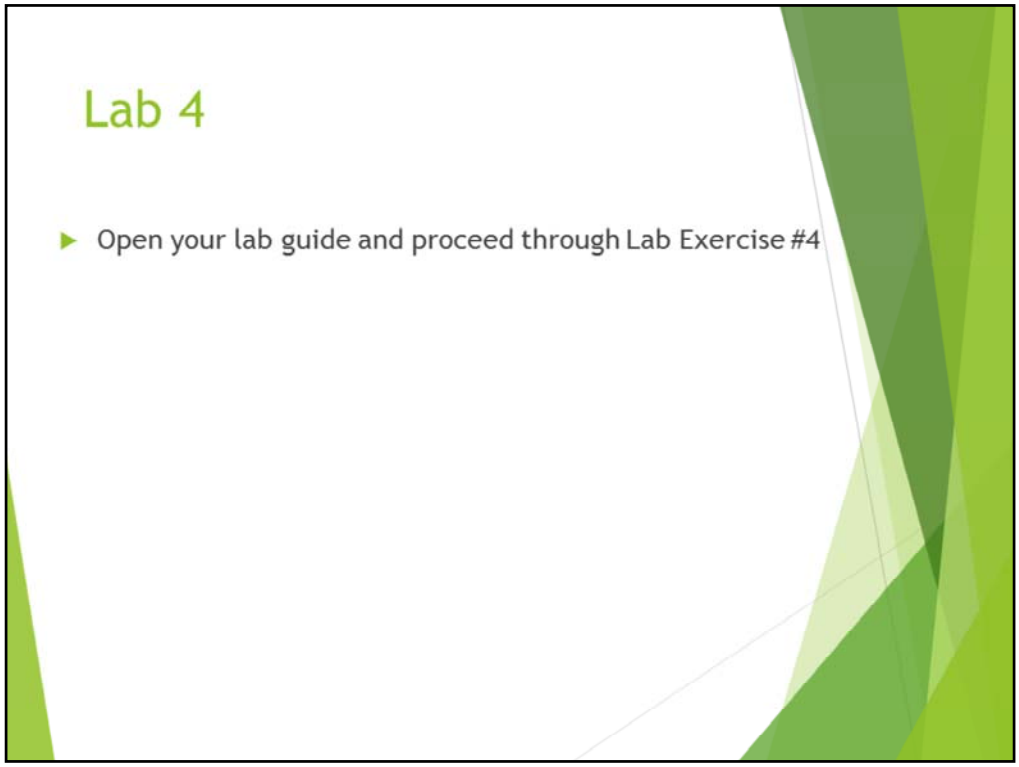
- Use git clone to replicate repository
- Get changes with
  - git fetch
  - git pull (fetches and merges)
- Propagate changes with
  - git push
- Protocols
  - Local filesystem (file://)
  - SSH (ssh://)
  - HTTP (http:// https://)
  - Git protocol (git://)

55

Working with remote Local filesystem

- Pros
  - Simple
  - Support existing access control
  - NFS enabled
- Cons
  - Public share is difficult to set up
  - Slow on top of NFS

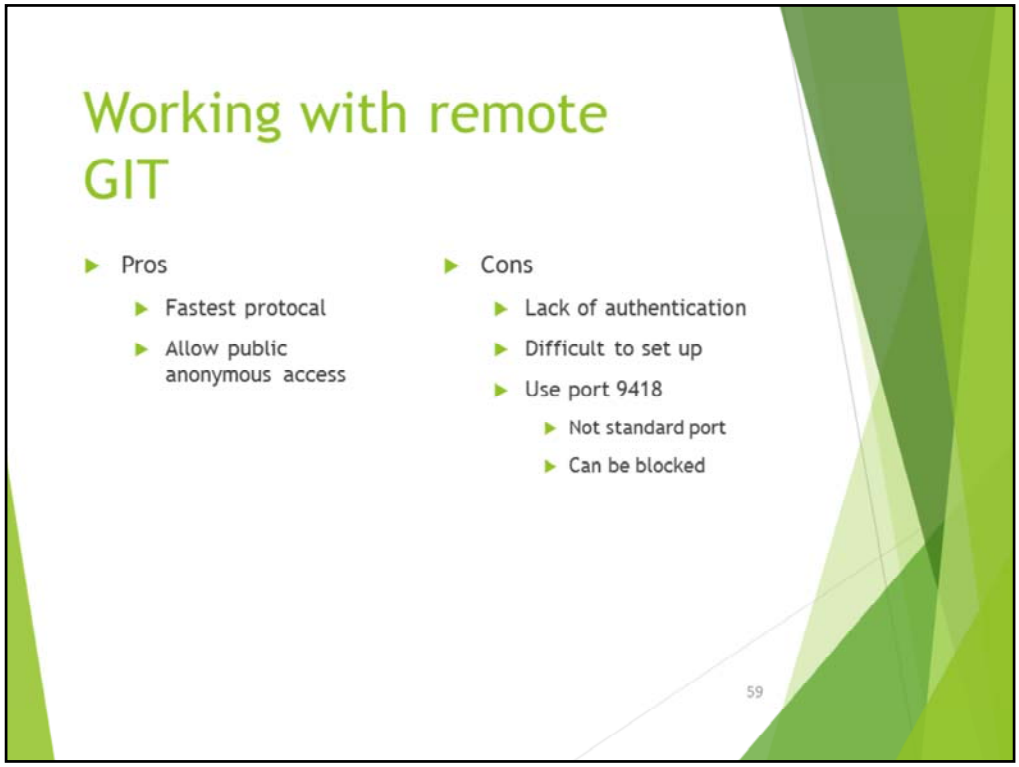
56


Lab 4
Open your lab guide and proceed through Lab Exercise #4

Working with remote SSH

- Pros
  - Support authenticated write access
  - Easy to set up as most system provide ssh toolsets
  - Fast
    - Compression before transfer
- Cons
  - No anonymous access
    - Not even for read access

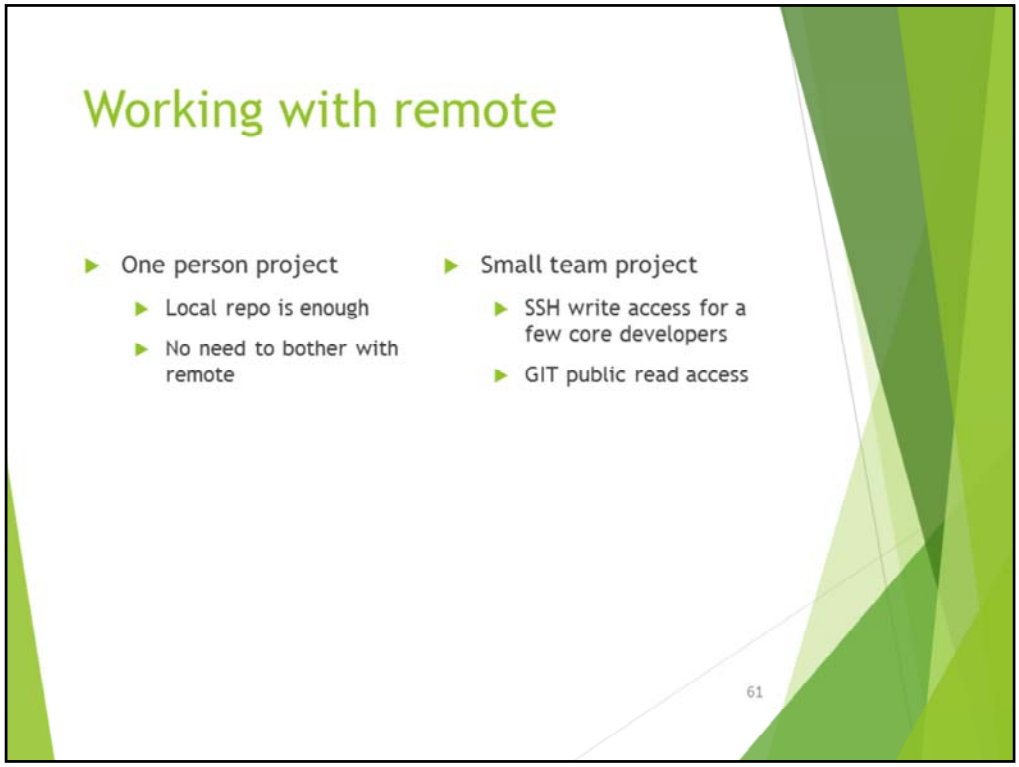
58


Working with remote GIT
Pros
Fastest protocal
Allow public anonymous access
Cons
Lack of authentication
Difficult to set up
Use port 9418
Not standard port
Can be blocked
59

Working with remote HTTP/HTTPS

- Pros
  - Very easy to set up
  - Unlikely to be blocked
    - Using standard port
- Cons
  - Inefficient

60


Working with remote
One person project
Local repo is enough
No need to bother with remote
Small team project
SSH write access for a few core developers
GIT public read access
61

Aside: So what is github?

- GitHub.com is a site for online storage of Git repositories.
- 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.

Question: Do I have to use github to use Git?

Answer: No!

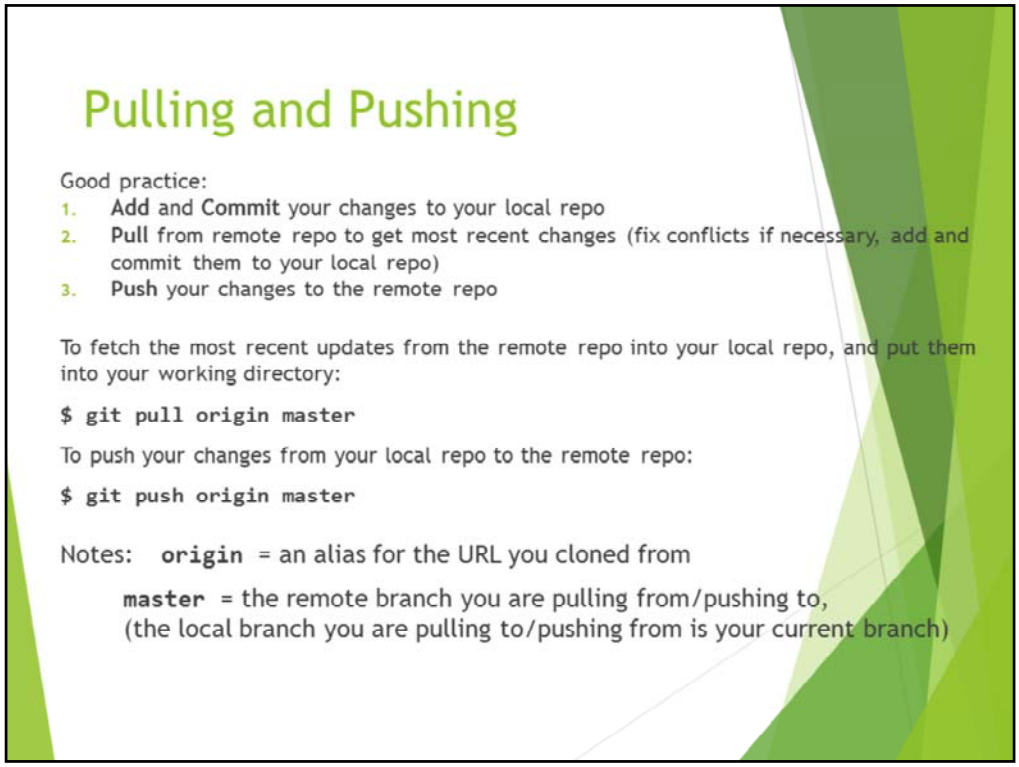
- you can use Git completely locally for your own purposes, or
- you or someone else could set up a server to share files, or
- you could share a repo with users on the same file system(as long everyone has the needed file permissions).

Working with remote

- Use git remote add to add an remote repository

```
Git remote add origin git@github.com:FreezingGod/vimcfg.git
zachary@zachary-desktop:~/.vim_runtime$ git remote
origin
```

63



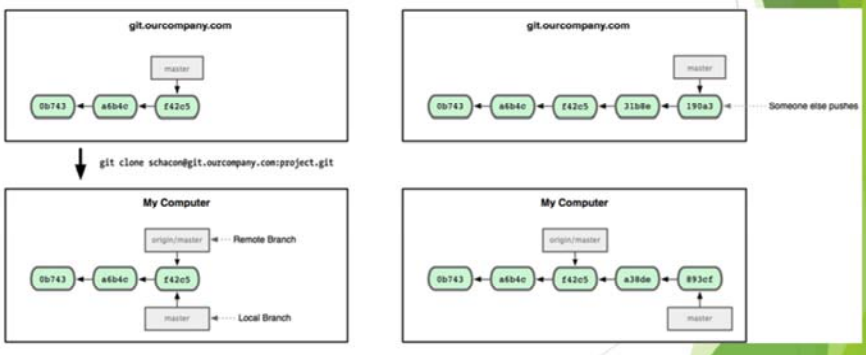
$ git remote -v

origin https://github.com/abc123/santalist.git (fetch)

origin https://github.com/abc123/santalist.git (push)

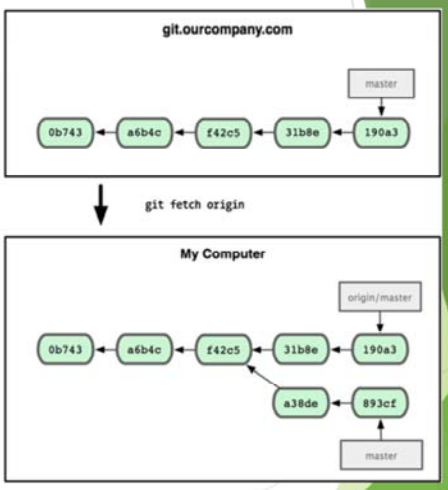
Working with remote

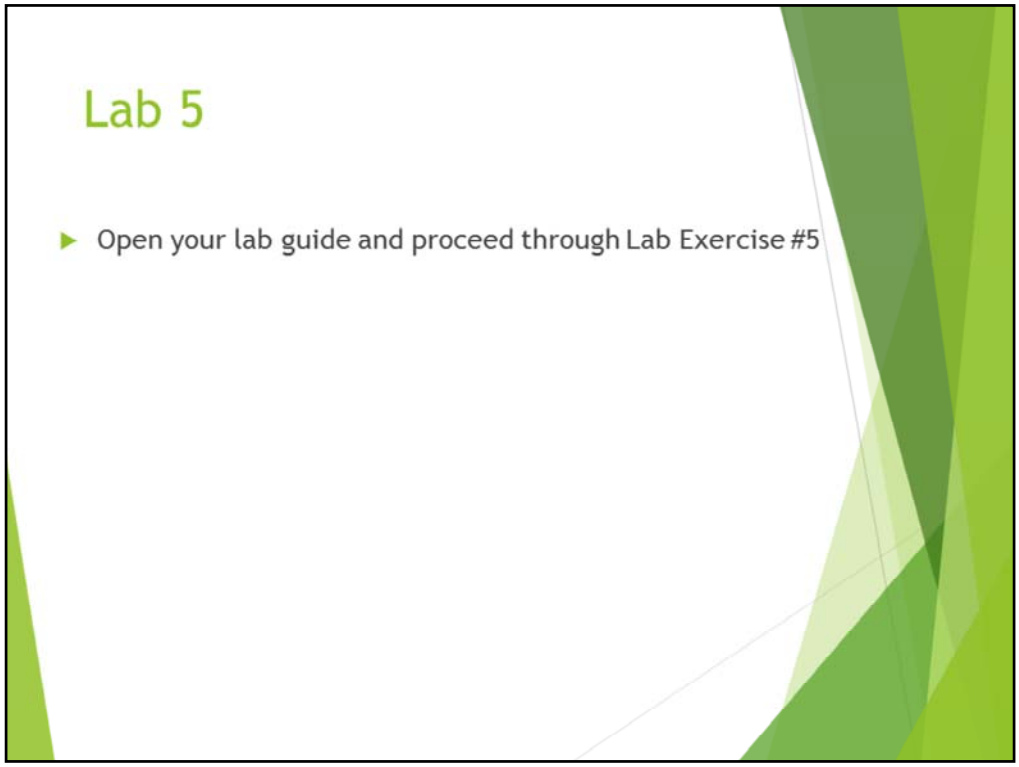
- Remote branching
  - Branch on remote are different from local branch



Working with remote

- Remote branching
  - Branch on remote are different from local branch
  - Git fetch origin to get remote changes
  - Git pull origin try to fetch remote changes and merge it onto current branch




Lab 5
Open your lab guide and proceed through Lab Exercise #5

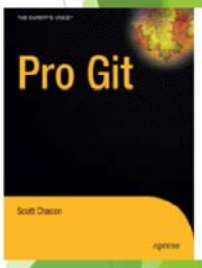
Summary

- We covered fundamentals of Git
  - Three trees of git
    - HEAD, INDEX and working directory
  - Basic work flow
    - Modify, stage and commit cycle
  - Branching and merging
    - Branch and merge
  - Remote
    - Add remote, push, pull, fetch
  - Other commands
    - Revert change, history view

68

Summary

- However, this is by no means a complete portray of git, some advanced topics are skipped:
  - Rebasing
  - Commit amend
  - Distributed workflow
- For more information, consult
  - Official document
  - Pro Git
    - Free book available at http://progit.org/book/



Q&A

- Any questions?

References

- Some of the slides are adopted from "Introduction to Git" available at

http://innovationontherun.com/presentation-files/Introduction%20To%20GIT.ppt

- Some of the figure are adopted from Pro GIT by Chacon, which is available at

http://progit.org/book/

- Some of the slides are adopted from "Git 101" available at http://assets.en.oreilly.com/1/event/45/Git%20101%20Tutorial%20Presentation.pdf

71