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/Updating
- Getting Information
- Branching/Remotes

4 Sharing
- Distributed Workflows
- 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

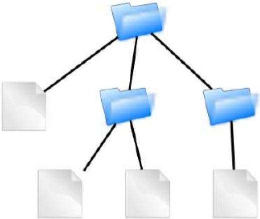
Implications:

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

Components

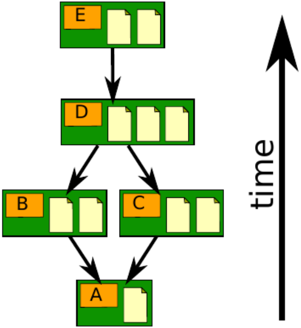
VCS Components (Working Tree)

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



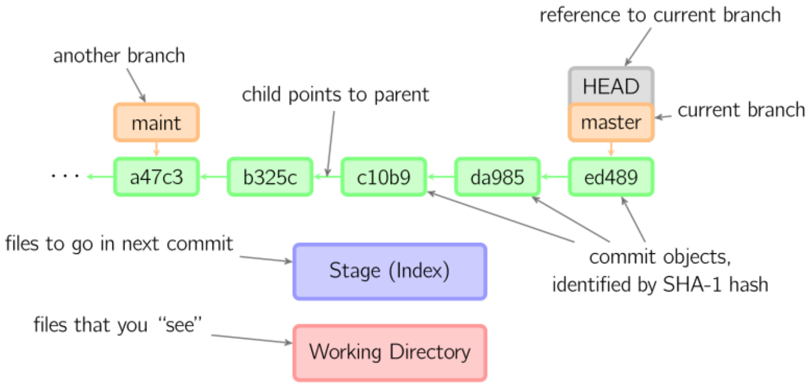
VCS Components (Repository)

- Files
- Commits
- Ancestry



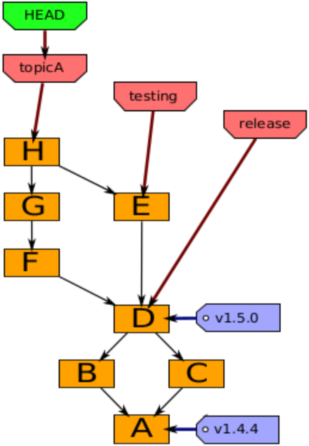
VCS Components (References)

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



[7]

VCS Components (Example Graph)



Operations

VCS Operations

Bootstrap

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

Modify

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

Information

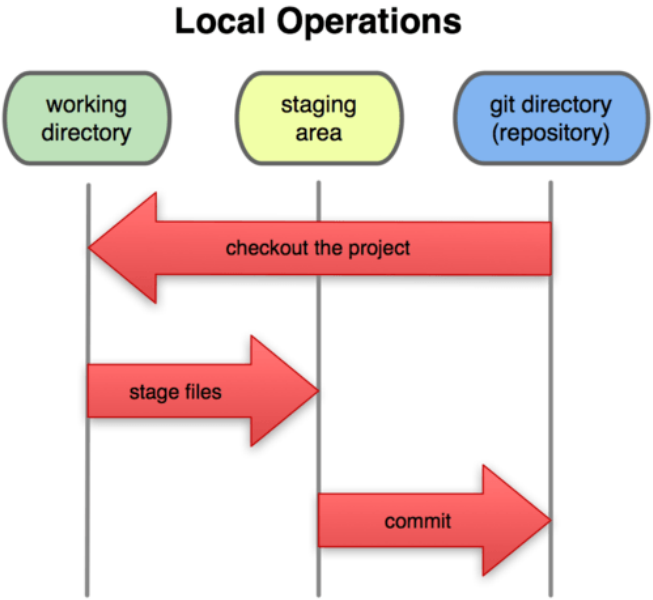
- `status`
- `diff`
- `log`

Reference

- `tag`
- `branch`

Sharing work, backing it up

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


Local Operations
working directory
staging area
git directory (repository)
checkout the project
stage files
commit

[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

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>..<until>] [-- <path>]`

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

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
```

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>
```

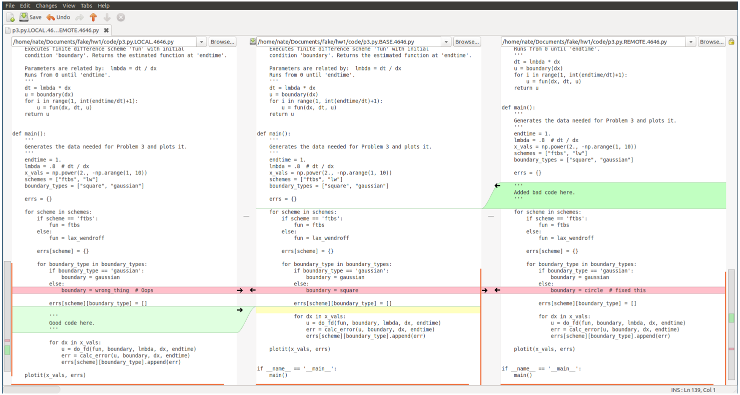
- Incorporates changes from the specified branch into the current branch.
- Conflicts may result
- Any conflicts must be resolved before merge is completed

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

Mergetool

`$ git mergetool`

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


File Edit Changes View Tabs Help
Save
Undo
p3.py.LOCAL.46…EMOTE.4646.py
Browse...
Browse...
Browse...
INS : Ln 139, Col 1

/home/nate/Documents/fake/hw1/code/p3.py.LOCAL.4646.py

```
    Executes finite difference scheme 'fun' with initial
    condition 'boundary'. Returns the estimated function at 'endtime'.

    Parameters are related by:  lmbda = dt / dx
    Runs from 0 until 'endtime'.
    '''
    dt = lmbda * dx
    u = boundary(dx)
    for i in range(1, int(endtime/dt)+1):
        u = fun(dx, dt, u)
    return u


def main():
    '''
    Generates the data needed for Problem 3 and plots it.
    '''
    endtime = 1.
    lmbda = .8  # dt / dx
    x_vals = np.power(2., -np.arange(1, 10))
    schemes = ["ftbs", "lw"]
    boundary_types = ["square", "gaussian"]

    errs = {}

    for scheme in schemes:
        if scheme == 'ftbs':
            fun = ftbs
        else:
            fun = lax_wendroff

        errs[scheme] = {}

        for boundary_type in boundary_types:
            if boundary_type == 'gaussian':
                boundary = gaussian
            else:
                boundary = wrong_thing  # Oops

            errs[scheme][boundary_type] = []

            '''
            Good code here.
            '''

            for dx in x_vals:
                u = do_fd(fun, boundary, lmbda, dx, endtime)
                err = calc_error(u, boundary, dx, endtime)
                errs[scheme][boundary_type].append(err)

    plotit(x_vals, errs)
```

/home/nate/Documents/fake/hw1/code/p3.py.BASE.4646.py

```
    Executes finite difference scheme 'fun' with initial
    condition 'boundary'. Returns the estimated function at 'endtime'.

    Parameters are related by:  lmbda = dt / dx
    Runs from 0 until 'endtime'.
    '''
    dt = lmbda * dx
    u = boundary(dx)
    for i in range(1, int(endtime/dt)+1):
        u = fun(dx, dt, u)
    return u


def main():
    '''
    Generates the data needed for Problem 3 and plots it.
    '''
    endtime = 1.
    lmbda = .8  # dt / dx
    x_vals = np.power(2., -np.arange(1, 10))
    schemes = ["ftbs", "lw"]
    boundary_types = ["square", "gaussian"]

    errs = {}

    for scheme in schemes:
        if scheme == 'ftbs':
            fun = ftbs
        else:
            fun = lax_wendroff

        errs[scheme] = {}

        for boundary_type in boundary_types:
            if boundary_type == 'gaussian':
                boundary = gaussian
            else:
                boundary = square

            errs[scheme][boundary_type] = []

            for dx in x_vals:
                u = do_fd(fun, boundary, lmbda, dx, endtime)
                err = calc_error(u, boundary, dx, endtime)
                errs[scheme][boundary_type].append(err)

    plotit(x_vals, errs)


if __name__ == '__main__':
    main()
```

/home/nate/Documents/fake/hw1/code/p3.py.REMOTE.4646.py

```
    Runs from 0 until 'endtime'.
    '''
    dt = lmbda * dx
    u = boundary(dx)
    for i in range(1, int(endtime/dt)+1):
        u = fun(dx, dt, u)
    return u


def main():
    '''
    Generates the data needed for Problem 3 and plots it.
    '''
    endtime = 1.
    lmbda = .8  # dt / dx
    x_vals = np.power(2., -np.arange(1, 10))
    schemes = ["ftbs", "lw"]
    boundary_types = ["square", "gaussian"]

    errs = {}

    '''
    Added bad code here.
    '''
    for scheme in schemes:
        if scheme == 'ftbs':
            fun = ftbs
        else:
            fun = lax_wendroff

        errs[scheme] = {}

        for boundary_type in boundary_types:
            if boundary_type == 'gaussian':
                boundary = gaussian
            else:
                boundary = circle  # fixed this

            errs[scheme][boundary_type] = []

            for dx in x_vals:
                u = do_fd(fun, boundary, lmbda, dx, endtime)
                err = calc_error(u, boundary, dx, endtime)
                errs[scheme][boundary_type].append(err)

    plotit(x_vals, errs)


if __name__ == '__main__':
    main()
```

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

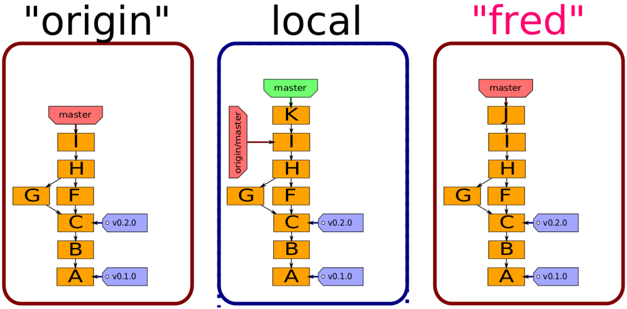
Git Naming–Disambiguation

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`

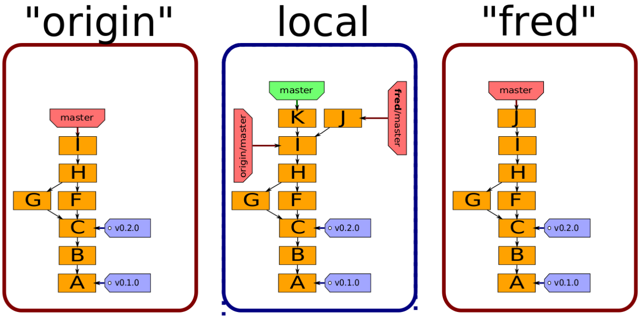
Remotes Example

“fred” cannot push to “origin”



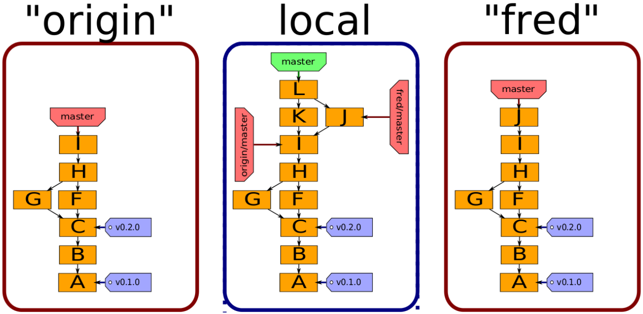
Remotes Example (continued)

Fetch from "fred"



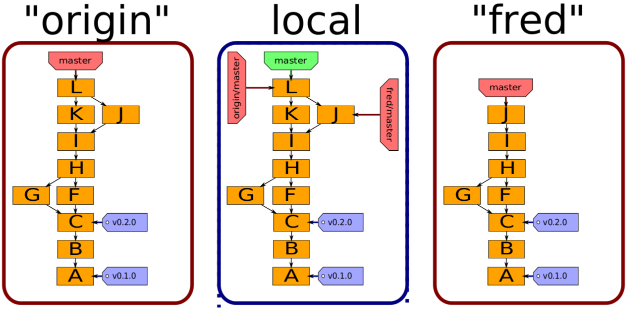
Remotes Example (continued)

Merge in the changes



Remotes Example (continued)

Push changes to “origin”



Challenge Problem

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

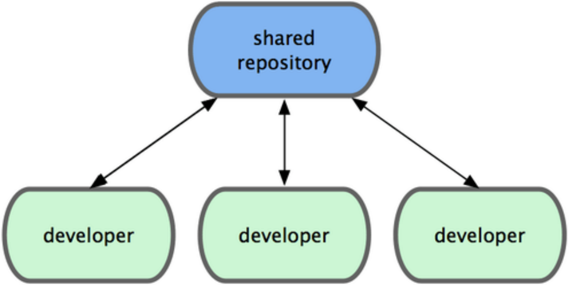
- Clone repository
- Locate and fix bug
- Push fix
  - You may need to fetch and merge with `origin/master`
  - Username: gswg
  - Password: siam2014

Sharing

Distributed

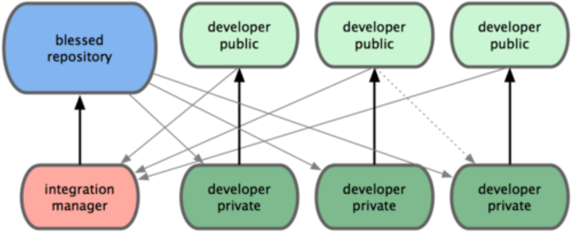
- No central location that keeps track of your data (no single place is more important than another)
- Encourages small commits and frequent merging
- Branches don’t affect the main repository and can commit changes without disturbing others
- Work offline
- Rely on a network of trust

Centralized Workflow



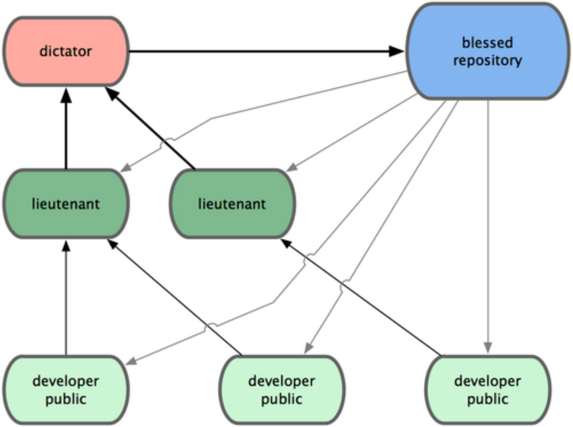
[1]

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.