

# SIAM: Getting Started with Git

based on <http://git-scm.com/book> and slides by Bart Trojanowski

Andrew Reisner and Nathan Bowman

March 11, 2014

# Table of Contents

- 1 Overview
- 2 Components
- 3 Operations
  - Creating and Updating
  - Getting Information
  - Branching and Remotes
- 4 Distributed Workflows
- 5 Git on the Web

# Overview

# Git

Git is a

- Free and Open Source
- Distributed
- Version Control System.



# Version Control System

Preserve a clear, timely record of software evolution

- Record changes to files
- History can be recalled/inspected

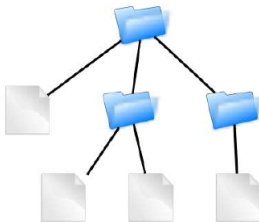
Implications:

- Rollback changes
- Know what collaborators are working on
- Investigate changes when bugs emerge
- Find how and where a particular bug was fixed

# Components

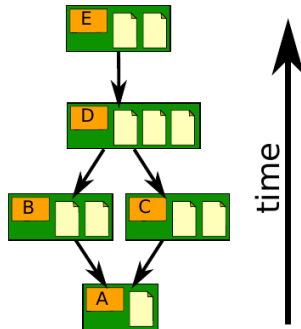
# VCS Components (Working Tree)

- Single checkout of one version of the project
- Directories
- Files



# VCS Components (Repository)

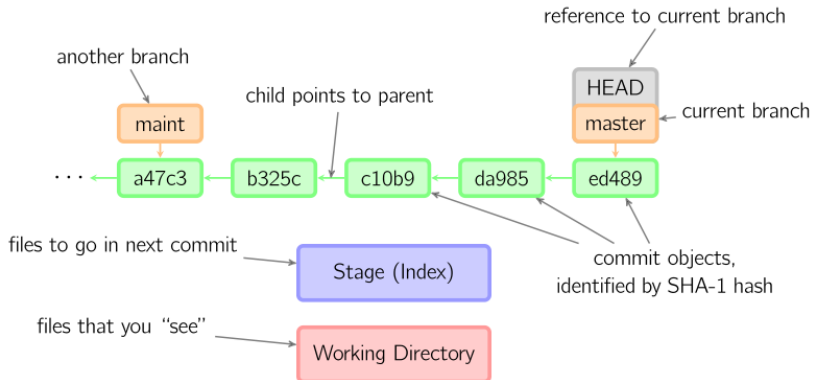
- Files
- Commits
- Ancestry





# VCS Components (References)

- Tags
- Branches
- HEAD
- Index (Staging area)



# Operations

# VCS Operations

## Bootstrap

- `init`
- `clone`
- `checkout`

## Modify

- `add`, `delete` (`rm`)
- `rename` (`mv`)
- `commit`

## Information

- `status`
- `diff`
- `log`

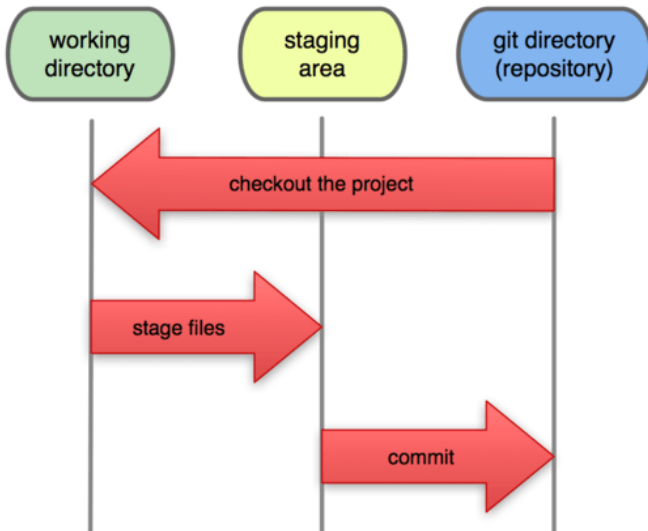
## Reference

- `tag`
- `branch`

## Sharing work, backing it up

- `pull`, `fetch`
- `push`

# Local Operations



# Bootstrapping

```
$ git init
```

- creates .git directory and initializes the repository

```
$ git clone <URL>
```

- replicates a remote repository
- checks out new working tree
- Git URLs
  - /home/user/my-project.git
  - http://github.com/user/my-project.git
  - git://remote.server/my-project.git
  - user@remote.server:my-project.git
  - ssh://user@remote.server/ user/my-project.git

# Staging

```
$ git add <path>
```

- Adds contents of <path> to index
- \$ git add .

```
$ git rm <file>
```

- Removes files from working tree and index

```
$ git mv <source> <destination>
```

- Moves or renames a file or directory

```
.gitignore
```

- Text file that specifies files to ignore

# Example .gitignore file

```
*.aux  
*.fdb_latexmk  
*.fls  
*.nav  
*.out  
*.snm  
*.toc  
*.vrb  
*~
```

# Changing Settings

```
$ git config --list
```

- Lists the current configuration settings

```
$ git config <key>
```

- Gets the current value of key

```
$ git config [level] <key> <value>
```

- Changes setting key to value
- Optional level determines scope of setting
  - Omitting level: repository
  - --global: user
  - --system: system





# Common Configuration Settings

A few settings you will want to update when first using Git:

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
$ git config --global core.editor emacs
$ git config --global core.excludesfile ~/.gitignore
$ git config --global merge.tool meld
```



# Committing

```
$ git commit -m <msg>
```

- Creates a commit of staged items
- `$ git commit -m "fixes issue #108"`

# Inspection

```
$ git status
```

- Displays the working tree status
- staged, unstaged, untracked

```
$ git diff
```

- Displays changes between index and working tree

```
$ git diff --staged
```

- Displays changes between HEAD and index

```
$ git diff HEAD
```

- Displays changes between HEAD and working tree

```
$ git diff <commit> <commit>
```

- Displays changes between two commits



# Demonstration of Staging

```
$ echo "foo" >> myfile
$ git diff myfile
diff --git a/myfile b/myfile
index e69de29..257cc56 100644
--- a/myfile
+++ b/myfile
@@ -0,0 +1 @@
+foo
```

# Referencing Objects

- `a88dbbe57b9e9fc01f701c45c405647c588e6a6a`
- `a88d`
- `v1.0.3`
- `master`
- `origin/master`
- `HEAD`
- `HEAD^ == HEAD~1`
- `feature_brach@{May.30}`

# Show and Log

```
$ git show <object>
```

- Show various types of objects

- `$ git show HEAD@{yesterday}`

- `$ git show HEAD:file`

```
$ git log [<since>..
```

- Show commit logs

- `$ git log HEAD~3..HEAD^`

- `$ git log -- file-with-bug.c`

# Branching

```
$ git branch -l
```

- List local branches

```
$ git branch <branchname>
```

- Create new branch on HEAD

```
$ git branch <branchname> <start-commit>
```

- Create new branch on specified commit

```
$ git checkout <branch>
```

- Checkout branch by name

```
$ git checkout -b <branchname> [<start-commit>]
```

- Create and switch to a new branch



# Merging

```
$ git merge <branch>
```

- Incorporates changes from the specified branch into the current branch.
- Conflicts may result
- Any conflicts must be resolved before merge is completed

```
var = 3
<<<<<<< HEAD
x = 0.5 * var
=====
x = 1/2. * var
>>>>>> origin/master
```

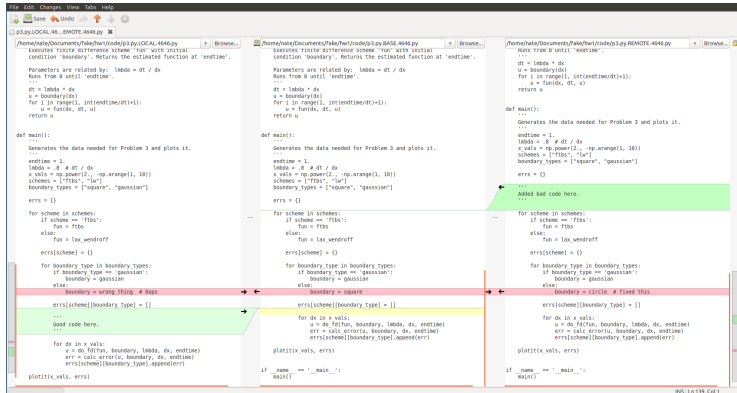




# Mergetool

\$ git merge <branch>

- Presents a visual interface to merging
- Example:
- \$ git mergetool --tool=meld



# Remotes

```
$ git remote add <name> <url>
```

- Adds a remote named <name> for the repository at <url>

```
$ git fetch <remote>
```

- Fetches updates from specified remote

- `$ git fetch --all`

```
$ git branch -r
```

- List remote branches
- Use `$ git merge` to merge these branches

```
$ git pull [<remote>] [<branch>]
```

- Short for a fetch followed by a merge

# Git Naming–Disambiguation

Git creates branches automatically in certain cases.

- HEAD: special reference that identifies the current branch
- master: Default branch created when a repository is first initialized
- origin: default name chosen for a remote when cloned
- `<remote_name>/<branch_name>`
  - origin/master
  - upstream/fix-issue-105



# Challenge Problem

Shape module at

<https://github.com/dattashantih/git-example.git>

- Fork and clone repository
- Locate and fix bug
- Push to your public repository
- Submit pull request (note: pull requests will be processed in order and must be up to date)

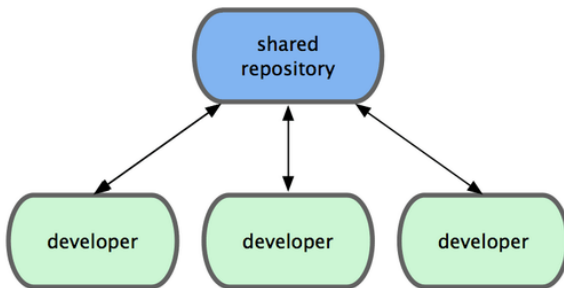


# Distributed Workflows

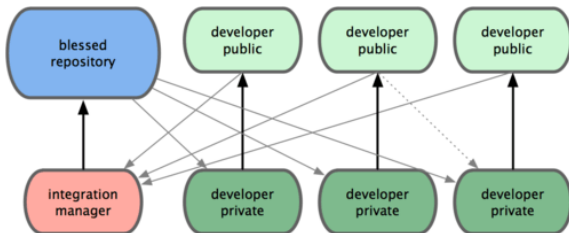
# Distributed

- No central location that keeps track of your data (no single place is more important than another)
- Encourages small commits and frequent merging
- Branches don't affect the main repository and can commit changes without disturbing others
- Work offline
- Rely on a network of trust

# Distributed Workflows: Centralized

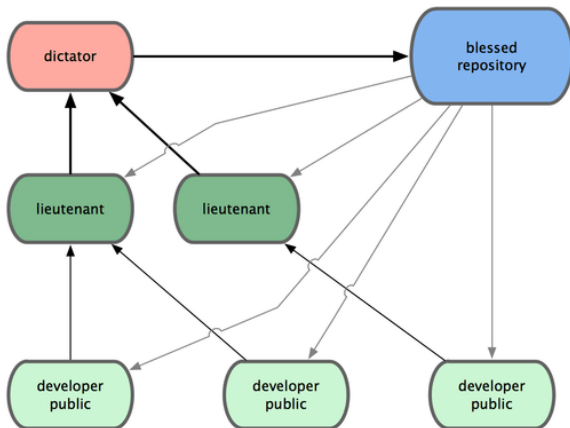


# Distributed Workflows: Integration-Manager





# Distributed Workflows: Dictator and Lieutenants



# Git on the Web

# Free and Open Source

- Downloads at <http://git-scm.com>
- Libgit2: free and open source library for writing custom Git applications



# GitHub

- Powerful web interface for publishing Git repositories
- Simple to view changes and track progress on repositories
- Wiki and bug tracking built into each repository



# Bitbucket

- Similar to GitHub
- Allows private repositories for students



# Resources

- 1 Git From the Bottom Up  
<http://ftp.newartisans.com/pub/git.from.bottom.up.pdf>
- 2 User Manual  
<http://git-scm.com/docs/user-manual.html>
- 3 Git Magic  
<http://www-cs-students.stanford.edu/~blynn/gitmagic/>
- 4 Git Book  
<http://git-scm.com/book>
- 5 Tech Talk: Linus Torvalds on git  
<http://youtu.be/4XpnKHJAok8>
- 6 Code School - Try Git  
<http://try.github.io>