SIAM: Getting Started with Git

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

Andrew Reisner and Nathan Bowman

April 6, 2017

Table of Contents

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

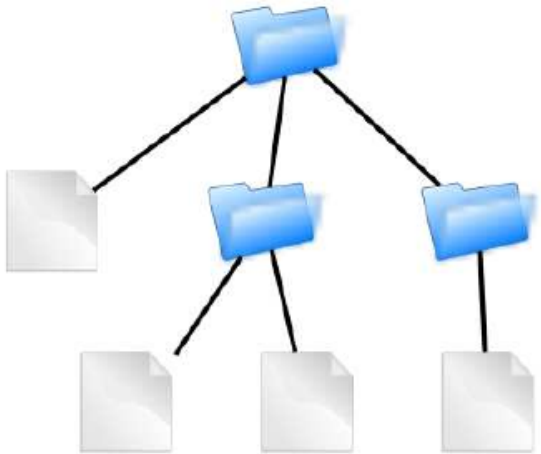
Implications:

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

Components

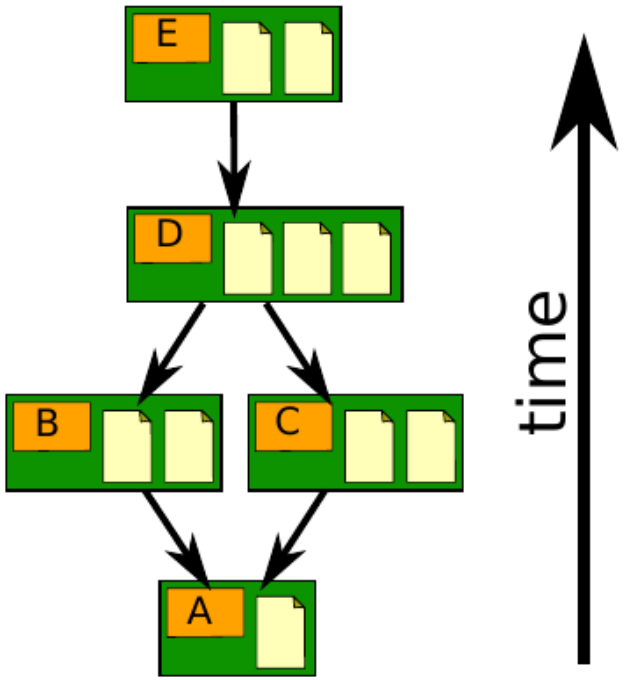
VCS Components (Working Tree)

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



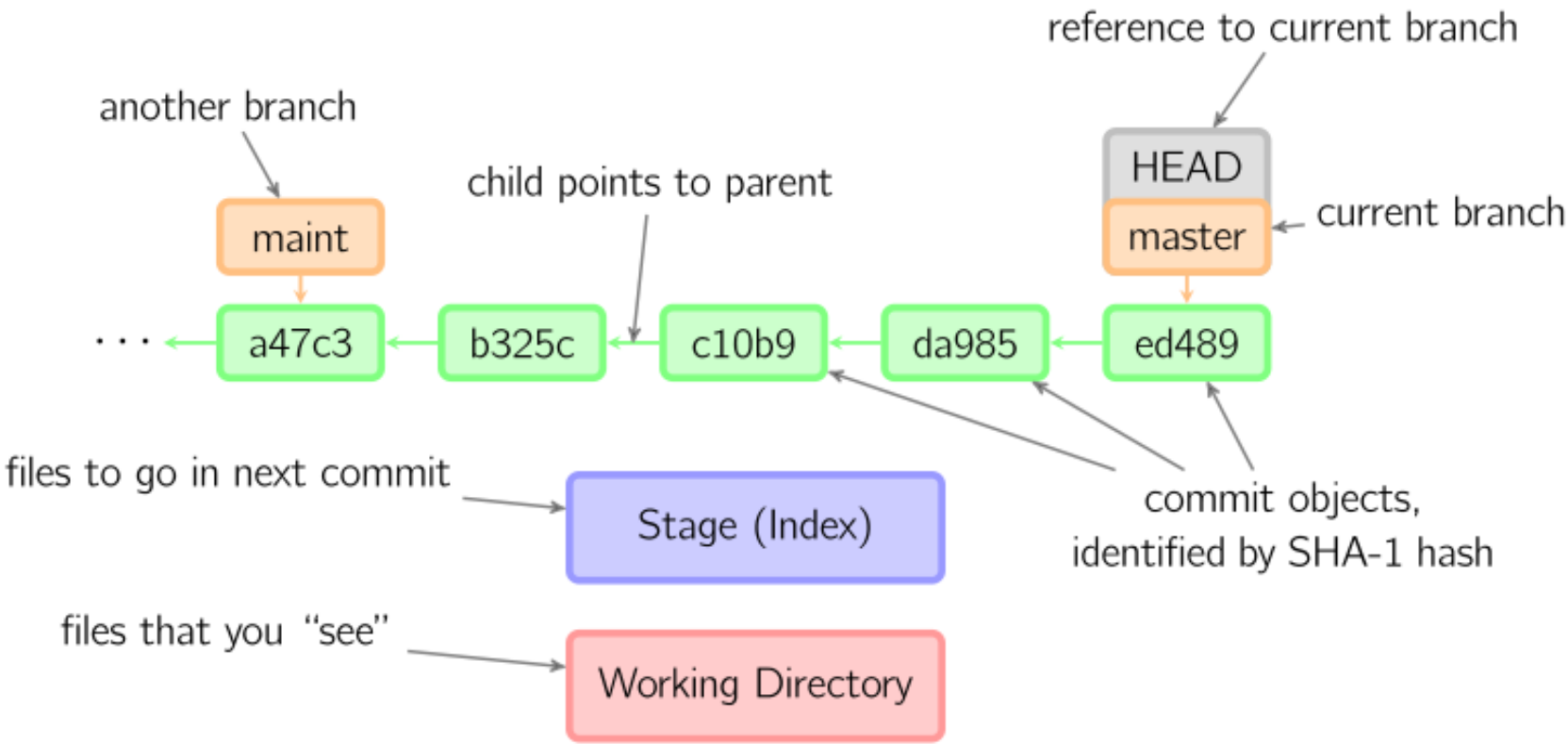
VCS Components (Repository)

- Files
- Commits
- Ancestry



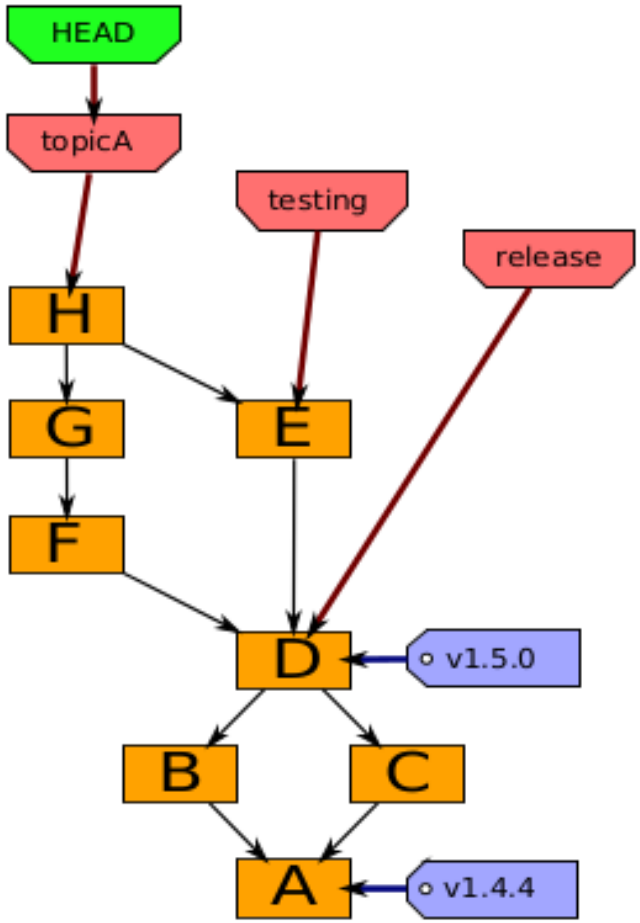
VCS Components (References)

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



[7]

VCS Components (Example Graph)



Operations

VCS Operations

Bootstrap

- `init`
- `clone`

Modify

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

Monitor Changes

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

Undo

- `checkout`
- `reset`

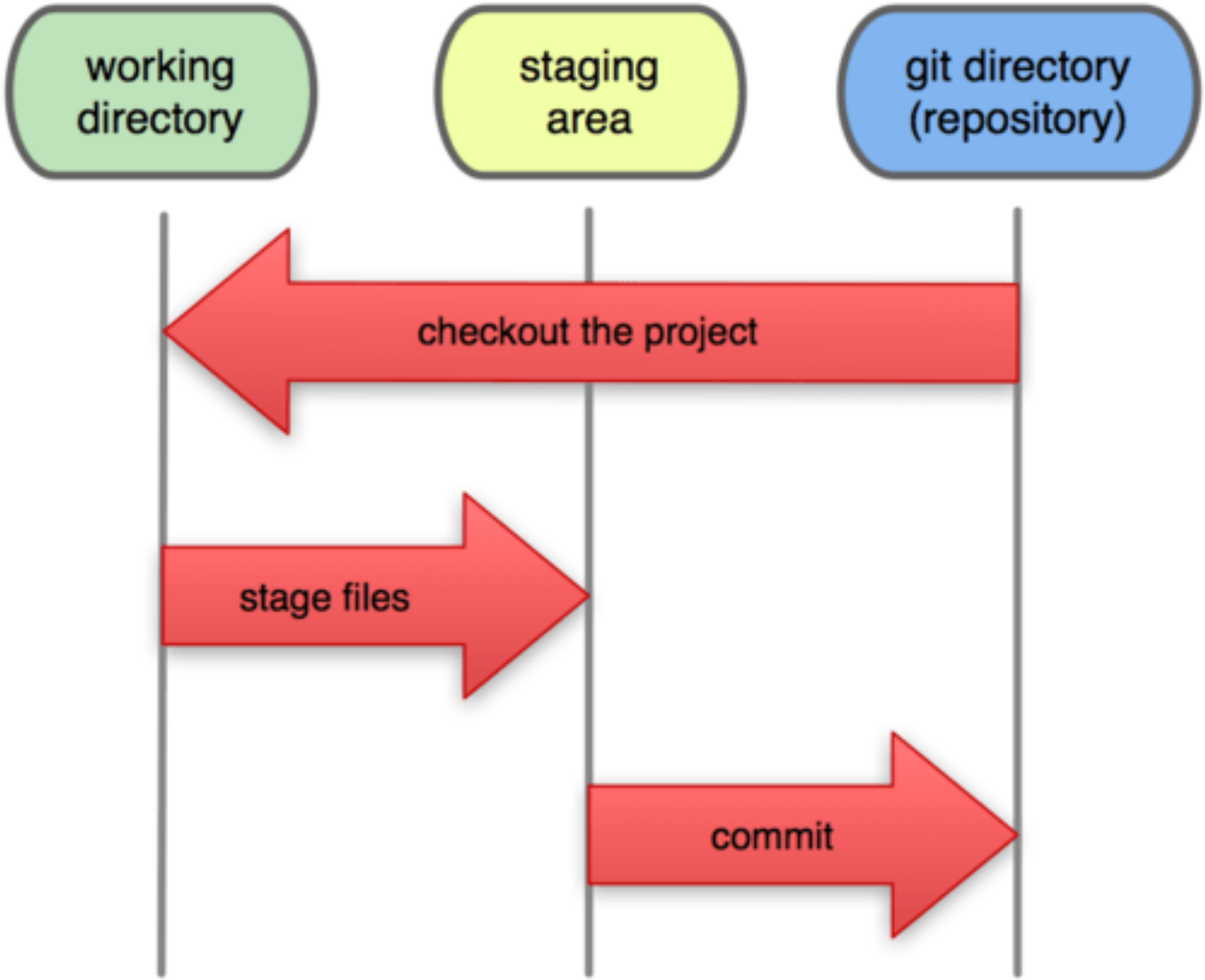
Branch

- `tag`
- `branch`
- `merge`

Share and Back Up

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

Local Operations



[1]

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

Initializing Empty Repository

```
$ ls -a
. ..
$ git init
Initialized empty Git repository in
    /home/user/my-project/.git/
$ ls -a
. .. .git
$ ls
$
```

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

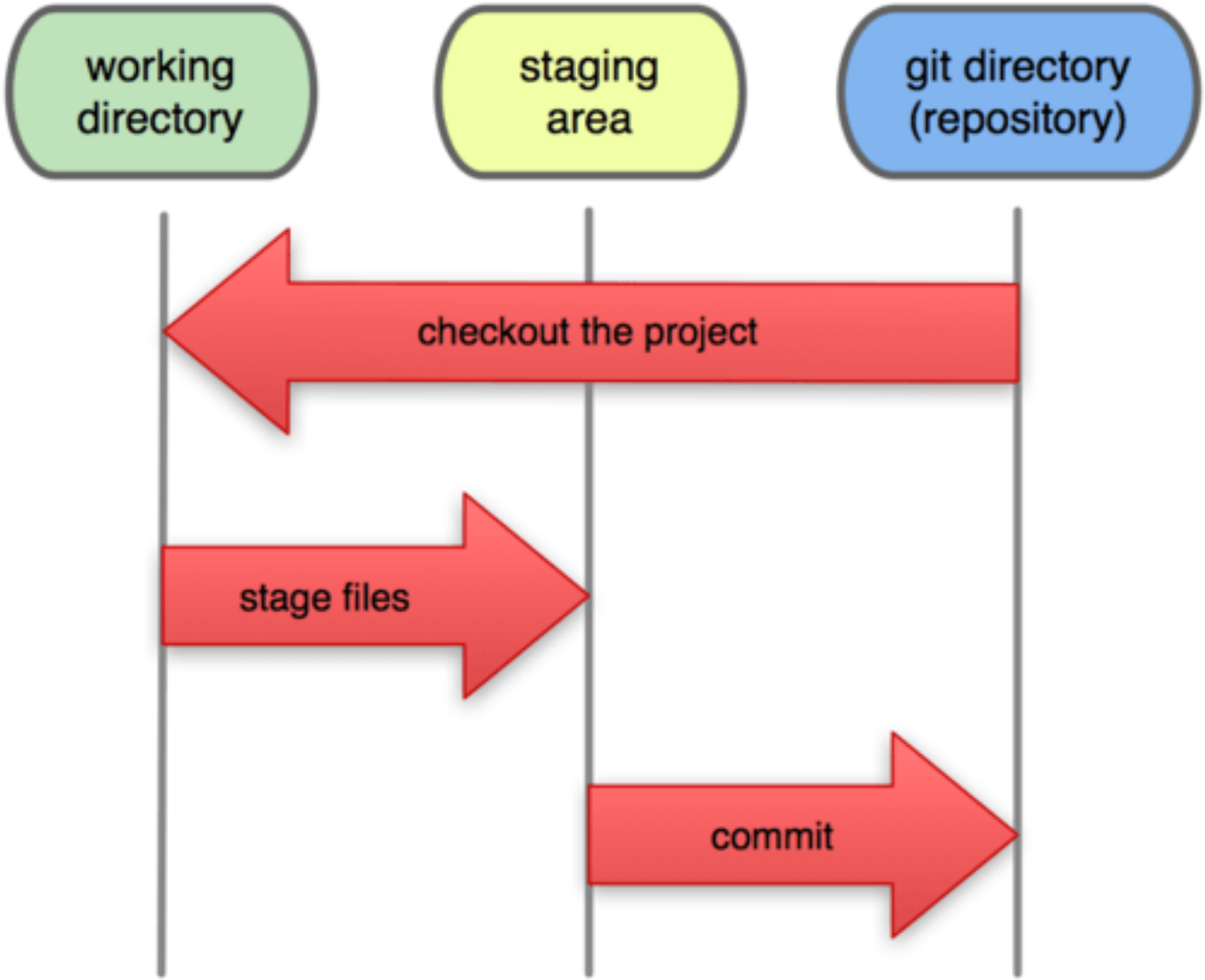
Adding our First File

```
$ echo 'Hi, my name is Andrew' > name_file.txt
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will
    be committed)

        name_file.txt

nothing added to commit but untracked files present
    (use "git add" to track)
```

Adding our First File

```
$ git add name_file.txt
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   name_file.txt
```

Local Operations



[1]

Committing

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

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

Creating our First Commit

```
$ git commit -m 'Add greeting'

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this
    repository.

fatal: empty ident name (for <(null)>) not allowed
```

Creating our First Commit

```
$ git commit -m 'Add greeting'
[master (root-commit) dec6e96] Add greeting
 1 file changed, 1 insertion(+)
 create mode 100644 name_file.txt
```

Ignoring Files

`.gitignore`

- Text file that specifies files to ignore

Example .gitignore file

```
*.out
todo_list.txt
```

Ignoring Files in Status

```
$ ls -a
.  ..  .git  .gitignore  name_file.txt  test2.out
   test.out  todo_list.txt
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will
   be committed)

        .gitignore

nothing added to commit but untracked files present
   (use "git add" to track)
```

Ignoring Files when Staging

```
$ ls -a
.  ..  .git  .gitignore  name_file.txt  test2.out
   test.out  todo_list.txt
$ git add .
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   .gitignore
$ git commit -m 'Add ignore file'
[master b488e63] Add ignore file
 1 file changed, 2 insertions(+)
 create mode 100644 .gitignore
```

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

Spotting Changes

```
$ echo 'I like git' >> name_file.txt
$ git add name_file.txt
$ echo 'Hello, world!' >> name_file.txt
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   name_file.txt

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:   name_file.txt
```

Spotting Changes

```
$ git diff
diff --git a/name_file.txt b/name_file.txt
index fa864f7..d5e2134 100644
--- a/name_file.txt
+++ b/name_file.txt
@@ -1,2 +1,3 @@
 Hi, my name is Andrew
 I like git
+Hello, world!
```

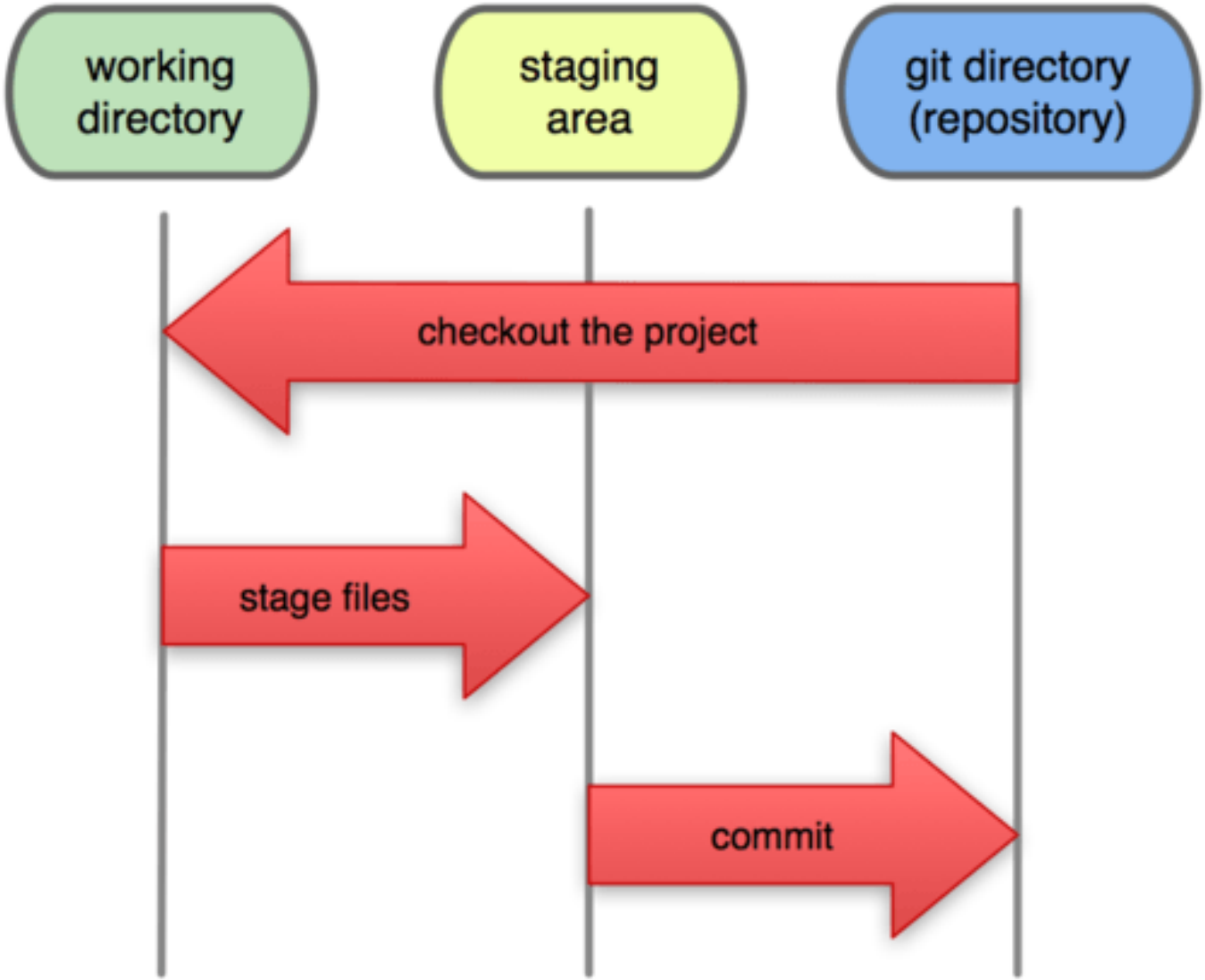
Spotting Changes

```
$ git diff --staged
diff --git a/name_file.txt b/name_file.txt
index c987f6b..fa864f7 100644
--- a/name_file.txt
+++ b/name_file.txt
@@ -1 +1,2 @@
 Hi, my name is Andrew
+I like git
```

Spotting Changes

```
$ git diff HEAD
diff --git a/name_file.txt b/name_file.txt
index c987f6b..d5e2134 100644
--- a/name_file.txt
+++ b/name_file.txt
@@ -1 +1,3 @@
 Hi, my name is Andrew
+I like git
+Hello, world!
```

Local Operations



[1]

Undoing Changes to Working Directory

`$ git checkout <filename>`

- Put file from staging area into working directory
- Undo unstaged changes to file

`$ git checkout <commit> -- <filename>`

- Put file from specified commit into working directory and staging area
- Overwrite unstaged changed to file

The `checkout` command has other uses when dealing with branches (discussed later). Be warned – `git checkout <commit>` without filename argument does not do what you expect.

Erasing Unstaged Changes

```
$ git checkout name_file.txt
$ cat name_file.txt
Hi, my name is Andrew
I like git
```

Undoing Changes to Staging Area

The `reset` command is similar to `checkout` for staging area

`$ git reset`

- Unstage all changes
- Reset staging area to HEAD

`$ git reset <filename>`

- Unstage changes to file
- Reset file in staging area to HEAD

The `reset` command will not touch the working directory unless passed an additional argument. Follow `reset` with `checkout` to undo changes to working directory.

The `reset` command, like `checkout`, has other uses related to branches, but we will not cover these.

Erasing Unstaged Changes

```
$ git reset name_file.txt
Unstaged changes after reset:
M       name_file.txt
$ git checkout name_file.txt
$ cat name_file.txt
Hi, my name is Andrew
```

Viewing History

`$ git log [<since>..<until>] [-- <path>]`

- Show commit logs
- `$ git log HEAD~3..HEAD^`
- `$ git log -- file-with-bug.c`

`$ git show <object>`

- Show various types of objects
- `$ git show HEAD@{yesterday}`
- `$ git show HEAD:file`

Viewing Log

```
$ git log
commit 4f6f4a4a4d432a8c22fda5dff7006dfc026e126f
Author: Your Name <you@example.com>
Date:   Mon Apr 3 22:05:50 2017 -0500

    Add ignore file

commit dec6e96fe5ad9d2f419e775c2f4a1b0ac52316e2
Author: Your Name <you@example.com>
Date:   Mon Apr 3 17:37:57 2017 -0500

    Add greeting
```

Referencing Objects

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

Examining Commit Object

```
$ git show dec6e
commit dec6e96fe5ad9d2f419e775c2f4a1b0ac52316e2
Author: Your Name <you@example.com>
Date:   Mon Apr 3 17:37:57 2017 -0500

    Add greeting

diff --git a/name_file.txt b/name_file.txt
new file mode 100644
index 0000000..c987f6b
--- /dev/null
+++ b/name_file.txt
@@ -0,0 +1 @@
+Hi, my name is Andrew
```

Log Formatting

```
$ git log --pretty=<format>
```

- `oneline`
- `full`
- `format:"hash: %h author: %an date: %ad"`
- see git-log(1) for more options

```
$ git log --graph --pretty=oneline
```

Using History

```
$ echo 'I like git' >> name_file.txt
$ echo 'Hello, world!' >> name_file.txt
$ git commit -am 'Share more information'
Share more information
 1 file changed, 2 insertions(+)
```

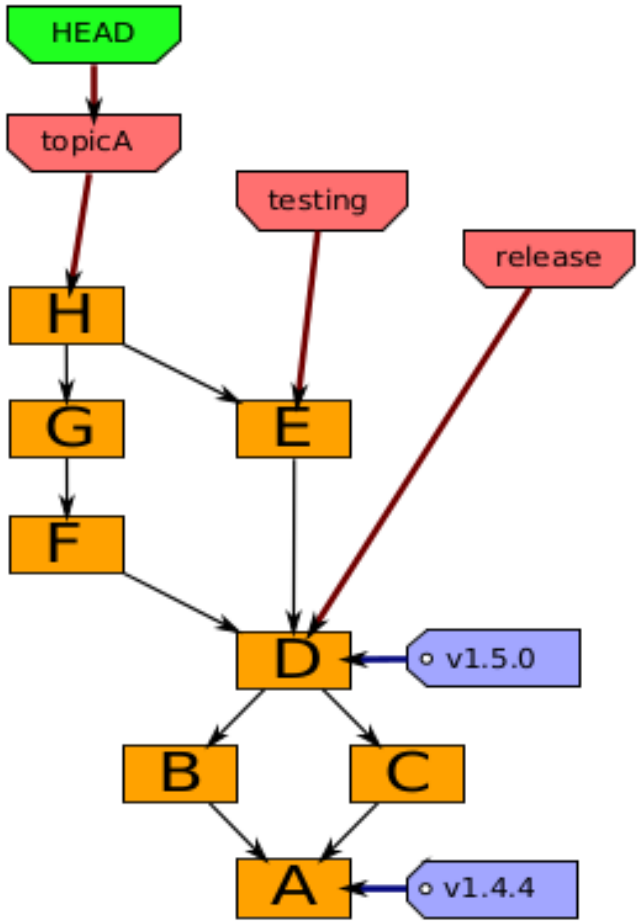
Using History

```
$ git diff HEAD~2
diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..d0833a3
--- /dev/null
+++ b/.gitignore
@@ -0,0 +1,2 @@
+*.out
+todo_list.txt
diff --git a/name_file.txt b/name_file.txt
index c987f6b..d5e2134 100644
--- a/name_file.txt
+++ b/name_file.txt
@@ -1 +1,3 @@
 Hi, my name is Andrew
+I like git
+Hello, world!
```

Using History

```
$ git show HEAD~1:name_file.txt
Hi, my name is Andrew
$ git checkout HEAD~1 -- name_file.txt
```

Three Branches



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

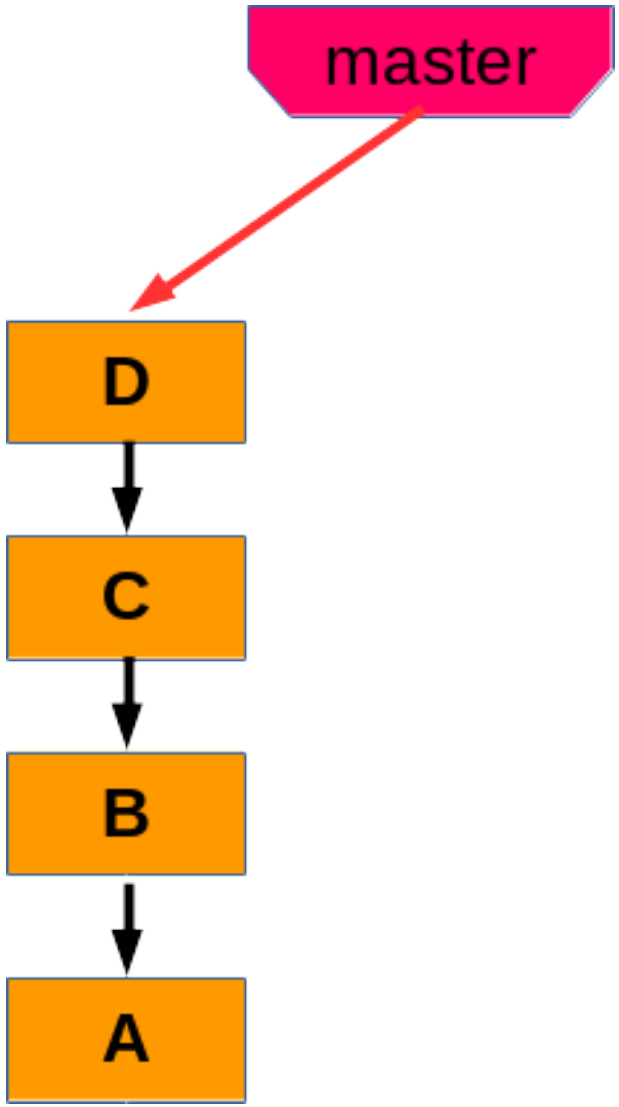
`$ git checkout -b <branchname> [<start-commit>]`

- Create and switch to a new branch

Branch Example

```
$ echo 'var = 3' > math_file.py
$ git add math_file.py
$ git commit -m 'Start programming'
[master af23d01] Start programming
 1 file changed, 1 insertion(+)
 create mode 100644 math_file.py
```

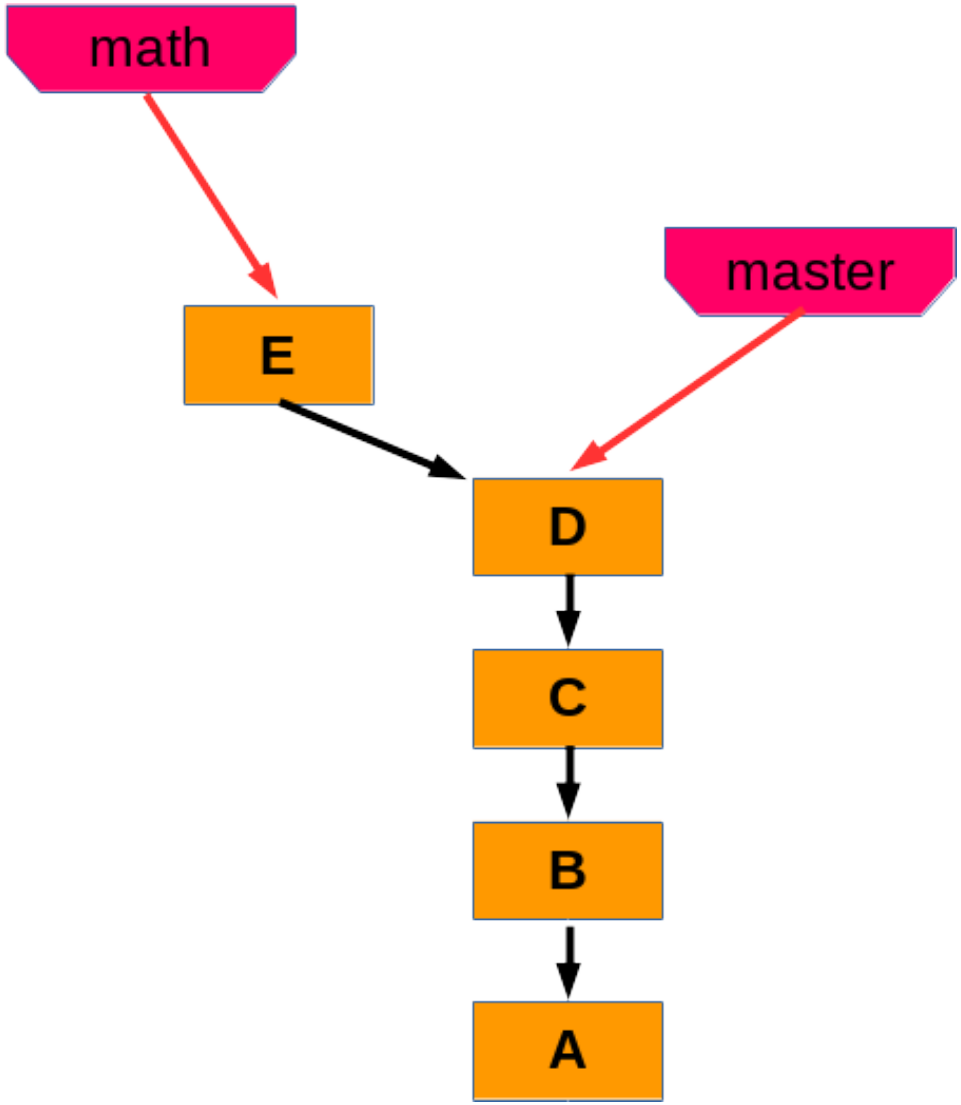
Before Branch



Branch Example

```
$ rm *.out todo_list.txt
$ touch file1 file2 file3
$ git branch math
$ git checkout math
Switched to branch 'math'
$ git add .
$ git commit -m 'Add more files'
[math db71718] Add more files
 3 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1
 create mode 100644 file2
 create mode 100644 file3
```

First Branch Commit



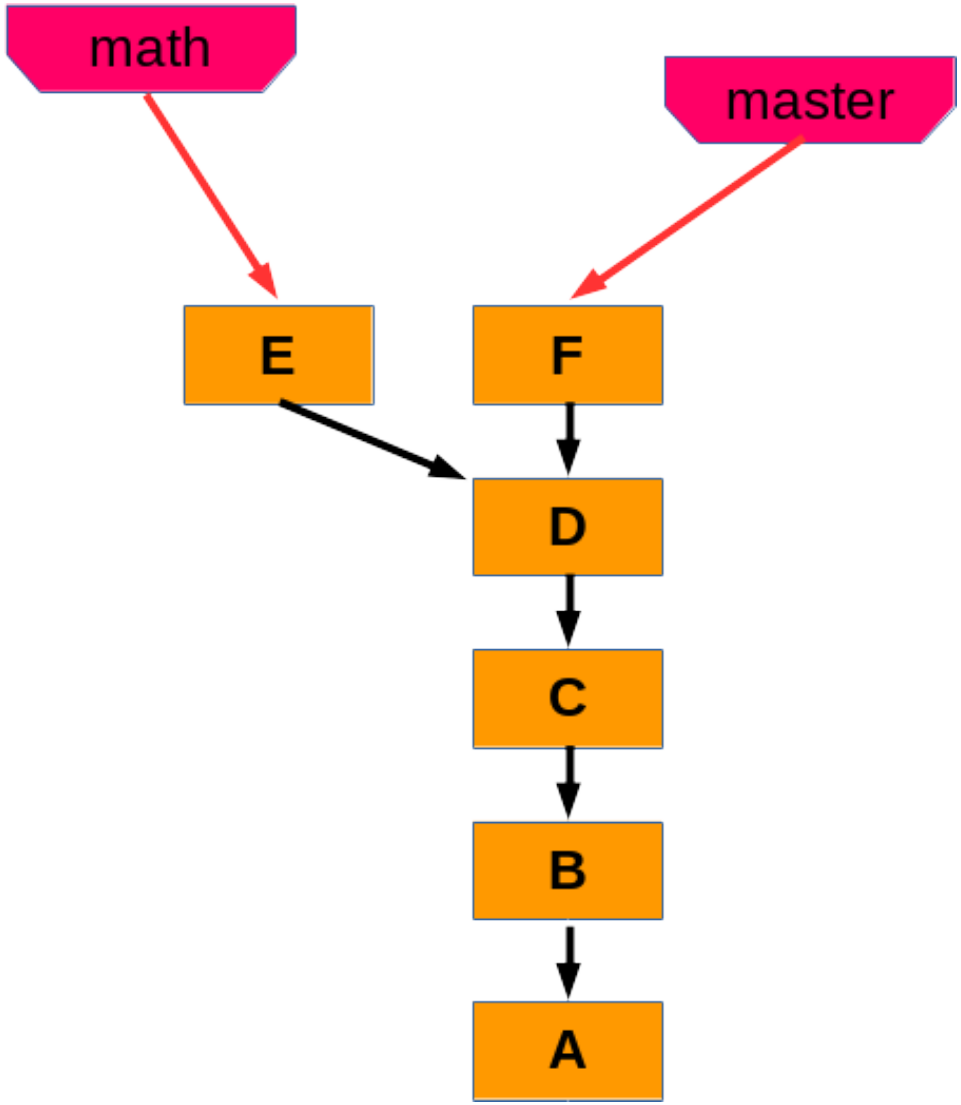
Branch Example

```
$ ls
file1  file2  file3  math_file.py  name_file.txt
$ git checkout master
Switched to branch 'master'
$ ls
math_file.py  name_file.txt
```

Branch Example

```
$ echo 'x = 0.5 * var' >> math_file.py
$ git commit -am 'Make change to master branch'
[master 7864aac] Make change to master branch
 1 file changed, 1 insertion(+)
$ git checkout math
Switched to branch 'math'
$ cat math_file.py
var = 3
```

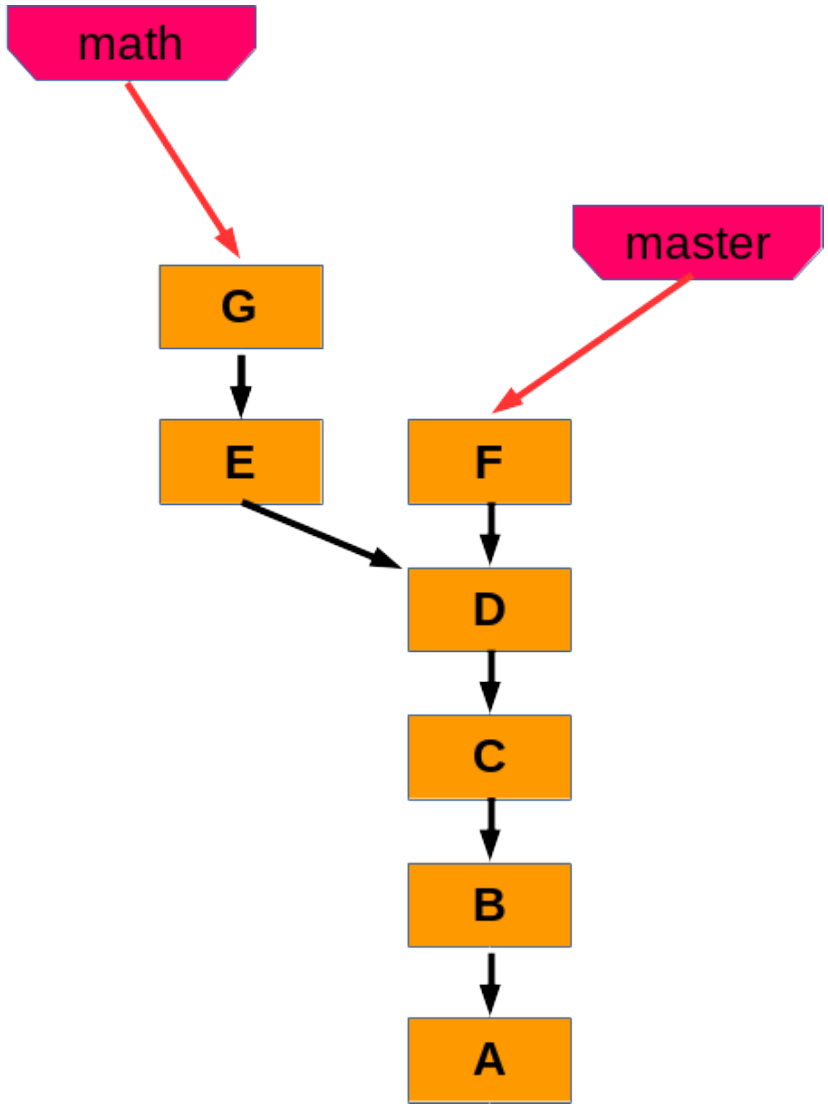
Master Branch Commit



Branch Example

```
$ cat math_file.py
var = 3
$ echo ' x = 1/2. * var' >> math_file.py
$ git commit -am 'Make conflicting change to math
    branch'
[math 6a015d3] Make conflicting change to math
    branch
 1 file changed, 1 insertion(+)
```

Another Math Commit



Merging

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

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

Merge Example

```
$ git checkout master
Switched to branch 'master'
$ git merge math
Auto-merging math_file.py
CONFLICT (content): Merge conflict in math_file.py
Automatic merge failed; fix conflicts and then
    commit the result.
$ cat math_file.py
var = 3
<<<<<<< HEAD
x = 0.5 * var
=======
 x = 1/2. * var
>>>>>>> math
```

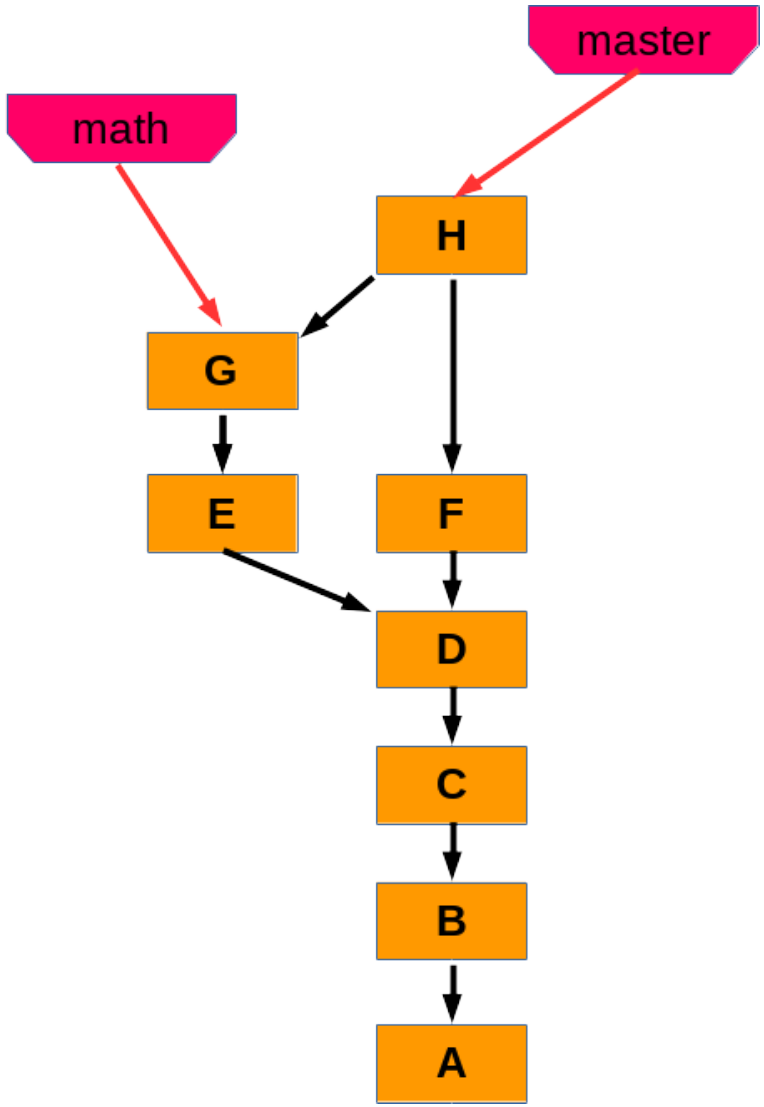
Merge Example

```
... fix changes
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Changes to be committed:

        new file:   file1
        new file:   file2
        new file:   file3

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   math_file.py
```

Merge Example

```
$ git add math_file.py
$ git commit
[master 211a76d] Merge branch 'math'
$ ls
file1  file2  file3  math_file.py  name_file.txt
```

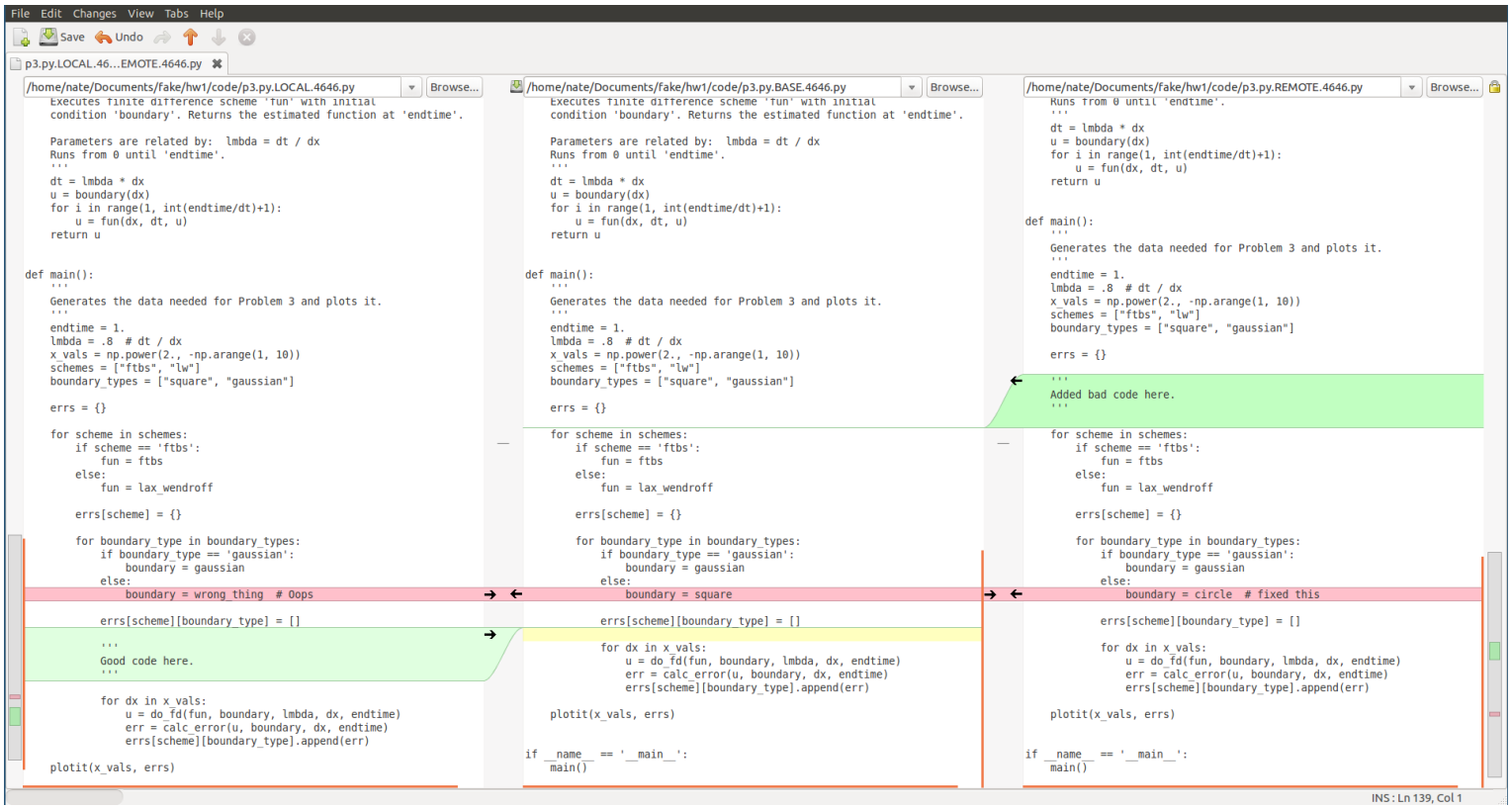
After Merge



Mergetool

`$ git mergetool`

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



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
```

Getting Help

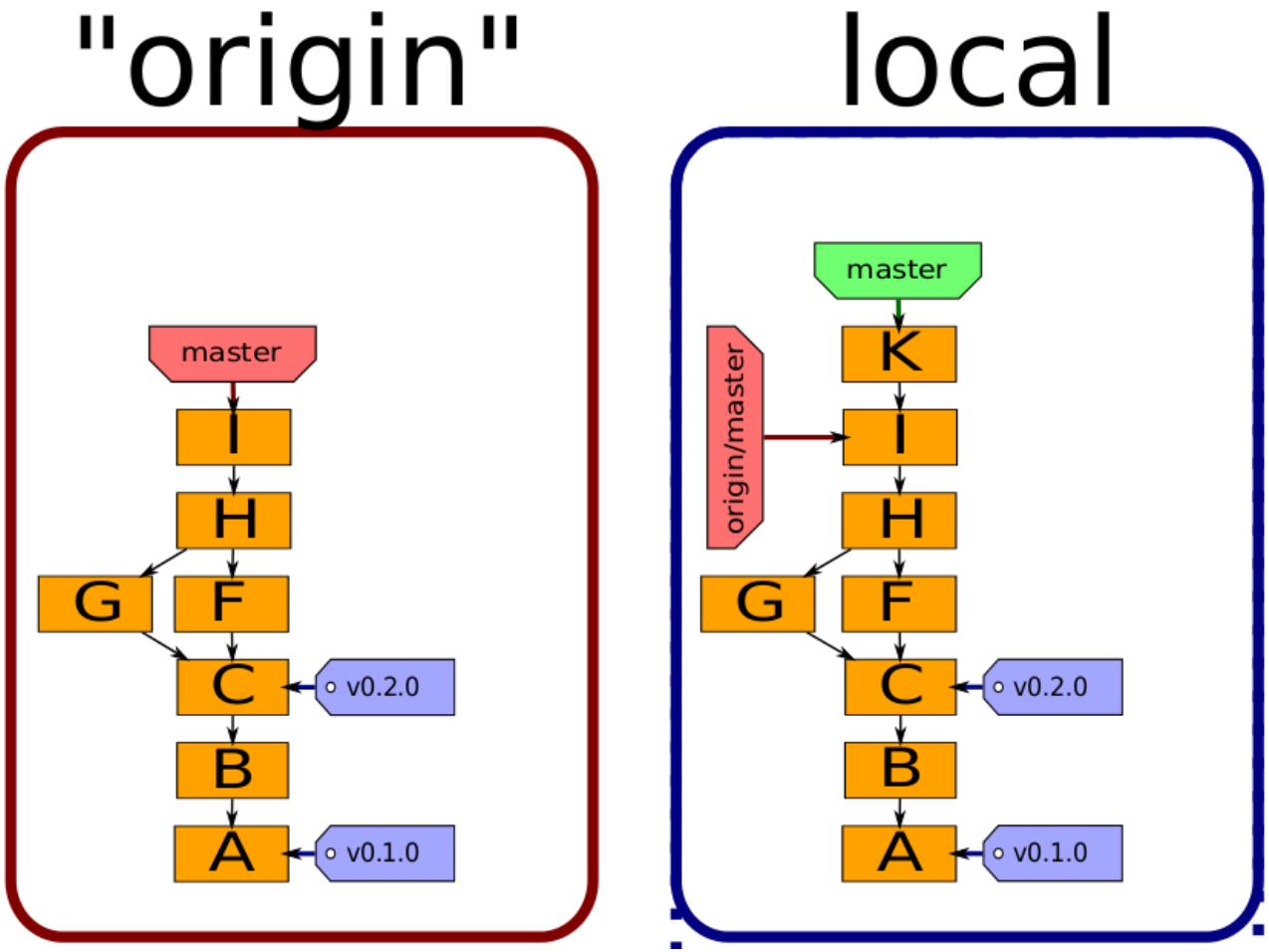
```
$ git help <command>
```

- Display *a lot* of information about command

Google and StackOverflow are great resources for quick questions. Chances are that almost any `git` question you have has been asked and answered already.

Sharing

Remotes Repository



Remotes

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

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

`$ git branch -r`

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

Remotes (2)

`$ git fetch <remote>`

- Fetches updates from specified remote
- `$ git fetch --all`

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

- Short for a fetch followed by a merge

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

- Send updates to remote
- Similar to running `pull` on remote machine
- No conflicts allowed

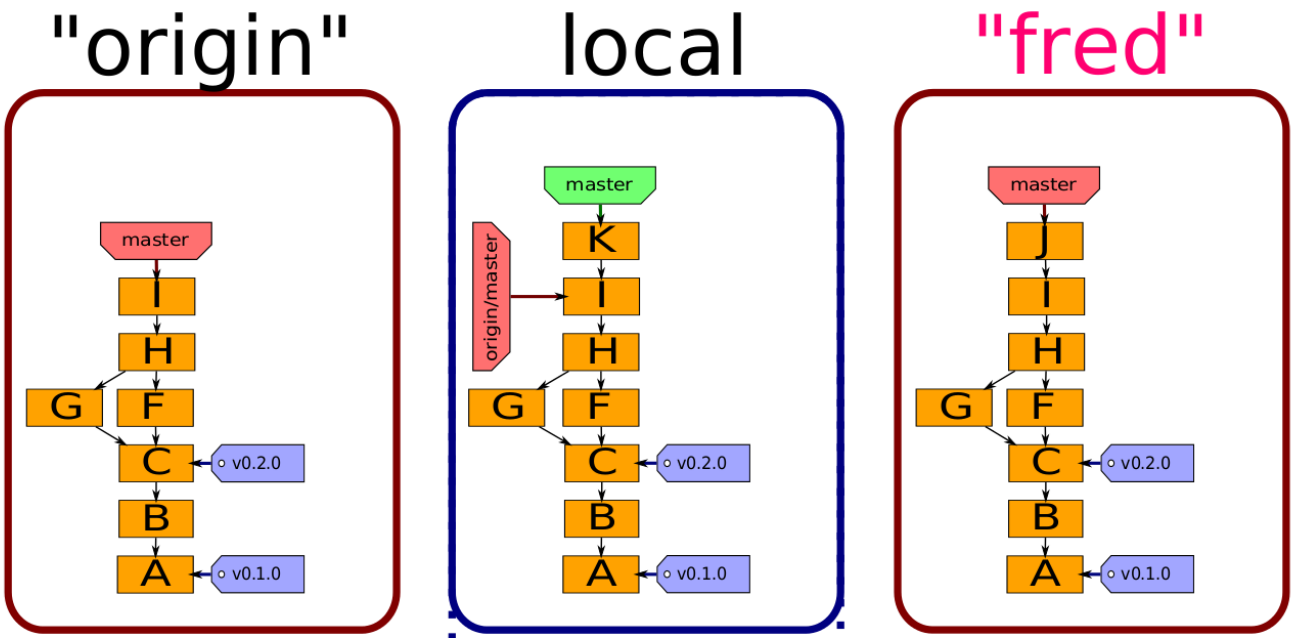
Git Naming–Disambiguation

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`

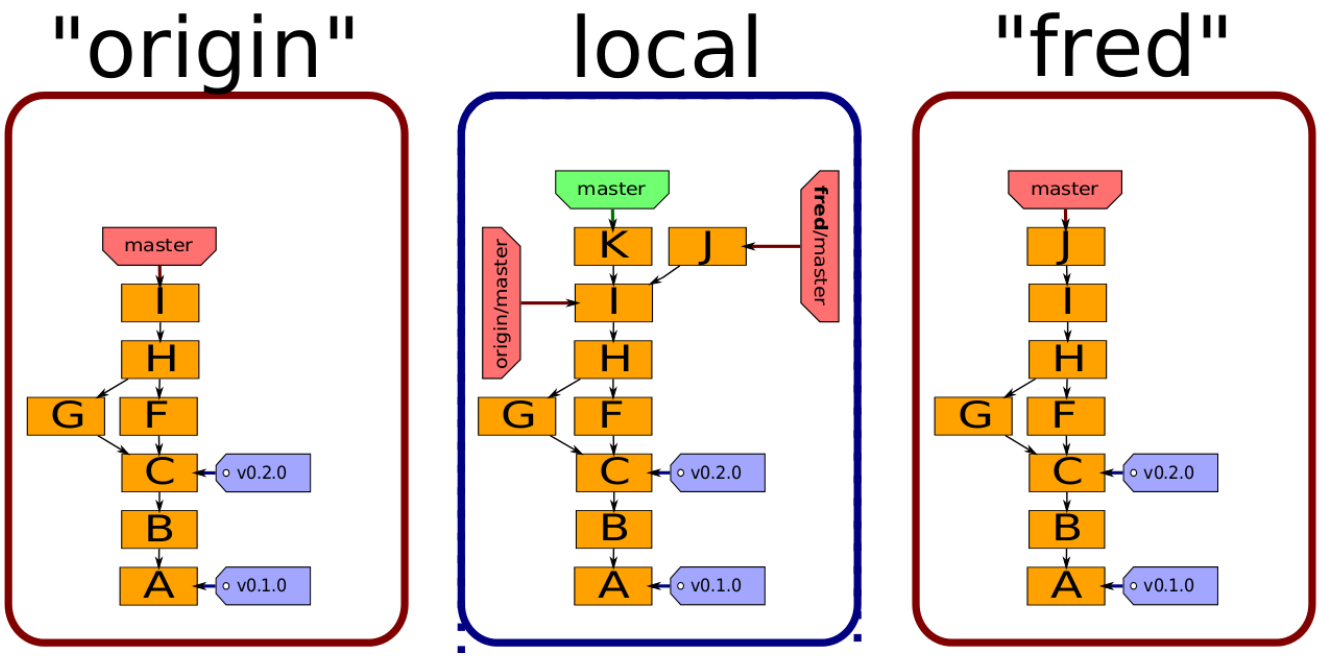
Remotes Example

"fred" cannot push to "origin"



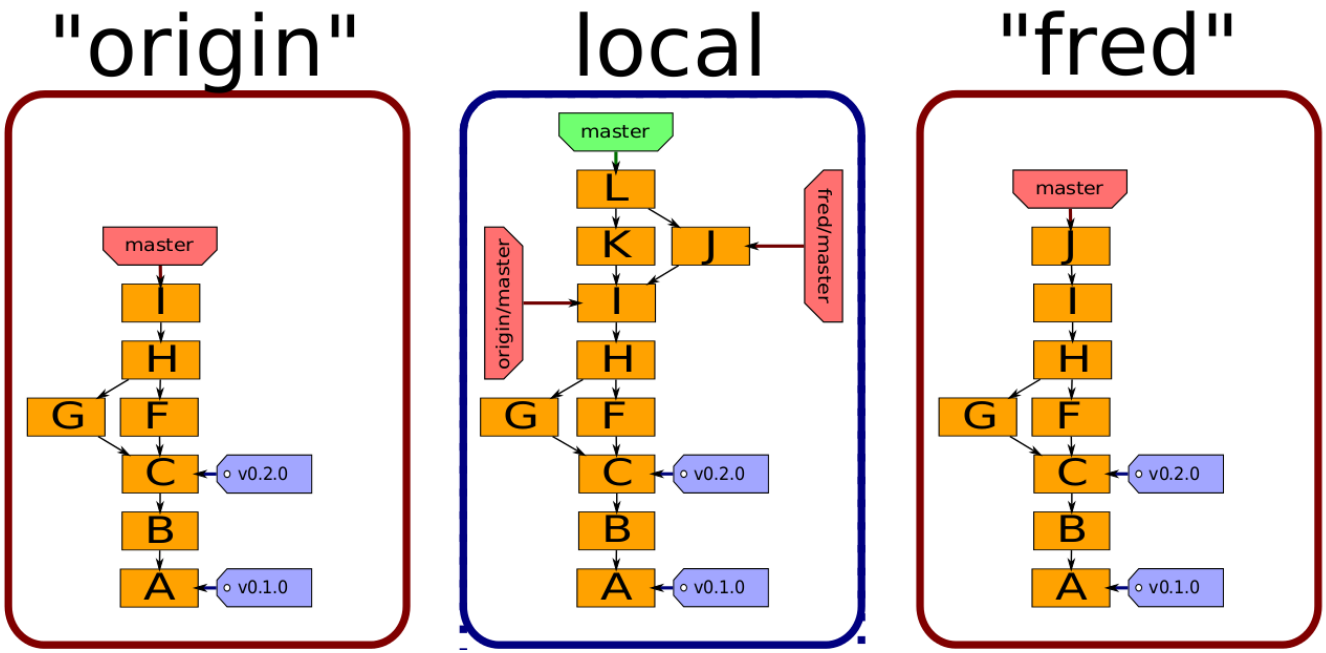
Remotes Example (continued)

Fetch from "fred"



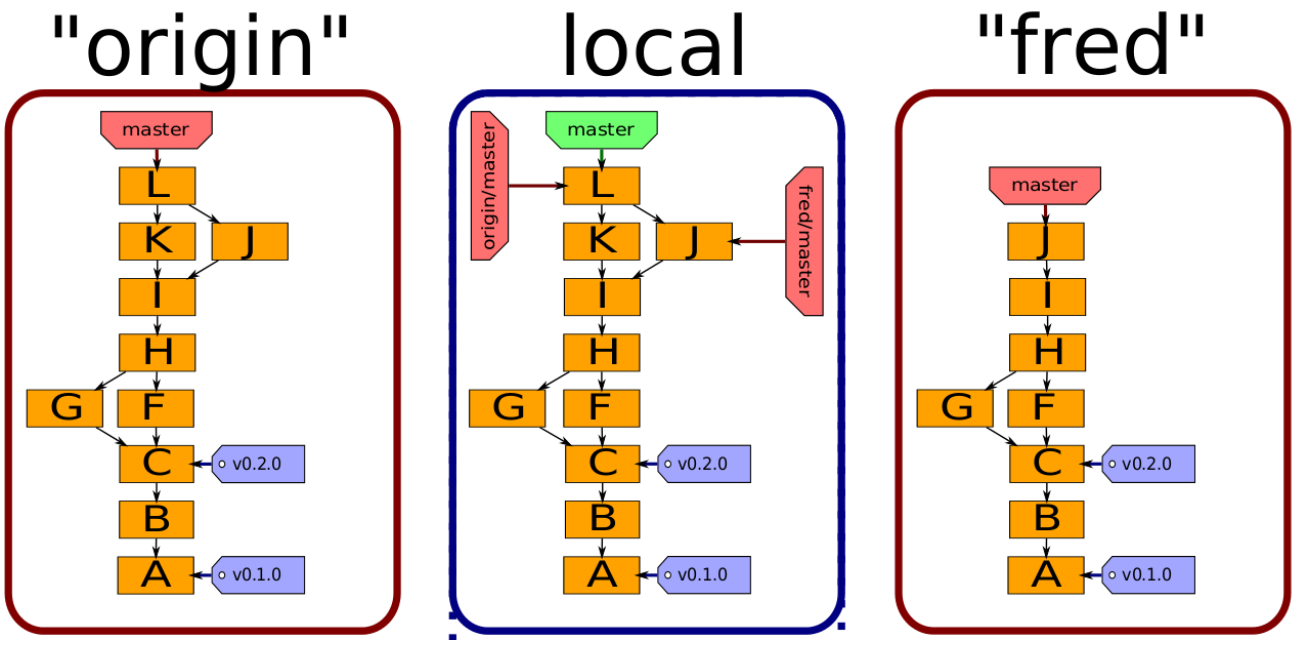
Remotes Example (continued)

Merge in the changes



Remotes Example (continued)

Push changes to "origin"



Challenge Problem

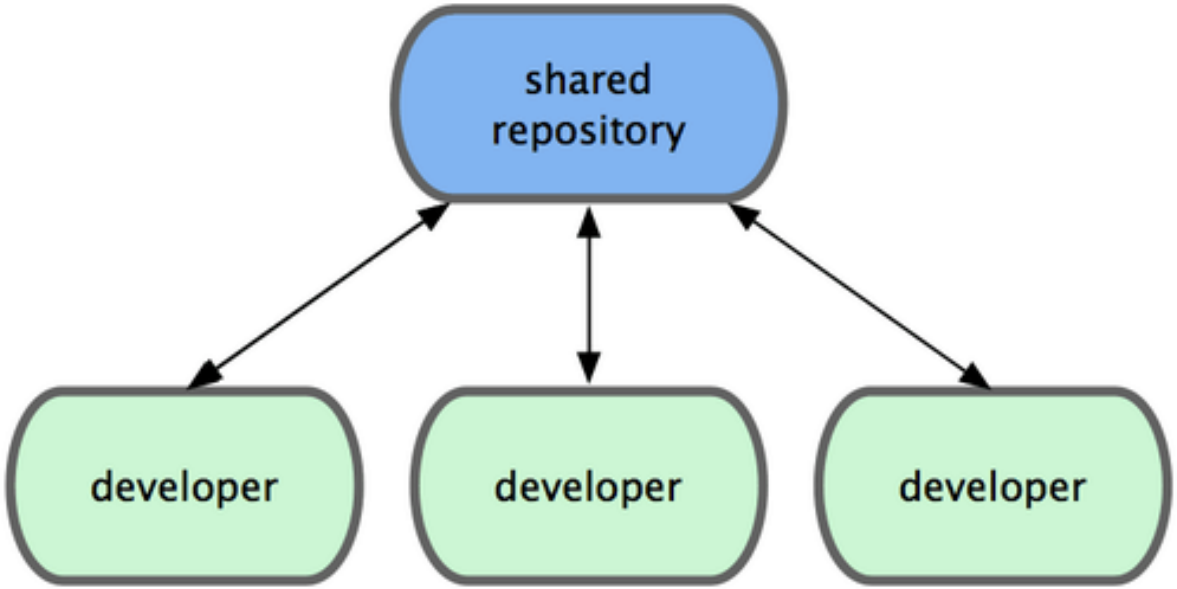
Shape module at `https://github.com/gswg/example.git`

- Clone repository
- Locate and fix bug
- Push fix
  - You may need to fetch and merge with `origin/master`
  - Username: gswg
  - Password: siam2014

Distributed

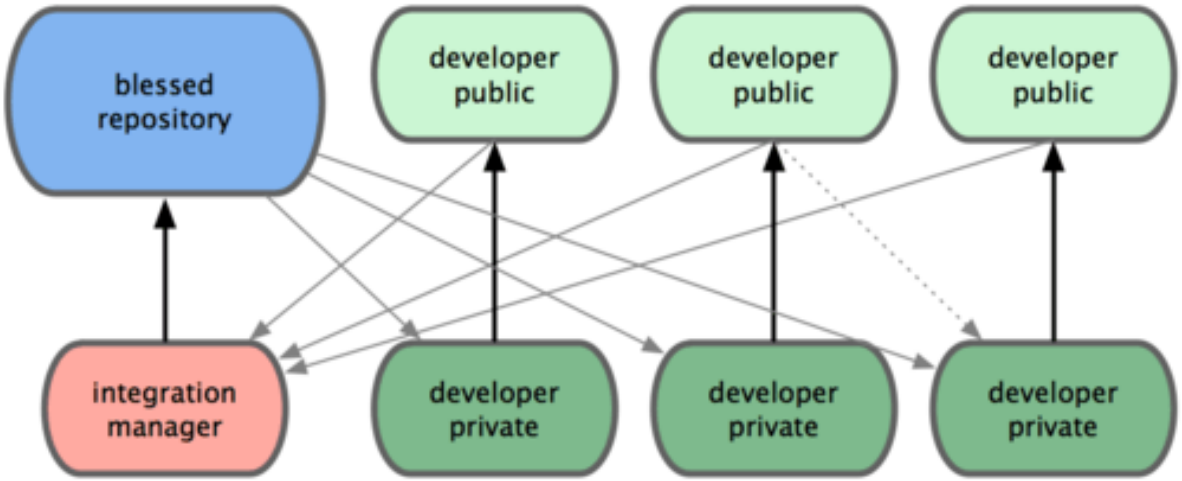
- No central location that keeps track of your data (no single place is more important than another)
- Encourages small commits and frequent merging
- Branches don't affect the main repository and can commit changes without disturbing others
- Work offline
- Rely on a network of trust

Centralized Workflow



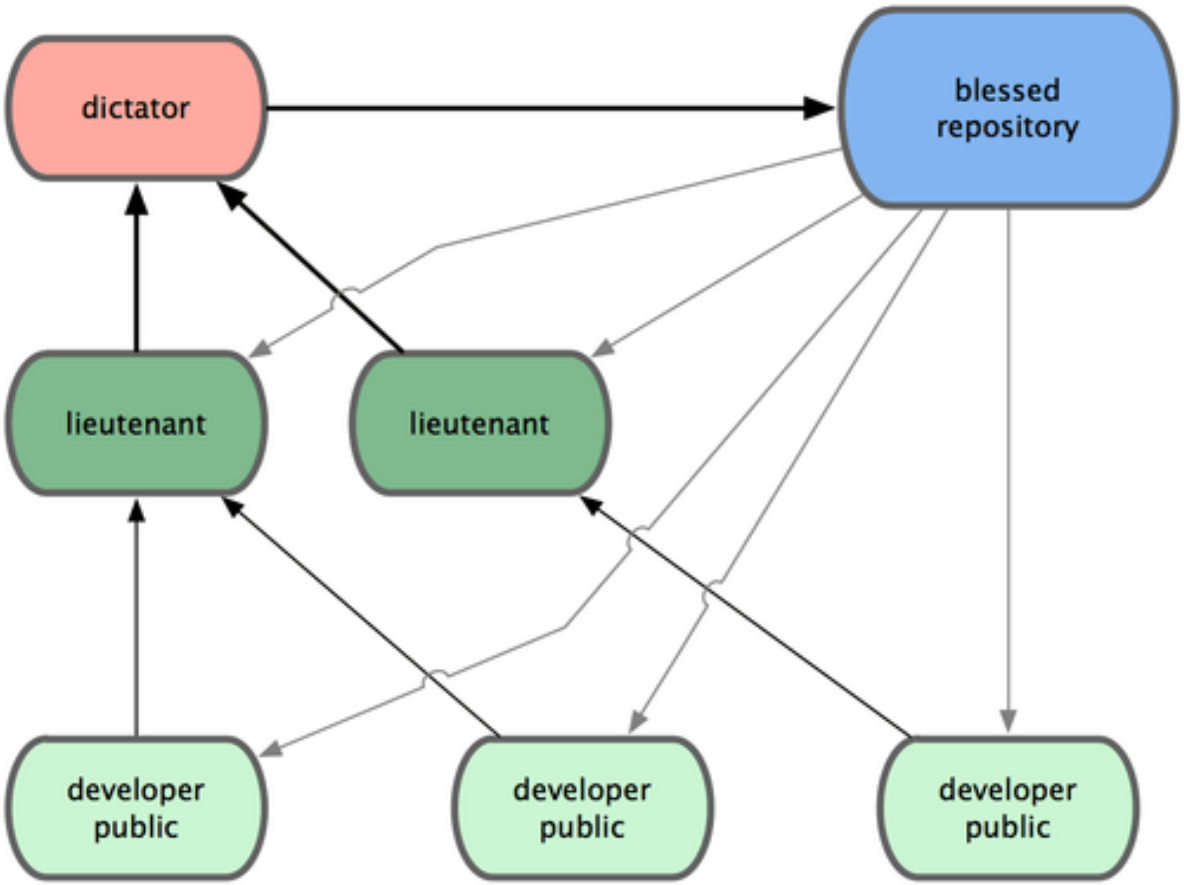
[1]

Integration-Manager Workflow



[1]

Dictator and Lieutenants Workflow



[1]

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

References

[1] Git Book. URL http://git-scm.com/book.

[2] Git From the Bottom Up. URL http://ftp.newartisans.com/pub/git.from.bottom.up.pdf.

[3] Git Magic. URL http://www-cs-students.stanford.edu/~blynn/gitmagic/.

[4] User Manual. URL http://git-scm.com/docs/user-manual.html.

[5] Code School – Try Git. URL http://try.github.io.

[6] Tech Talk: Linus Torvalds on Git. URL http://youtu.be/4XpnKHJAok8.

[7] Mark Lodato. A Visual Git Reference. URL marklodato.github.io/visual-git-guide/.

[8] Bart Trojanowski. Bart's Blog–Intro to Git. URL www.junkie.net/~bart/blog.