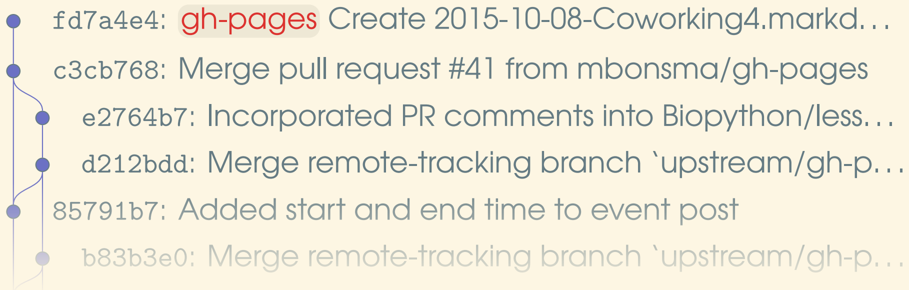
An Intermediate Look at Git + GitHub

U of T Scientific Coders

University of Toronto

October 1, 2015



```
fd7a4e4: gh-pages Create 2015-10-08-Coworking4.markd...
c3cb768: Merge pull request #41 from mbonsma/gh-pages
  e2764b7: Incorporated PR comments into Biopython/less...
  d212bdd: Merge remote-tracking branch `upstream/gh-p...
85791b7: Added start and end time to event post
  b83b3e0: Merge remote-tracking branch `upstream/gh-p...
```

Outline

Section 1

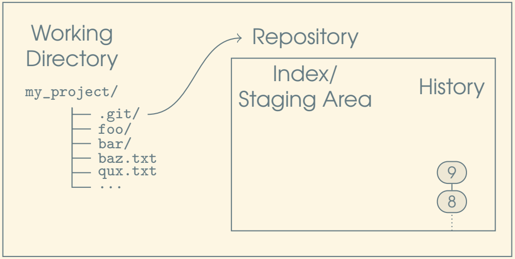
Review

Review

- Configure your git client (`git config user.name` + `user.email`)
- Create a git repository (`git init`)

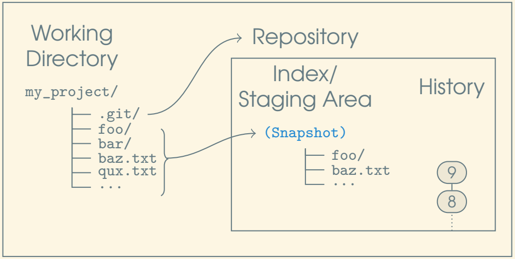
Review

- Configure your git client (`git config user.name` + `user.email`)
- Create a git repository (`git init`)



Review

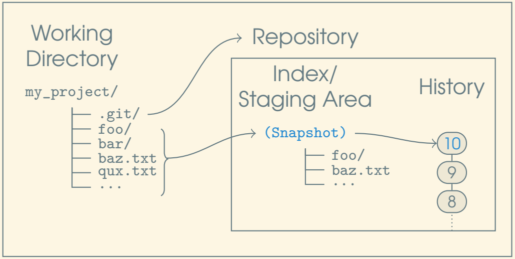
- Configure your git client (`git config user.name` + `user.email`)
- Create a git repository (`git init`)



- Start tracking a file with git (`git add`)

Review

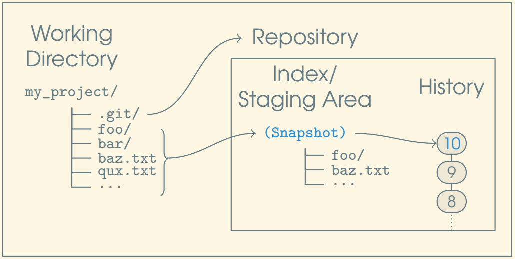
- Configure your git client (`git config user.name` + `user.email`)
- Create a git repository (`git init`)



- Start tracking a file with git (`git add`)
- Commit changes to the history (`git commit`)

Review

- Configure your git client (`git config user.name` + `user.email`)
- Create a git repository (`git init`)



- Start tracking a file with git (`git add`)
- Commit changes to the history (`git commit`)
- Check what's going on (`git status`)
- Compare a file with the one in the history (`git diff`)
- Look into your history (`git log`)

Section 2

Viewing History

Viewing History

Viewing the log allows you to “see” history:

- `git log`

Viewing History

Viewing the log allows you to “see” history:

- `git log`
- `git log <start>..<end>`
- `git log -- <file>`
- `git log --oneline`
- `git log --graph`
- `git log --graph --decorate`

Viewing History

Viewing the log allows you to “see” history:

- `git log`
- `git log <start>..<end>`
- `git log -- <file>`
- `git log --oneline`
- `git log --graph`
- `git log --graph --decorate`
- `git blame <file>`

Viewing History

Viewing the log allows you to “see” history:

- `git log`
- `git log <start>..<end>`
- `git log -- <file>`
- `git log --oneline`
- `git log --graph`
- `git log --graph --decorate`
- `git blame <file>`
- `gitk` + `gitg` + other viewers

Section 3

Branching

Branches

What are branches?

- Divergent commits (two commits with the same parent) could be considered “virtual” branches

Branches

What are branches?

- Divergent commits (two commits with the same parent) could be considered "virtual" branches
- Branches are simply a named pointer to a commit

Branches

What are branches?

- Divergent commits (two commits with the same parent) could be considered “virtual” branches
- Branches are simply a named pointer to a commit
- Branches automatically *move forward* as commits are made

Branches

What are branches?

- Divergent commits (two commits with the same parent) could be considered “virtual” branches
- Branches are simply a named pointer to a commit
- Branches automatically *move forward* as commits are made

Why use them?

Branches

What are branches?

- Divergent commits (two commits with the same parent) could be considered “virtual” branches
- Branches are simply a named pointer to a commit
- Branches automatically *move forward* as commits are made

Why use them?

- They’re cheap! Just pointers. No heavy changes, e.g., an extra directory in svn.

Branches

What are branches?

- Divergent commits (two commits with the same parent) could be considered “virtual” branches
- Branches are simply a named pointer to a commit
- Branches automatically *move forward* as commits are made

Why use them?

- They’re cheap! Just pointers. No heavy changes, e.g., an extra directory in svn.
- To keep experimental work apart

Branches

What are branches?

- Divergent commits (two commits with the same parent) could be considered "virtual" branches
- Branches are simply a named pointer to a commit
- Branches automatically *move forward* as commits are made

Why use them?

- They're cheap! Just pointers. No heavy changes, e.g., an extra directory in svn.
- To keep experimental work apart
- To separate trials

Branches

What are branches?

- Divergent commits (two commits with the same parent) could be considered “virtual” branches
- Branches are simply a named pointer to a commit
- Branches automatically *move forward* as commits are made

Why use them?

- They’re cheap! Just pointers. No heavy changes, e.g., an extra directory in svn.
- To keep experimental work apart
- To separate trials
- To ease collaboration

Branches

Managing branches:

- `git branch <name> [commit]`
- `git branch -d <name>`
- `git branch [-l]`

Branches

Managing branches:

- `git branch <name> [commit]`
- `git branch -d <name>`
- `git branch [-l]`

Switching branches:

- `git checkout <branch name>`
- `git checkout -b <branch name> [commit]` — Create and switch in one go

Branches

Managing branches:

- `git branch <name> [commit]`
- `git branch -d <name>`
- `git branch [-l]`

Switching branches:

- `git checkout <branch name>`
- `git checkout -b <branch name> [commit]` — Create and switch in one go

Merging branches:

- `git merge <other branch name>`
- `git merge --ff-only <other branch name>`
- `git merge --no-ff <other branch name>`

Section 4

Collaborating with Others

Clones and Remotes

Clones are complete copies of a repository’s history (i.e., excluding the index and working directory)

Clones and Remotes

Clones are complete copies of a repository's history (i.e., excluding the index and working directory)

- `git clone <URI>`

Clones and Remotes

Clones are complete copies of a repository's history (i.e., excluding the index and working directory)

- `git clone <URI>`

Remotes are *pointers* to other clones

Clones and Remotes

Clones are complete copies of a repository's history (i.e., excluding the index and working directory)

- `git clone <URI>`

Remotes are *pointers* to other clones

- `git remote [-v]`
- `git remote add <name> <URI>`
- `git remote rm <name>`
- Local branches can *track* remote branches
  `git branch -u <remote branch> <local branch>`

Clones and Remotes

Clones are complete copies of a repository's history (i.e., excluding the index and working directory)

- `git clone <URI>`

Remotes are *pointers* to other clones

- `git remote [-v]`
- `git remote add <name> <URI>`
- `git remote rm <name>`
- Local branches can *track* remote branches
  `git branch -u <remote branch> <local branch>`

You are responsible for syncing

- `git push [<remote>] [<branch>]`
- `git fetch [<remote>]`
- `git pull [<remote>]` — `fetch` + `merge`

GitHub Example

Section 5

Advanced Topics

Advanced Topics

- `git add --patch`

Advanced Topics

- `git add --patch`
- `git rebase`