Basic Git and GitHub Usage

Anthony J Souza

Topics Covered

- What a VCS is
- Why using a VCS is a good idea
- Git basics, including:
  - How to obtain & install git
  - How to use it for basic, common operations
  - Where to go for more information

What is a Version Control System (VCS)?

- A way to keep track of changes to files (& folders)
- Between multiple authors (developers)
- A record of who did what, when
- Why is provided by commit messages!
- Centralized VCS (Subversion, CVS)
  - Server has the master repo, all commits go to the server
  - CVS → Concurrent Versions System
- Distributed VCS (Git, Mercurial)
  - Server has the master repo, but you have a copy (clone) of the repo on your machine
  - Visual Source Safe, Perforce are some others

Git is (a VCS that is)...

- An open source VCS designed for speed and efficiency
- Better than competing tools
- Created by Linus Torvalds (for managing Linux kernel)
- Your best insurance policy against:
  - Accidental mistakes like deleting work
  - Remembering what you changed, when, why
  - Your hard drive blowing up*

Git is (a VCS that is)...

- Your best insurance policy against:
  - Accidental mistakes like deleting work
  - Remembering what you changed, when, why
  - Your hard drive blowing up*
- The version control solution I recommend to manage code of all types
- Hosted solutions include Github.com (more widespread, more expensive) or Bitbucket.com (cheaper, integrates with Jira)
- or running your own Git server

Installing Git

- Download the software - it's free
  - http://git-scm.com/downloads
  - or on Mac (homebrew), $ brew install git
  - Liniux , sudo apt-get install git, sudo yum install git
- Download a GUI, optional
  - http://git-scm.com/downloads/guis
- Read the manual and/or the book
  - http://git-scm.com/docs
  - http://git-scm.com/book

How does Git compare?

- An open source, distributed version control software designed for speed and efficiency
- Unlike Subversion, Git is distributed.
- This means that you can use Git on your local machine without being connected to the internet.
- Revisions (commits) you make to your local repository are available to you only
- The next time you connect to the internet, push your changes to a remote repository to share them & back them up offsite & off your computer

How does Git compare?

- The distributed nature of Git makes it insanely fast, because most things you do happen on your local machine
- The local nature of Git makes it effortless to create branches to isolate your work
- The local nature of Git makes it possible to coalesce a series of changes (local commits) into a single commit on the remote branch

Understanding Git Workflow

- Obtain a repository
  - Either via git init, or git clone, or if you already have the repo, pull changes!
- Make some edits
  - Use your favorite text editor or source code IDE
    - Most IDEs have Git integration, including NetBeans
  - git tracks changes to binary files too: images, pdf, etc.
    - Less useful though, than text-based files
- Stage your changes
  - using git add
- Commit your work
  - git commit -m "Always write clear commit messages!"
- Push to remote
  - git push remotename localbranch:remotebranch

Key Concepts: Snapshots

- The way git keeps track of your code history
- Essentially records what all your files look like at a given point in time
- You decide when to take a snapshot, and of what files
- Have the ability to go back to visit any snapshot
  - Your snapshots from later on will stay around, too

Key Concepts: Commit

- The act of creating a snapshot
- Can be a noun or verb
  - “I committed code”
  - “I just made a new commit”
- Essentially, a project is made up of a bunch of commits

Key Concepts: Commit

- Commits contain three pieces of information:
- Information about how the files changed from previously
- A reference to the commit that came before it
  - Called the “parent commit”
- A hash code name
  - Will look something like:
- fb2d2ec5069fc6776c80b3ad6b7cbde3cade4e

Key Concepts: Repositories

- Often shortened to ‘repo’
- A collection of all the files and the history of those files
  - Consists of all your commits
  - Place where all your hard work is stored

Key Concepts: Repositories

- Can live on a local machine or on a remote server (GitHub!)
- The act of copying a repository from a remote server is called cloning
- Cloning from a remote server allows teams to work together

Key Concepts: Repositories

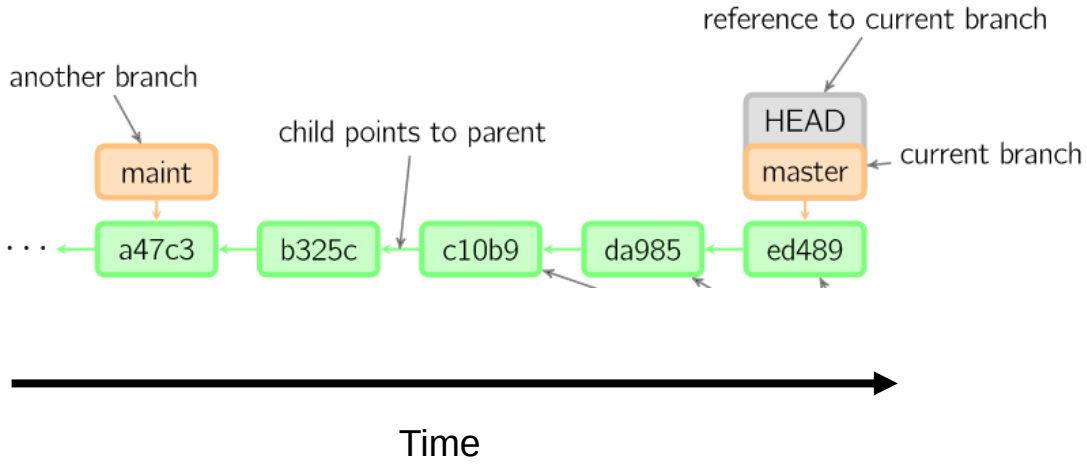
- The process of downloading commits that don’t exist on your machine from a remote repository is called pulling changes
- The process of adding your local changes to the remote repository is called pushing changes

Key Concepts: Branches

- All commits in git live on some branch
- But there can be many, many branches
- The main branch in a project is called the master branch

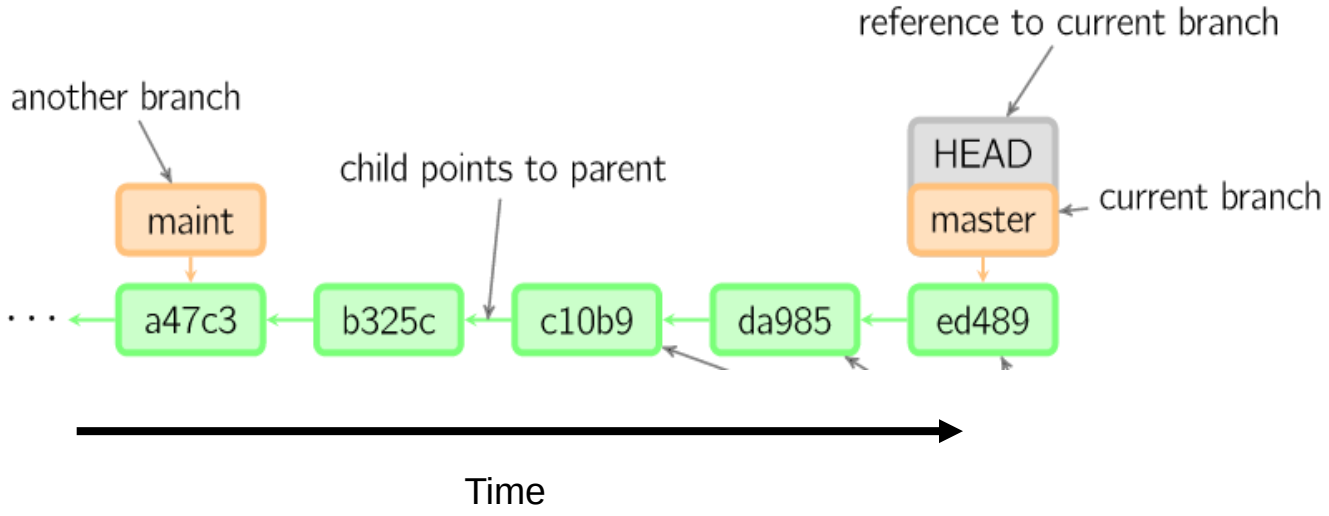
So, what does a typical project look like?

- A bunch of commits linked together that live on some branch, contained in a repository



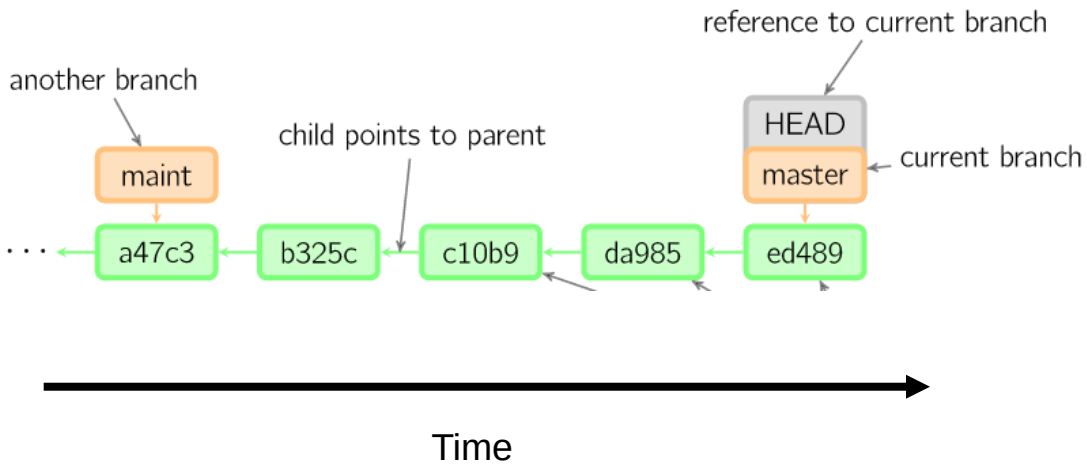
So, what is HEAD?

- A reference to the most recent commit
  - (in most cases – not always true!)



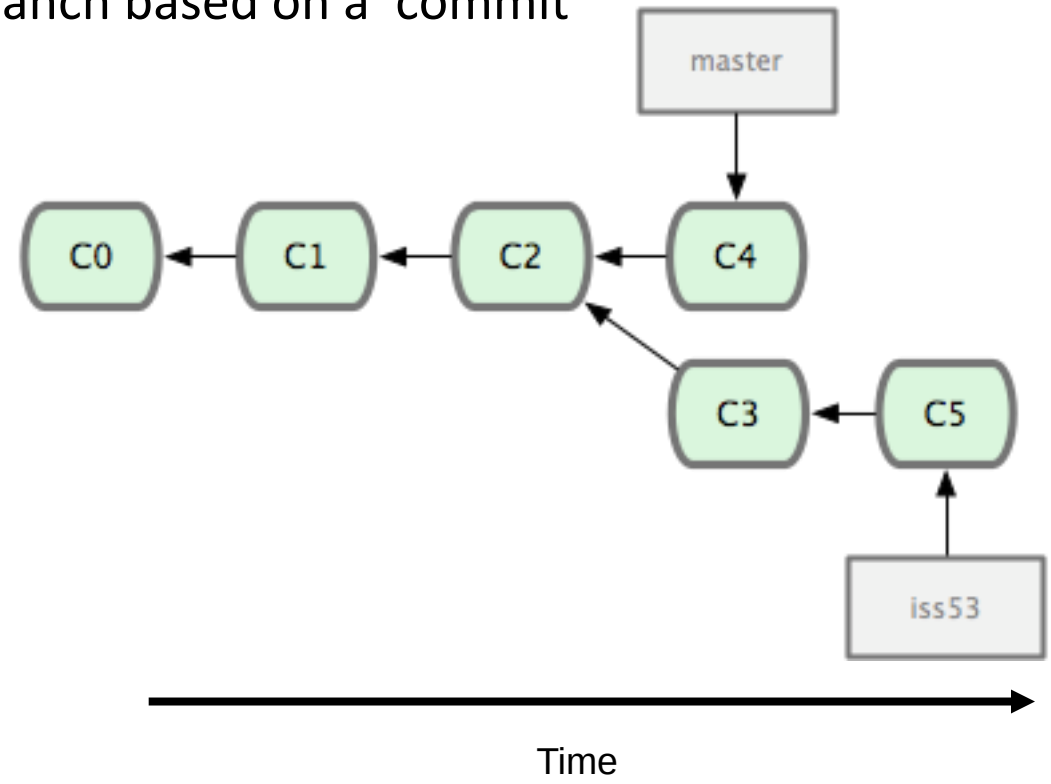
So, what is MASTER?

- The main branch in your project
- Doesn't have to be called master, but almost always is!



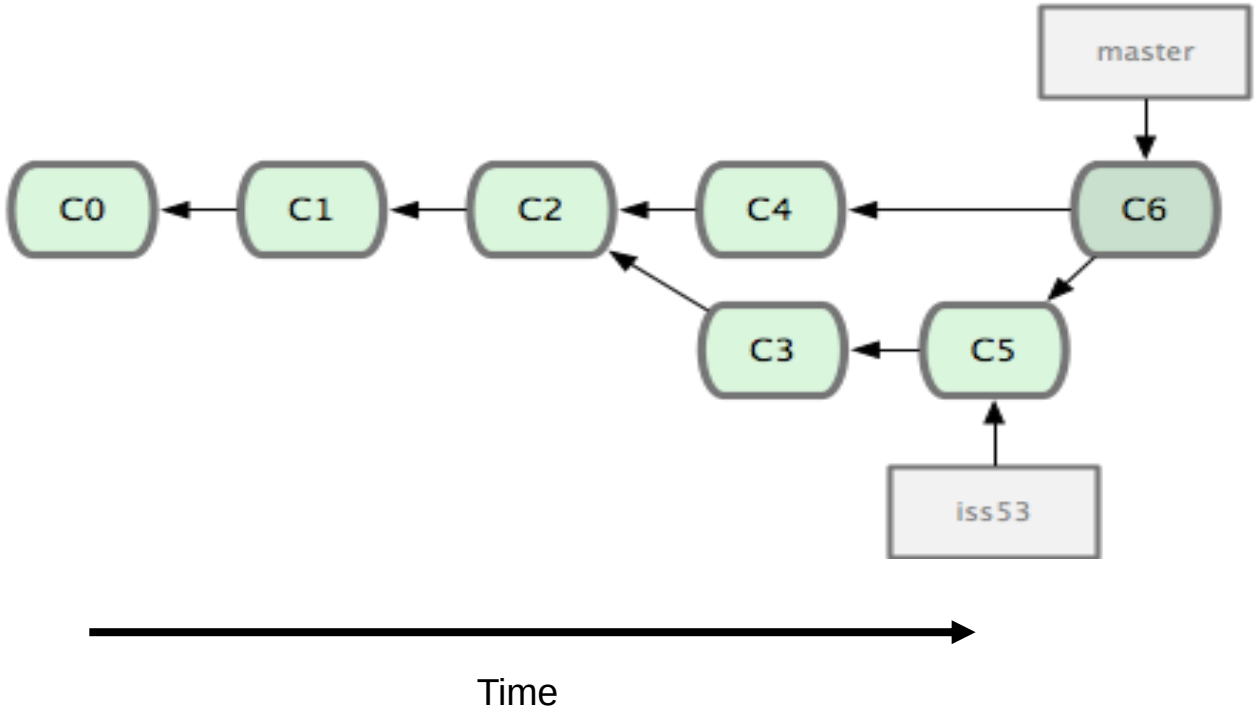
Key Concepts: Branching off of the master branch

- The start of a branch points to a specific commit
- When you want to make any changes to your project you make a new branch based on a commit



Key Concepts: Merging

- Once you're done with your feature, you merge it back into master



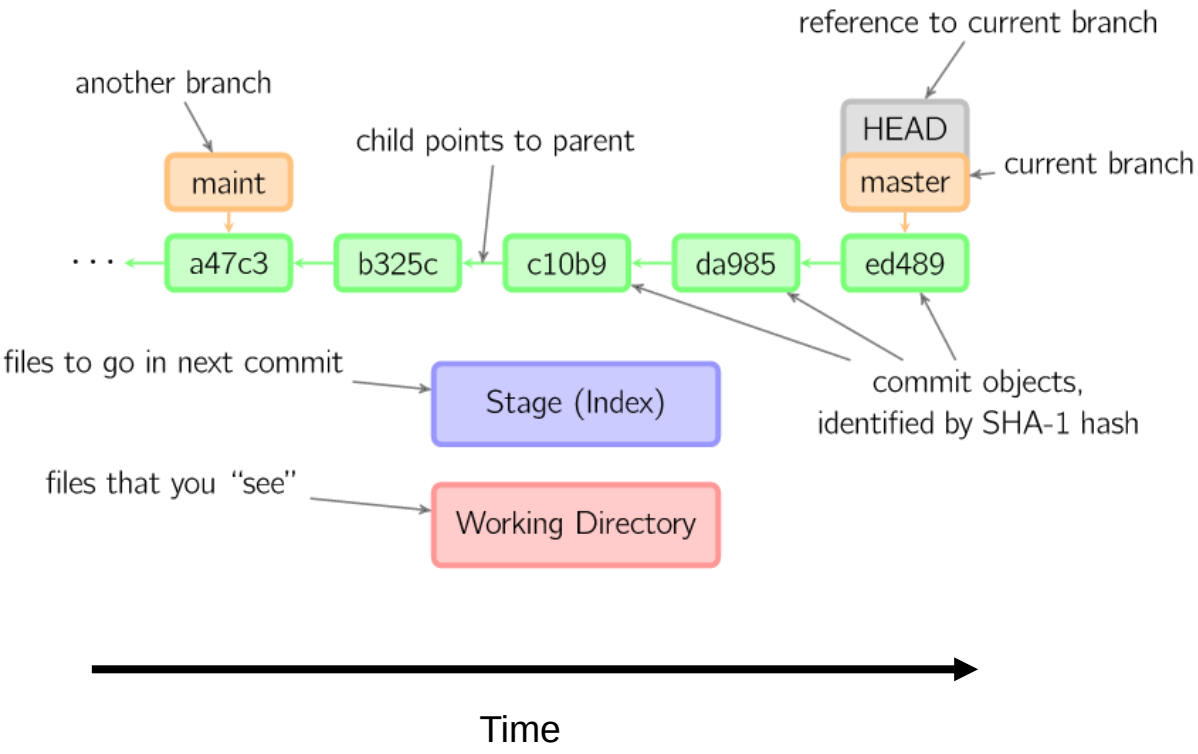
Key Concepts: How do you make a commit anyway?

- There are a lot of ‘states’ and ‘places’ a file can be
- Local on your computer: the ‘working directory’
- When a file is ready to be put in a commit you add it onto the ‘index’ or ‘staging’
  - Staging is the new preferred term – but you can see both ‘index’ and ‘staging’ being used

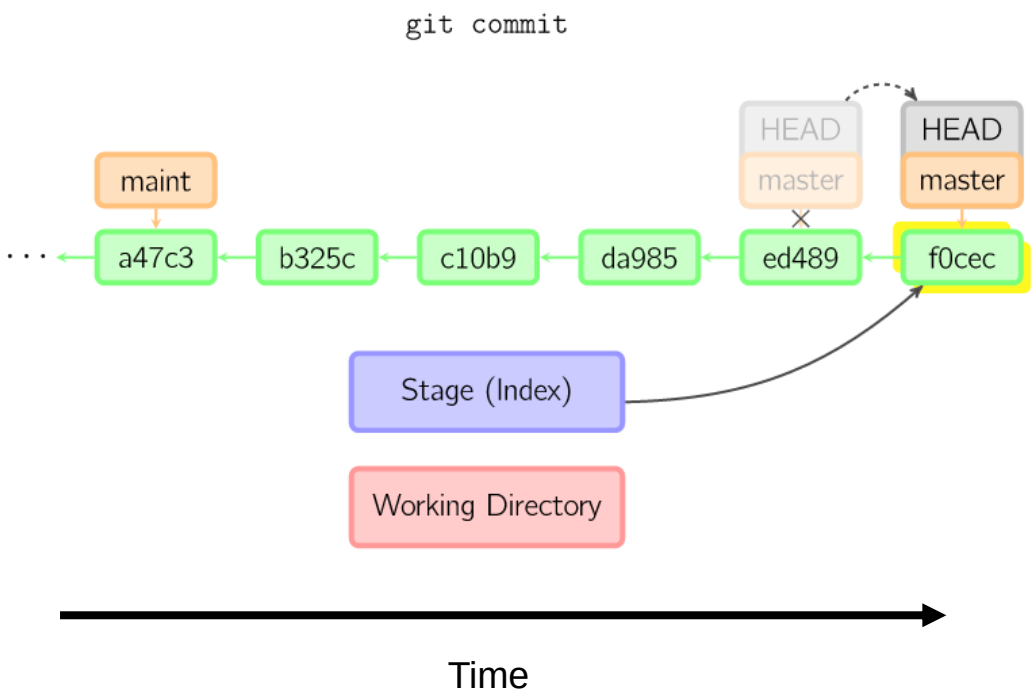
Key Concepts: How do you make a commit anyway?

- The process:
  - Make some changes to a file
  - Use the ‘git add’ command to put the file onto the staging environment
  - Use the ‘git commit’ command to create a new commit

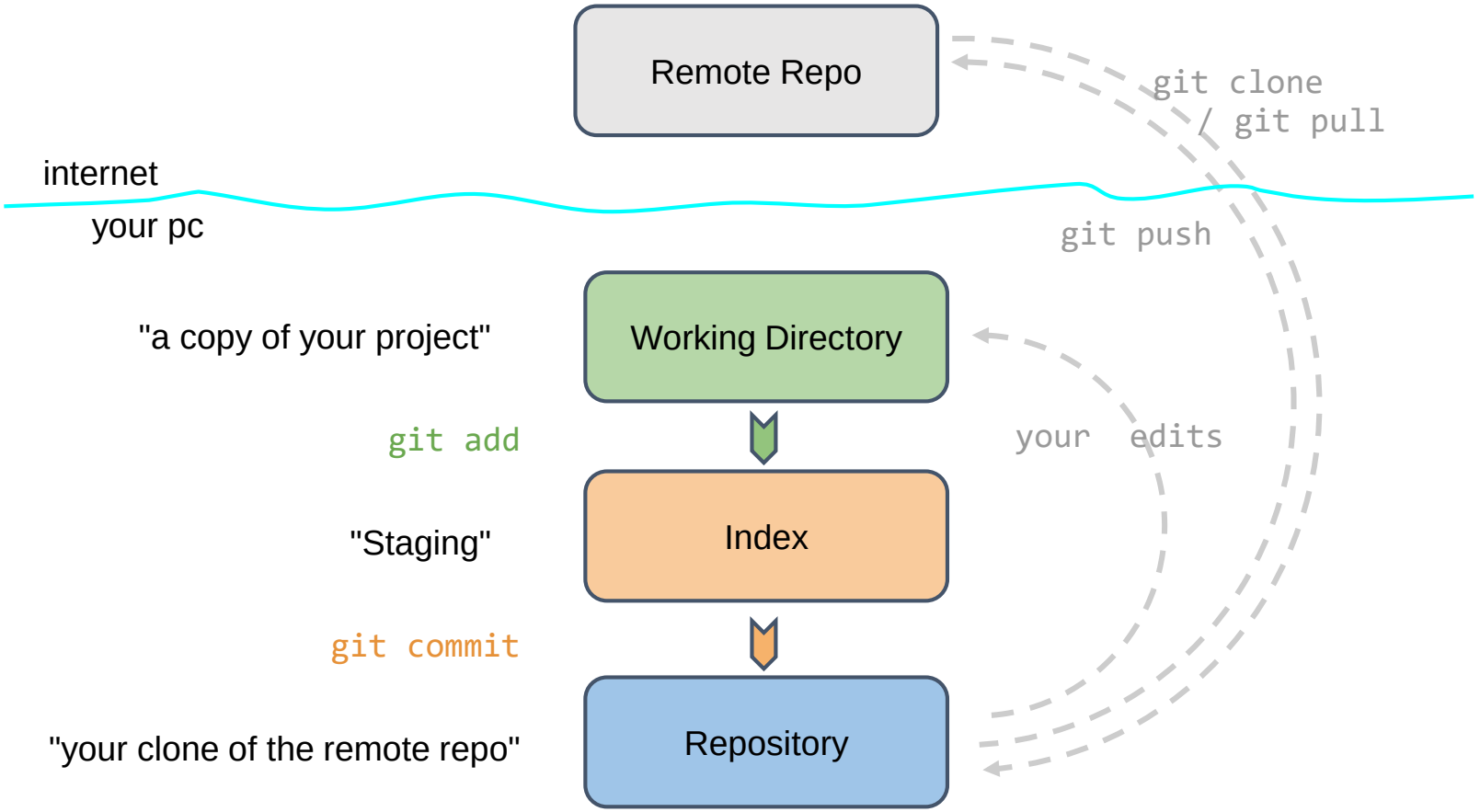
Key Concepts: How do you make a commit anyway?



Key Concepts: How do you make a commit anyway?



Understanding Git



First Steps: Check installation

- Check if Git is installed:
  - $ git --version
  - git version 1.7.10.2 (Apple Git-33)
- If not, brew install git or download the installer
- Set your Git username and email to your global config:
  - $ git config --global user.name “Anthony J Souza"
  - $ git config --global user.email “ajsouza88@gmail.com"
- How to enter your username and password only once:
  - https://help.github.com/articles/generating-ssh-keys

First Steps: Obtaining a repository

- Creating a repository (if one does not exist remotely):
  - $ git init
  - @see http://git-scm.com/docs/git-init
- Or, cloning a remote repository:
  - $ git clone https://github.com/asouza88/csc648home.git localdir
  - $ git clone https://github.com/asouza88/csc648home.git
    - localdir is optional. The repo name is used in this case.
    - @see http://git-scm.com/docs/git-clone
  - $ cd csc648home
  - $ ls –la
- Inspect the repo configuration:
  - $ vim .git/config

First Steps: Inspecting your repo

- Change Directory into the repository & issue a git status
  - $ cd path/to/repo
  - $ git status
    - @see http://git-scm.com/docs/git-status
  - "Displays paths that have differences between the index file and the current HEAD commit, paths that have differences between the working tree and the index file, and paths in the working tree that are not tracked by Git (and are not ignored by gitignore(5))"
    - @see http://git-scm.com/docs/gitignore
- In other words, shows what you added, modified or deleted since your last commit.

First Steps: Inspecting your repo

- Change Directory into the repository & issue a git log
  - $ cd path/to/repo
  - $ git log
  - $ git log --pretty=oneline
  - $ git log -1
    - @see http://git-scm.com/docs/git-log
- "Shows the commit logs."
- In other words, why you committed something
  - Hopefully you wrote descriptive commit messages in the past!

First Steps: Making Edits & Committing

- You can use whatever program you normally use to edit files.
- Make a new file - newfile.txt
- Edit the file as desired
- Save it in your working directory
- $ git status
  - will show the file as "Changes not staged for commit"
- $ git add newfile.txt
- $ git status
  - will show newfile.txt as "Changes to be committed"
  - git commit -m "Added newfile.txt as an example"
    - [Master 1e77a20] Added newfile.txt as an example
- 1 file changed, 0 insertions(+), 0 deletions(-)

First Steps: Adding (staging) changes

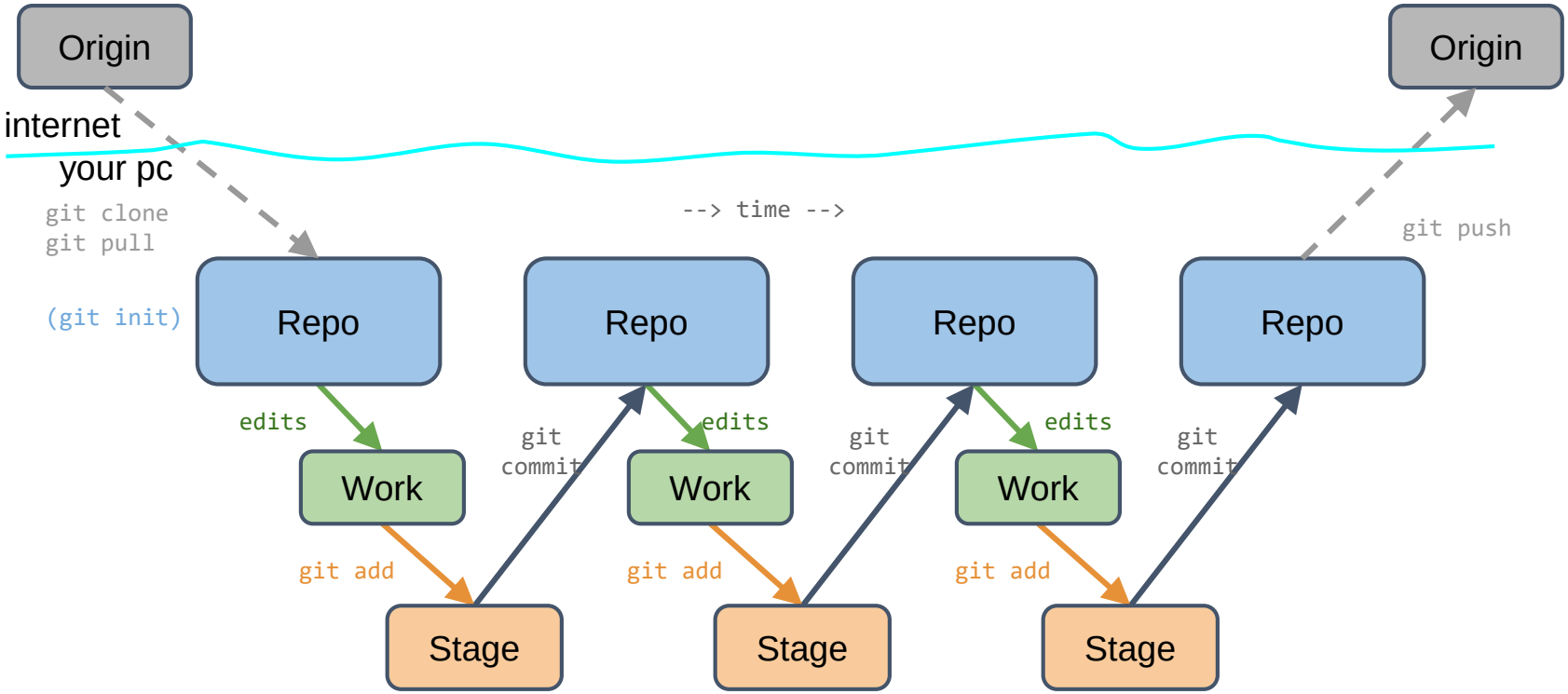
- Change Directory into the repository & issue a git status, then add new/modified files
  - $ cd path/to/repo
  - $ git add (file)
  - $ git add –A
    - @see http://git-scm.com/docs/git-add
  - "This command updates the index using the current content found in the working tree, to prepare the content staged for the next commit."
  - In other words, stage your added, modified or deleted files for committing.

First Steps: Committing (staged) changes

- Change Directory into the repository & issue a git status, then add new/modified files, then commit them to your local repository
  - $ cd path/to/repo
  - $ git add (file)
  - $ git commit -m "What did you change? Log messages matter."
    - @see http://git-scm.com/docs/git-commit
  - "Stores the current contents of the index in a new commit along with a log message from the user describing the changes."
  - In other words, store the added, modified or deleted files you staged in your repository.

Git workflow visualization

Standard Workflow



First Steps: Pushing

- At this point, none of those changes you've made have left your computer. (You still have local "infinite undo".) Let's push those changes to the remote repo.
  - git push <repository> <refspec>
  - git push origin master
  - git push origin master:master
- You now have an annotated, offsite backup of the work you performed!
  - REMEMBER:
    - If you never commit, you have no history.
    - If you never push, you have no offsite backup.
  - As a rule of thumb, commit every hour, push every day.
    - There is no reason not to commit after every "logical stopping-point" and push when a good section of work is complete, unit tests pass, etc.
  - After you commit, your colleagues can pull changes down from the origin repository to their local working repository.

First Steps: Pushing your changes offsite

- (local -> remote)
- Change Directory into the repository & issue a git push
  - $ cd path/to/repo
  - $ git push origin master
    - @see http://git-scm.com/docs/git-push
  - "Updates remote refs using local refs, while sending objects necessary to complete the given refs."
  - In other words, send your local changes back to the remote repo you cloned from.

First Steps: Pulling someone else's changes

- (remote -> local)
  - Change Directory into the repository & issue a git pull
    - $ cd path/to/repo
    - $ git pull origin master
      - @see http://git-scm.com/docs/git-pull
    - "Incorporates changes from a remote repository into the current branch. In its default mode, git pull is shorthand for git fetch followed by git merge FETCH_HEAD."
    - In other words, retrieve changes made to the remote repo and merge them into your local repo, updating your working copy to match.

Next Steps:

- Working smart(er)
- Branches

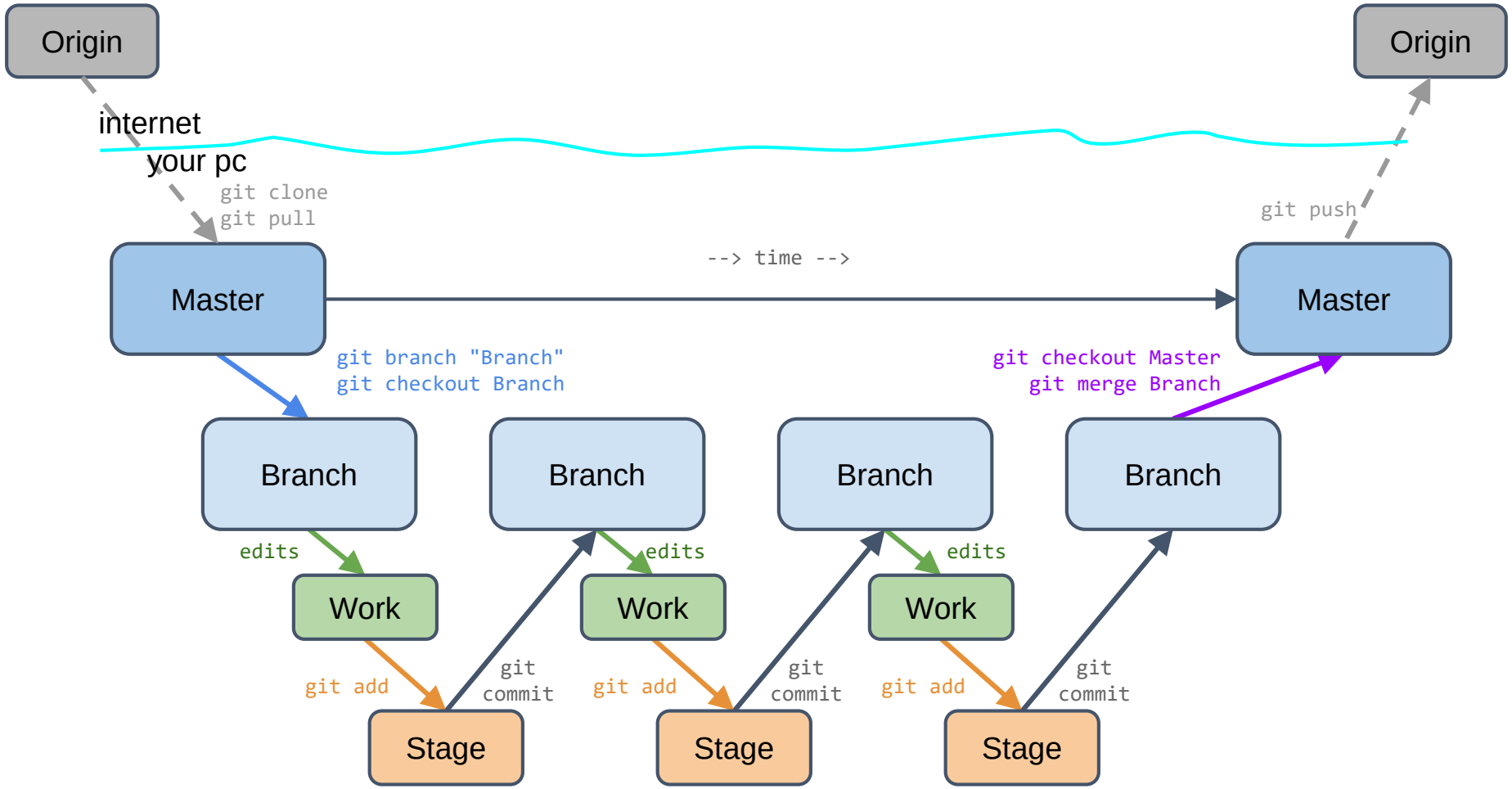
Next Steps: Working smart(er)

- Branches in Git can be created very quickly (because they happen locally). Creating a branch, and doing work in a branch, is smart because branches allow you to:
  - Try out an idea - experiment
  - Isolate work units - keep unfinished work separate
  - Work on long-running topics - come back to things later
  - Share your work in progress with others without altering the code in production or in development (remote feature branches)

Next Steps: Working smart(er)

- Why do branches matter? An example to illustrate...
  - Monday: You put all the client's code into a repo, clone it locally
  - Tuesday: You create a branch to implement a new feature, making several commits
  - Wednesday: You realize that it would be better to split out the feature into a couple different include files, so you create 2 new files and delete the old one. You commit your work.
  - Thursday: The client contacts you frantically wanting a hot fix for an issue discovered on the version of the code on the live site.
  - What do you do now?
    - You deleted their version of the file on Wednesday, right?
    - No! You made a branch.
    - Their original file is in the master branch still!
    - If you hadn't branched, you would be either downloading their original file again, or reverting your changes back to the version you had on Monday.

Next Steps: Working smart(er) Branch Workflow



Next Steps: Working smart(er)

- Change Directory into the repository, create a branch and check it out
  - $ cd path/to/repo
  - $ git branch featurename
  - $ git checkout featurename
    - @see http://git-scm.com/docs/git-branch
  - "If --list is given, or if there are no non-option arguments, existing branches are listed; the current branch will be highlighted with an asterisk. Option -r causes the remote-tracking branches to be listed, and option -a shows both local and remote branches. "
  - In other words, list or create local or remote branches depending on arguments provided.

Next Steps: Working smart(er)

- Merge changes made in a feature branch into another branch, typically development, to then be tested, and ultimately merged to the master branch and tagged.
  - $ cd path/to/repo
  - $ git branch
  - $ git checkout master
  - $ git merge featurename
    - @see http://git-scm.com/docs/git-merge
  - "Incorporates changes from the named commits (since the time their histories diverged from the current branch) into the current branch. This command is used by git pull to incorporate changes from another repository and can be used by hand to merge changes from one branch into another."
  - In other words, apply changes from another branch to the current one.

Next Steps: Working smart(er)

- Branch Best Practices
  - "master" is generally accepted to mean the main branch, that tracks what is deployed on the client's site. This is the default; when creating a new repository it will have one branch named master.
  - "develop" or "development" is typically the name of a single branch that has newer code than what is in master. Un-named bug fixes and improvements.
  - "featureXYZ" or "issue123" - Branches named for specific features or specific issue tickets in a 3rd party bug tracker.
  - You can have branches in your local repository that are not tracked, meaning they do not exist in the remote repository. It's up to you.
  - Even if you never make a feature branch, it's recommended to have at least one "development" branch separate from "master"

Final Steps: Deploying Code

- When we're satisfied with our work, everything is tested and it's ready to provide to the client, what should we do? Tag it!
  - Git has the ability to tag specific points in history as being important
  - Use this functionality to mark release points (v1.0, and so on).
  - A Tag should be used anytime you provide code to the client.
  - It should be annotated both in Git and in code comments somewhere, what you changed and why, who asked for the change, their email address and anything else that might be important.
  - Remember that the next person who works on the tag code may not be you! Pay it forward!
  - The client should be provided with access to download the Tagged version of the code
  - Github makes this a zip/tgz automatically

Final Steps: Deploying Code

- Change Directory into the repository, list the current tags, check the branch you're on, and tag the current branch/revision
  - $ cd path/to/repo
  - $ git tag –l
  - $ git branch
  - $ git tag -a v1.1 -m "Tagging version 1.1 with featurename"
    - @see http://git-scm.com/docs/git-tag
    - @see http://git-scm.com/book/en/Git-Basics-Tagging
  - "Add a tag reference in refs/tags/, unless -d/-l/-v is given to delete, list or verify tags. Unless -f is given, the named tag must not yet exist. If one of -a, -s, or -u <key-id> is passed, the command creates a tag object, and requires a tag message."
  - In other words, create a named/numbered reference to a specific revision of the code, or list prior tags created

Final Steps: Deploying Code

- A tag is a special type of commit... as such, just like any other commit, the tag only exists locally. The final step in tagging is to push the tag to the remote repository, just like we do with regular commits.
  - $ cd path/to/repo
  - $ git push origin v1.1
  - $ git push origin --tags
    - @see http://git-scm.com/book/en/Git-Basics-Tagging
  - "By default, the git push command doesn’t transfer tags to remote servers. You will have to explicitly push tags to a shared server after you have created them."
  - In other words, don't forget to push after tagging, otherwise the tag only exists on your local machine.

Most used commnads for Git

Download/Create a local repo

- $ git clone
- $ git init

Checking what's changed

- $ git status
- $ git log

Storing edits

- $ git add
- $ git commit

Updating your local repo

- $ git pull (git fetch)
- $ git branch
- $ git merge

Storing changes offsite/offbox

- $ git commit
- $ git push

Storing changes offsite/offbox

- $ git tag
- $ git push