

Getting Git



Intros

- Name
- Job/Role
- Length of time at Rackspace
- What you hope to get from this class
- Previous experience with version control tools

Learning Objectives

- Learn what version control is
- Recognize what Git and Github are and how they are used
- Practice day to day Git usage
- Collaborate online through GitHub

What is Version Control

- A time machine for files and projects
- Snapshots/checkpoints of the project as a timeline
- Also referred to as revision control

Why use Version Control?

Return to Zero

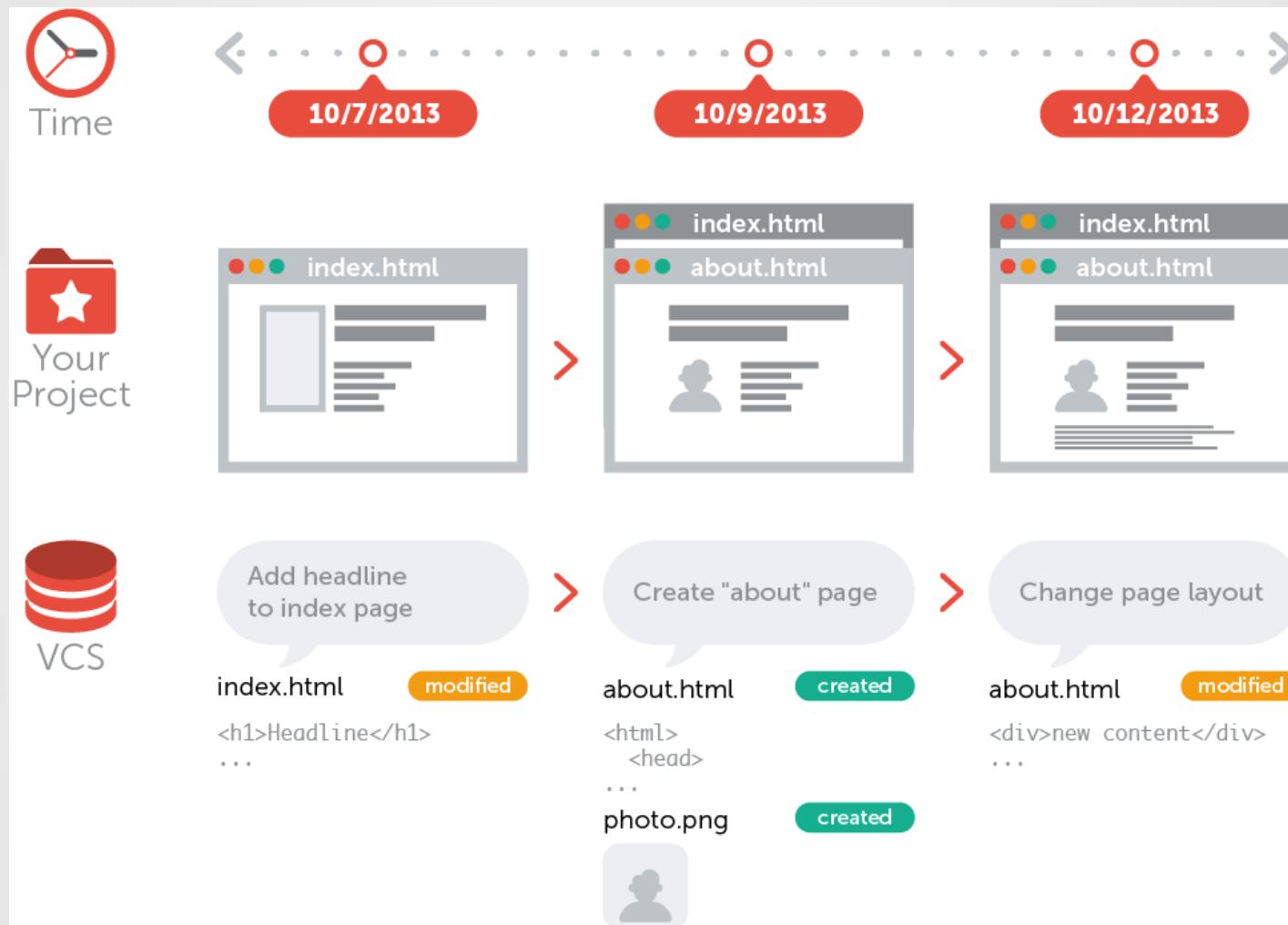


EEWeb.com

What should we use Version Control for?

- Different kinds of files
 - Text files
 - Scripts, config files, software, etc.
 - Binary files (with caveats)
 - Images / logos (with caveats)
- Files that are shared with others
 - Anything that can be changed by multiple folks
- Managing the flow of changes
 - Deploying known versions of files

What is Version Control



Graphic from <http://www.git-tower.com/learn/git/ebook/command-line/basics/what-is-version-control>

Nomenclature

- Version control
- Source control
- Revision control

Use Case 1:

- Multiple people (team members) working on the same stuff

Use Case 2:

- Keeping track of changes as you manage a system

Use Case 3:

- Tracking / managing common configs across multiple systems

War Stories

Work Flow

- Get / open a repository
 - Pull down changes shared with you by others
- Create a branch for your work
- Do some work
 - Check in (commit) changes periodically
- Push your branch to a server to share your work with others
- Merge branches to bring changes into the main "master" branch
- Lather, rinse, repeat

Work Flow

- Each step in that workflow maps to a Git command
 - or a set of Git commands
- You can map your own workflow activities to Git
 - It is really flexible
- We will work on labs that illustrate all these workflow steps

What is Git?

What is Git

- From the web: Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.
- Git is a suite of tools for you to implement version control
 - The core stuff is command line
 - GUIs are available (more on that later)

Installing Git

- Git is available for all the major OSes
 - Linux, Windows, OS X
- <http://git-scm.com/downloads>
- We will experience this in the upcoming lab

What is a repository?

- A repository is the place that stores all the data for your version controlled directory
 - database for all the versions, metadata, file contents, etc.
- A repository can live on a local machine, cloud server, or other nebulous device
- Repositories can be cloned and shared

Git Commands - init

- Get / open / create a repository

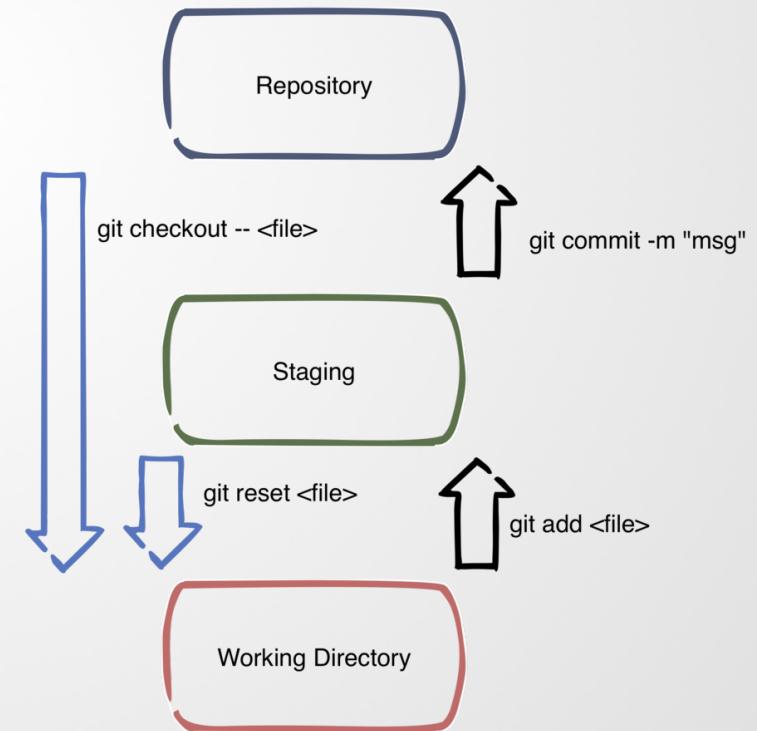
```
git init .
```

Git Commands - add

- Do some work
 - Check in (commit) changes periodically
- To commit your changes first you have to stage them
 - Tell Git which changes you will be committing

```
git add my_file.txt image.jpg
```

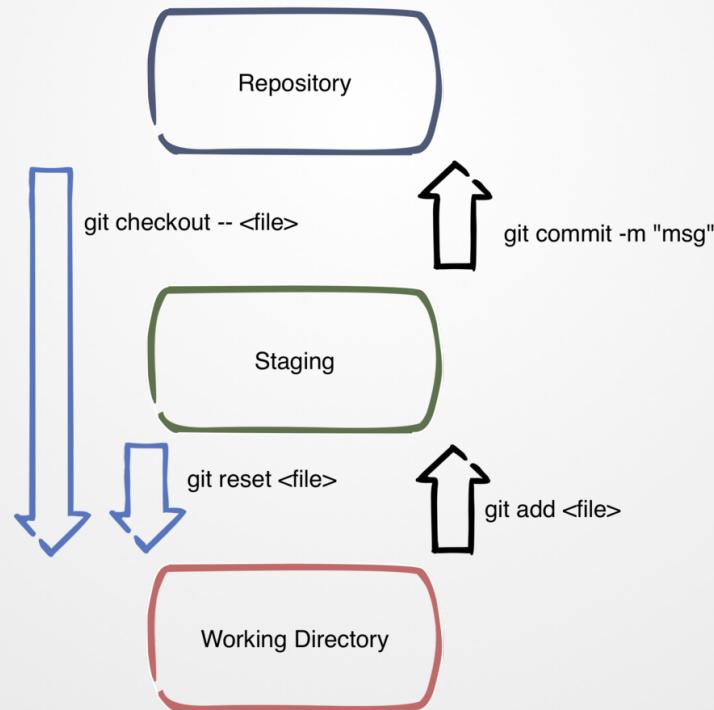
```
git commit -m "my cool comment"
```



Git Commands - rm

- Remove a file (or files) from the repo

```
git rm my_file.txt image.jpg
```



Git Commands - status

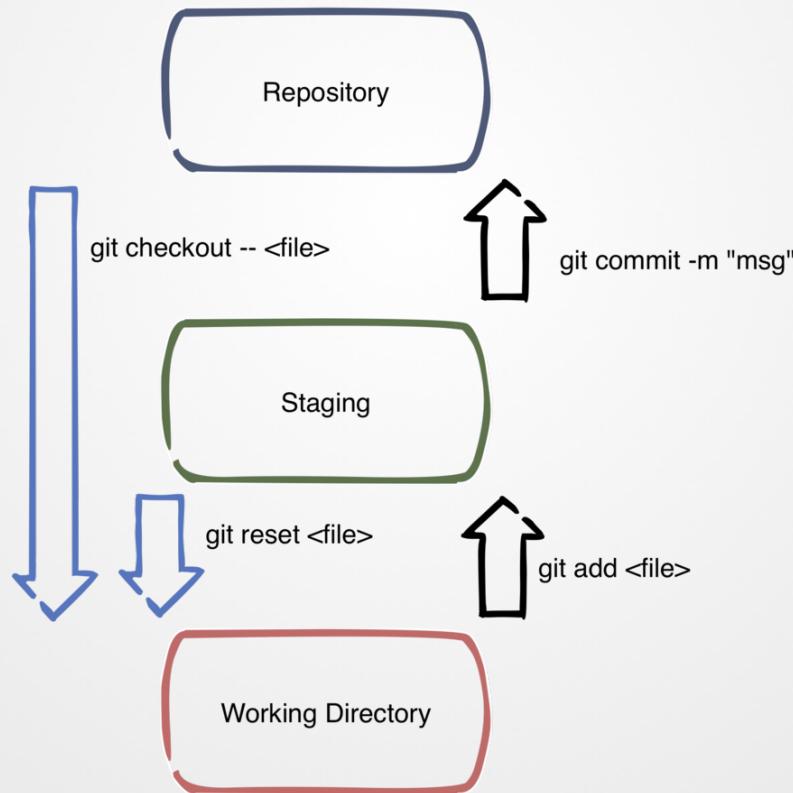
- Find out the status of the repo

```
git status
```

Git Commands - reset

- Unstage a change
- (after an add)

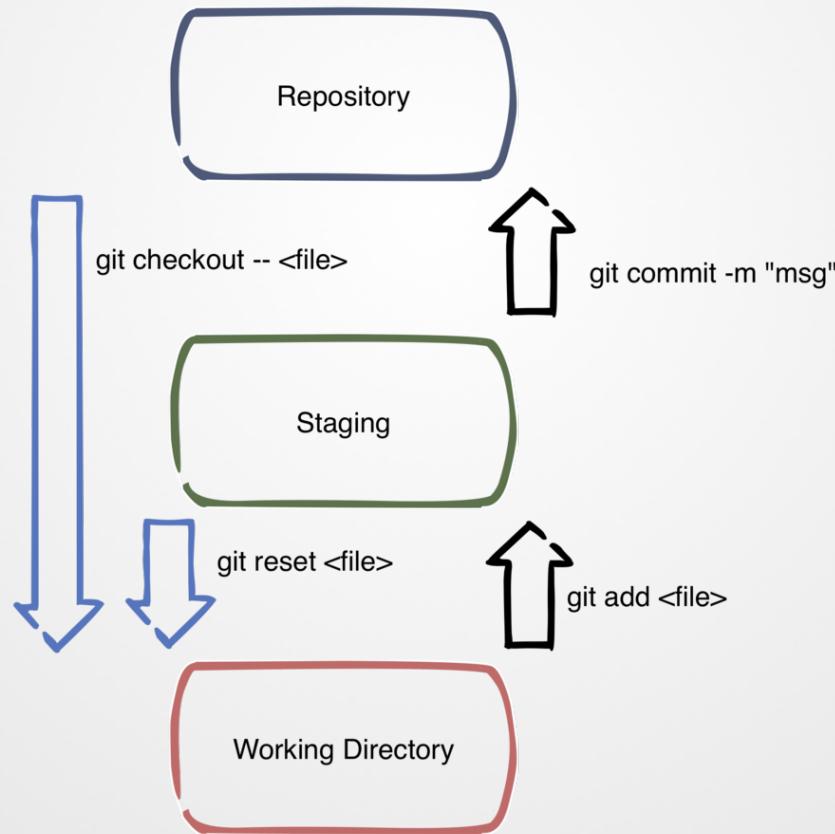
```
git reset new_file.txt
```



Git Commands - undo an edit

- Undo all edits to a file
- **Great caution is advised here**

```
git checkout -- changed_file.txt
```



Lab 1

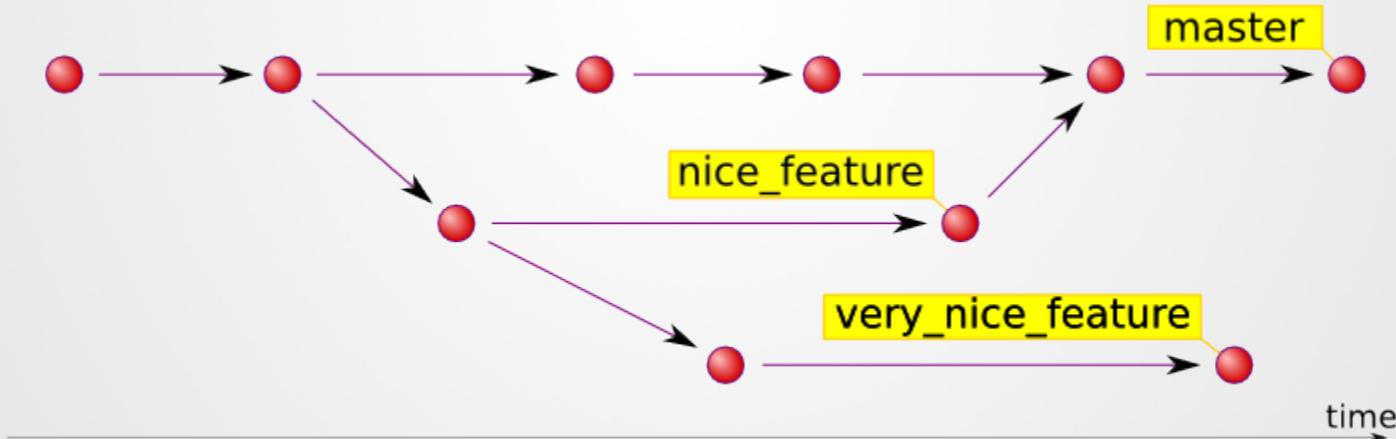
<https://github.com/jeremyprice/GettingGit/blob/master/lab01.md>

Branching, Merging and Conflicts

Workflow Continued

- Working linearly on a project doesn't scale well
- We need to be able to work on parallel sets of changes
- Thankfully, we have branching and merging

Branching and Merging Illustrated



Branching

- Creating a branch called nice_feature

```
git branch nice_feature
```

- Make that branch the active branch

```
git checkout nice_feature
```

Branching

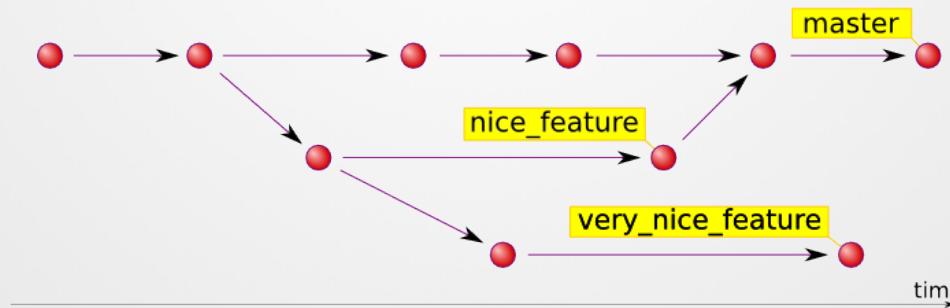
- Shortcut: Creating a branch called nice_feature and make it active

```
git checkout -b nice_feature
```

Merging Branches

- Merging branches asks Git to take the changes from one branch and put them in the other
- Merge changes from nice_feature into master

```
git checkout master  
git merge nice_feature
```



A note about the master branch

- The **master** branch is special
- It is considered the core branch that changes are merged into
- By convention, master is usually a known state of the files in the repository
- It is not the only branch you can merge changes into
- From Git's perspective it is just any old branch

Conflicts

- Merge conflicts happen when you try to merge changes that overlap from one branch to another
- Example: in your README file
 - Branch A: Hello world!
 - Branch B: Goodbye world!
- Git can't tell which one it should go with, so it tells you to pick

Lab 2

<https://github.com/jeremyprice/GettingGit/blob/master/lab02.md>



GitHub for Collaboration

GitHub

- GitHub is a web-based Git repository hosting service
 - Centralized repository
- Social coding features
 - pull requests
 - documentation (markdown)
 - wikis
 - issue tracking
 - small websites
 - history browsing
 - code reviews
- Example: <https://github.com/jeremyprice/GettingGit>

Lab 3

<https://github.com/jeremyprice/GettingGit/blob/master/lab03.md>

Other Topics

GUI Clients

- (Mac) Tower - <http://www.git-tower.com>
- (Mac) GitHub - <http://mac.github.com>
- (Win) GitHub - <http://windows.github.com>
- (Mac, Win) SourceTree - <https://www.sourcetreeapp.com/>
- (Linux) gitg - <https://wiki.gnome.org/Apps/Gitg/>

Good list of GUI apps:

- <http://git-scm.com/downloads/guis>
- <https://git.wiki.kernel.org/index.php/InterfacesFrontendsAndTools>

Demo of Git GUI

SourceTree

Further reading

- Git Pro book - <http://git-scm.com/>
- Git Immersion tutorial - <http://gitimmersion.com/>
- Try Git - <https://try.github.io/>
- Learn Git branching - <http://pcottle.github.io/learnGitBranching/>
- Git Tower - <http://www.git-tower.com/learn/git/ebook/>
- Getting started with GitHub - <https://www.youtube.com/watch?v=TAObtTxBUzk>
- Good advice about Git Style - <https://github.com/agis-/git-style-guide>

Desk Reference Cheat Sheet

Git Cheat Sheet

<http://git.or.cz/>

Remember: `git command --help`

Global Git configuration is stored in `$HOME/.gitconfig` (`git config --help`)

Create

From existing data

```
cd ~/projects/myproject
git init
git add .
```

From existing repo

```
git clone ~/existing/repo ~/new/repo
git clone git://host.org/project.git
git clone ssh://you@host.org/proj.git
```

Show

Files changed in working directory
`git status`

Changes to tracked files
`git diff`

What changed between \$ID1 and \$ID2
`git diff $id1 $id2`

History of changes
`git log`

History of changes for file with diffs
`git log -p $file | dir/ectory/`

Who changed what and when in a file
`git blame $file`

A commit identified by \$ID
`git show $id`

A specific file from a specific \$ID
`git show $id:$file`

All local branches
`git branch`
(star * marks the current branch)

Cheat Sheet Notation

\$id : notation used in this sheet to represent either a commit id, branch or a tag name
\$file : arbitrary file name
\$branch : arbitrary branch name

Concepts

Git Basics

master : default development branch
origin : default upstream repository
HEAD : current branch
HEAD~n : n-th commit
HEAD-4 : the great-great grandparent of HEAD

Revert

Return to the last committed state
`git reset --hard` ⚠ you cannot undo a hard reset

Revert the last commit
`git revert HEAD` Creates a new commit

Revert specific commit
`git revert $id` Creates a new commit

Fix the last commit
`git commit -a -amend` (after editing the broken files)

Checkout the \$id version of a file
`git checkout $id $file`

Branch

Switch to the \$id branch
`git checkout $id`

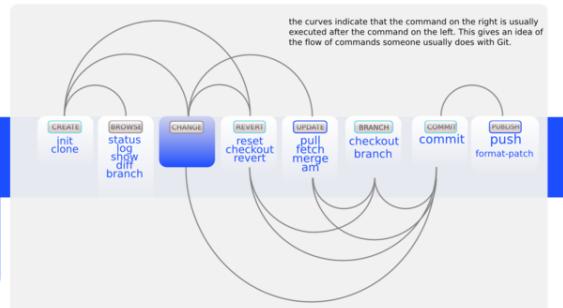
Merge branch1 into branch2
`git checkout $branch2
git merge branch1`

Create branch named \$branch based on the HEAD
`git branch $branch`

Create branch \$new_branch based on branch \$other and switch to it
`git checkout -b $new_branch $other`

Delete branch \$branch
`git branch -d $branch`

Commands Sequence



Update

Fetch latest changes from origin
`git fetch` (but this does not merge them)

Pull latest changes from origin
`git pull` (does a fetch followed by a merge)

Apply a patch that some sent you
`git am -3 patch mbox` (in case of a conflict, resolve and use git am --resolved)

Useful Commands

Finding regressions
`git bisect start` (to start)
`git bisect good $id` (\$id is the last working version)
`git bisect bad $id` (\$id is a broken version)
`git bisect bad/good` (to mark it as bad or good)
`git bisect visualize` (to launch gitk and mark it)
`git bisect reset` (once you're done)

Check for errors and cleanup repository
`git fsck
git gc --prune`

Search working directory for foo()
`git grep "foo()"`

Publish

Commit all your local changes
`git commit -a`

Prepare a patch for other developers
`git format-patch origin`

Push changes to origin
`git push`

Mark a version / milestone
`git tag v1.0`

Resolve Merge Conflicts

To view the merge conflicts
`git diff` (complete conflict diff)
`git diff --base $file` (against base file)
`git diff --ours $file` (against your changes)
`git diff --theirs $file` (against other changes)

To discard conflicting patch
`git reset --hard
git rebase --skip`

After resolving conflicts, merge with
`git add $conflicting_file` (do for all resolved files)
`git rebase --continue`

Carl Ruus
Based on the work of
Svenning Løkken

<http://byte.kde.org/~zrusin/git/git-cheat-sheet-large.png>

Your feedback is important to us

- Please take a moment to fill out a brief evaluation for this class:
- Go to mylearn.rackspace.com
- Scroll down to My Tasks on your myLearn welcome page
- Click on the evaluation link for this class
- Fill out the evaluation then submit

We appreciate your feedback!