



DevOps for Defense

May 2018

DoD Transition

JD Black

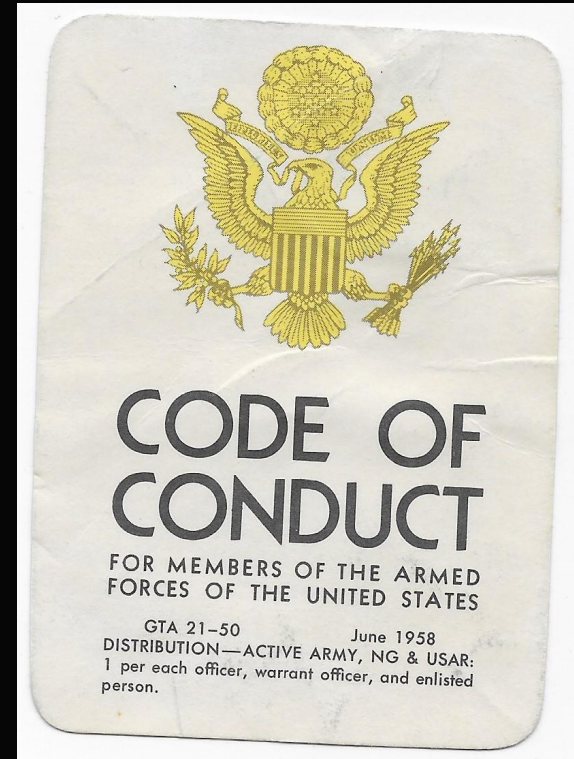
Git Overview

Ashley Hunt

<https://www.meetup.com/DevOps-for-Defense/>
<https://github.com/jondavid-black/DevOpsForDefense>
devopsfordefense@gmail.com

DevOps for Defense Meetup: Code of Conduct

- UNCLASSIFIED ONLY!!!!
- Treat each other with respect and professionalism.
- Do not talk about private, sensitive, or proprietary work.
- Do talk about your experiences, needs, desires to improve work in our domain.
- Do share your thoughts.
- Do learn from others.
- Do respect & tip your bartenders!



“Innovation cycle faster than any adversary”

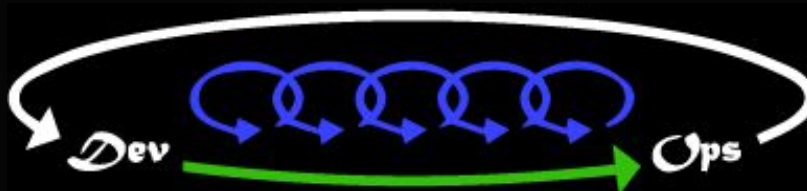
"Greater speed in translating technology into fielded capability is where we can achieve and maintain our technological edge. ... we're not out of innovators, we're not out of innovations, we're out of time. ... it is about pace. ... One of my key priorities is to enable the Department to drive the military innovation cycle faster than any adversary to sustain technological superiority."

(House Armed Services Committee (HASC) Testimony, 17 April, 2018)

<https://www.c-span.org/video/?444170-1/defense-department-officials-testify-drones-budget-request>



Dr. Michael Griffin
Undersecretary of Defense
Research & Engineering



Focus on Speed: Rapid Prototyping & Fielding

“Component Acquisition Executives (CAEs) are accountable for the management and delegation of this authority as appropriate. Organizations utilizing this authority will issue appropriate procedures for Middle Tier of Acquisition efforts to adhere to the following:

- A. **Rapid prototyping**
- B. **Rapid Fielding**



Ms. Ellen Lord
Undersecretary of Defense
Acquisition & Sustainment

Don't Sweat It - Agile is your X-Factor

“If your program contains a significant amount of software, **you automatically have an X-factor in agile development.** Agile is fundamentally different than “waterfall” development, so **traditional estimation techniques (whether cost, fielding, or risk) simply do not apply.** So don't sweat it: create an acquisition strategy based on prototyping that culminates in a separate fielding decision.”



Dr. William B. Roper
Assistant Secretary of the Air
Force

Acquisition Technology &

Software Development - Evolved

“Software development in the commercial world has undergone significant change in the last 15 years, while development of software for defense systems has continued to use techniques developed in the 1970s through the 1990s. Traditional “Waterfall” software development practices have evolved in the commercial world into an iterative process, called “Agile” or “continuous iterative development,” [and] was mostly abandoned by commercial companies years ago.”



Maintain Technological Superiority

“The DoD can leverage today’s commercial development best practices to its advantage, including on its weapons systems. Doing so will enable the DoD to move from a capabilities-based acquisition model to a threat-based acquisition model. Making this transition is necessary if the United States is to maintain its technological superiority and counter rapidly growing adversary capabilities. Our adversaries are acquiring capabilities not previously anticipated and are doing so at a pace that now challenges U.S. technological superiority.”



Defense Science Board Recommendations



1. Software Factory
 - a. aka Continuous Integration / Continuous Delivery
2. Continuous Iterative Development
 - a. aka Agile
3. Risk Reduction & Metrics for New Programs
 - a. "...all new programs, starting immediately, the following best practices should be implemented.."
4. Current and Legacy Programs in Development, Production, and Sustainment
 - a. "...immediately task the PMs with the PEOs for current programs to plan transition..."
5. Workforce
 - a. "Over the next two years, the service acquisition commands need to develop workforce competency and a deep familiarity of current software development techniques."
6. Software is Immortal – Software Sustainment
 - a. "Starting immediately, ... direct that requests for proposals (RFPs) ... should specify the basic elements of the software framework supporting the software factory..."
7. IV&V for Machine Learning

Listen to Our DoD Customers: DevOps Speed & Quality

High Performers Are More Agile

46x

more frequent
deployments

440x

faster lead times
than their peers

Source: Puppet/DORA 2017 State Of DevOps Report <https://puppet.com/resources/whitepaper/state-of-devops-report>

@Puppet

High Performers Are More Reliable

5x

lower change
failure rate

96x

faster mean time
to recover (MTTR)

Source: Puppet/DORA 2017 State Of DevOps Report <https://puppet.com/resources/whitepaper/state-of-devops-report>

@RealGenetix



Git in DevOps

by Ashley Hunt



DevOps for Defense

05/ 03 / 18

Preface

- Acknowledgements: DO4D
- This talk is based on my talk “Stop, Drop, and Stash!!!”
- This is “fun” – not particular syntax
- It’s easy to get overwhelmed (if you are new to Git)
- Git is popular, but the overall VCS concept is important here
- My examples are with the cmdline

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.

COOL. HOW DO WE USE IT?

NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.



What is Version Control? (DevOps)

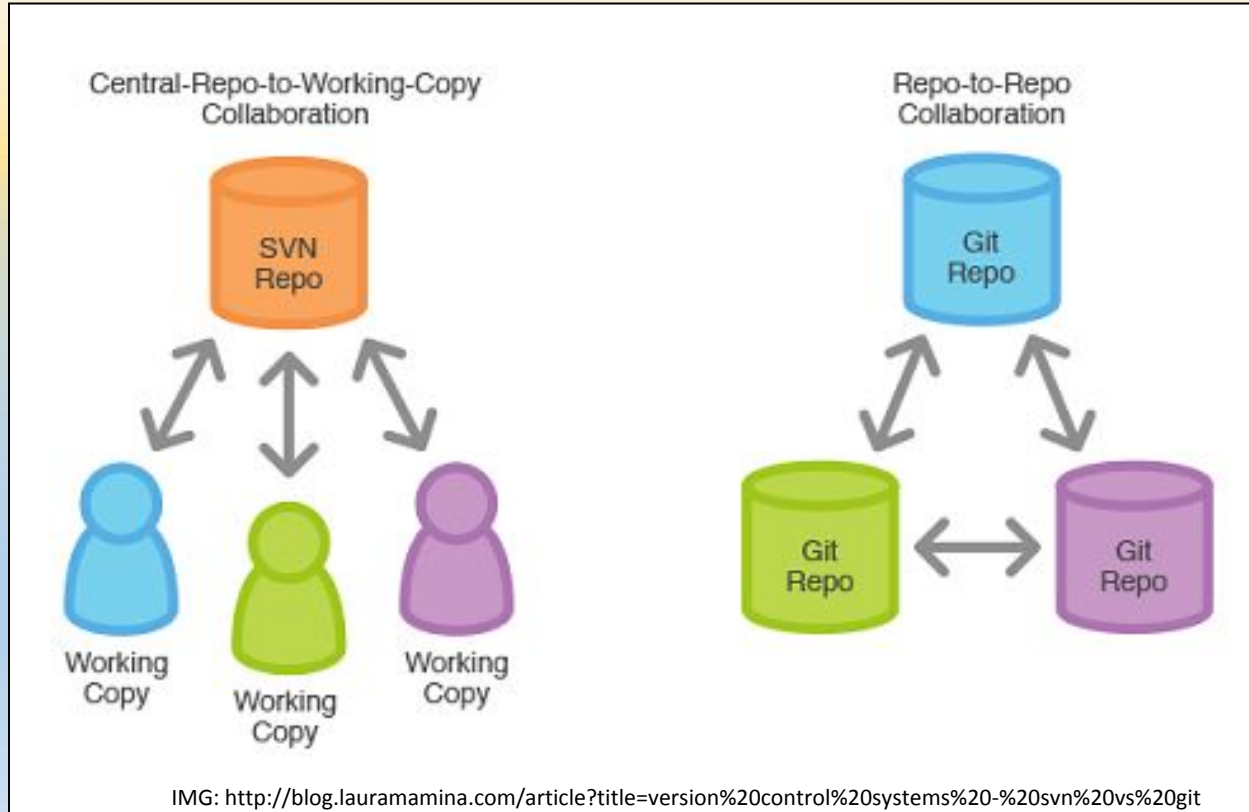
- Starting DevOps? Start with Version Control
- It's a need we have: writing code, saving text files
 - Hope that the name / date is good enough info to ID
 - Variety of types to fit every workflow / situation
 - “Everything” types vs. “VCS only” types (back-end)
 - Proprietary vs. Open Source
 - Centralized vs. Distributed



Saving your own versions can be confusing

- serverDataList.csv
- serverDataListNew.csv
- serverDataList_7-4-2014.csv
- serverDataList_Bob.csv
- serverDataList_BobFix.csv
- serverDataList_really.csv
- serverDataList_reallyFinal.csv
- serverDataList_reallyFinalNewVersion.csv

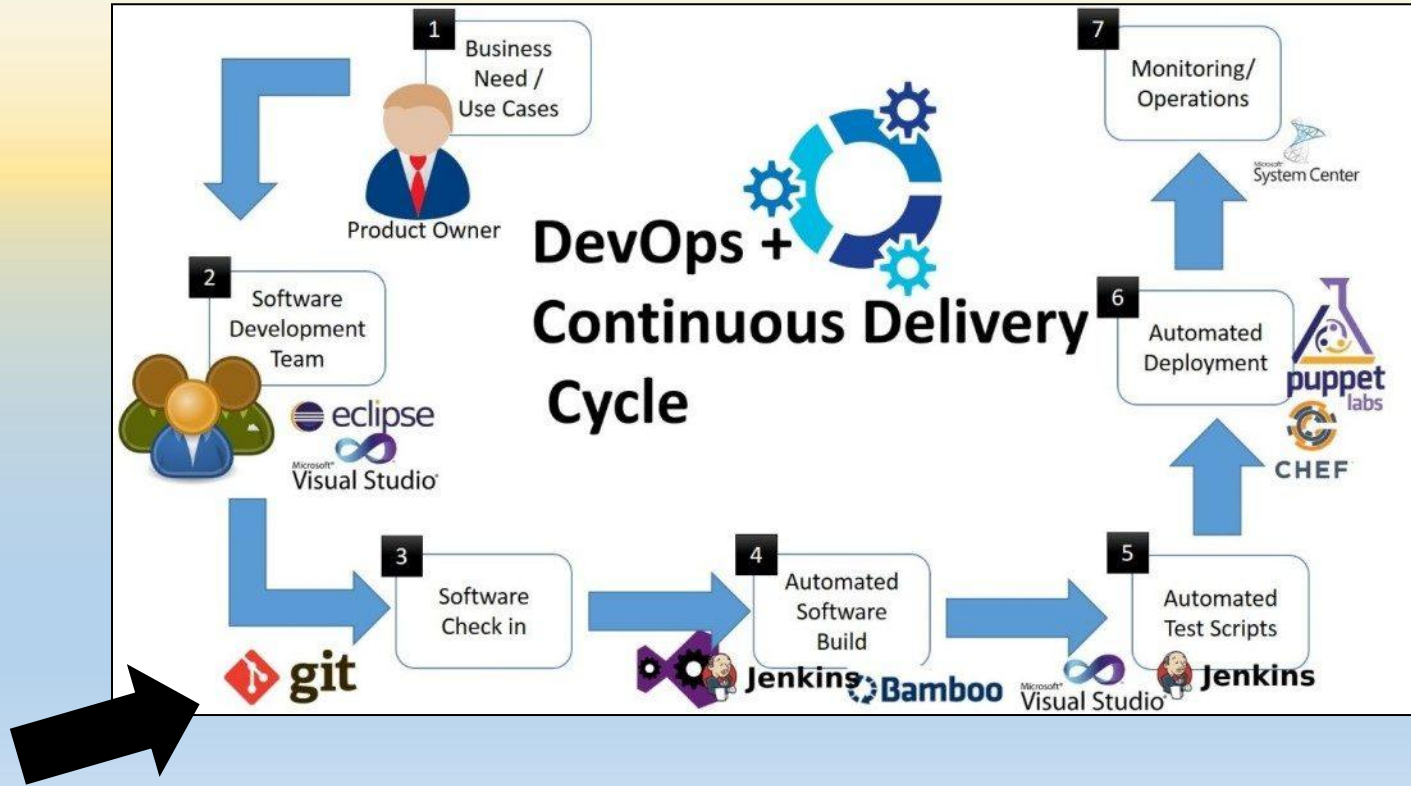
Centralized vs. Distributed



Main features of a VCS

- “Versions” of files are kept in a tracking system
- Commit access control
 - “atomic” operations, locks (concurrent access)
- Merging of changes
- Labeling a saved state (“release” or “tag”) and ability to retrieve it

Git / Version Control Role



DevOps Tools for the Job

- (Unix way) The best tools are either:
 - Simple tools: “do one job, do it well”
 - Assemblies of simple tools that work together
- VCS: The most popular dev tools are simple ones
 - Git design - the “stupid version control”
 - YES--Git, SVN, Mercurial, Perforce – VCS only
 - MAYBE NOT--Clearcase (suite), CVS, IBM Apex, Harvest, etc. – VCS + other things (env / process)
- Ultimately, the best tool is the one that you feel comfortable using.
 - SVN is **not** a bad tool! ☺

How is Git Gud?

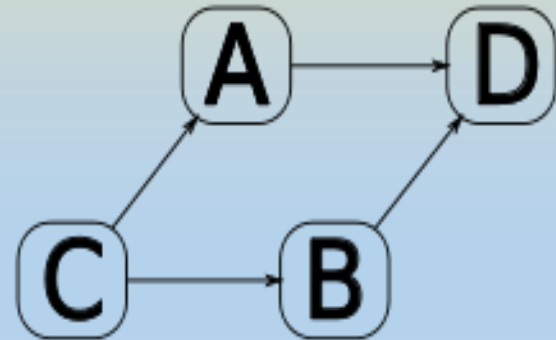
- Speed
 - Design: ~3 second patch application in mind
 - Works off of file deltas, little expense for branches & tags
 - You might not want to have LOTS of branches...but it can be done
- Distribution!
 - Everyone is a “master”
 - Potentially very backed up
 - No reliance on a central server
 - Small repo size
- It is safe. Really safe.
 - Hashes to track
 - It’s hard to “truly” erase a file
 - You have to try to lose your work*
- You can do anything* you need to do



MACGYVER
Ballpoint pen + paperclip

Some Terminology

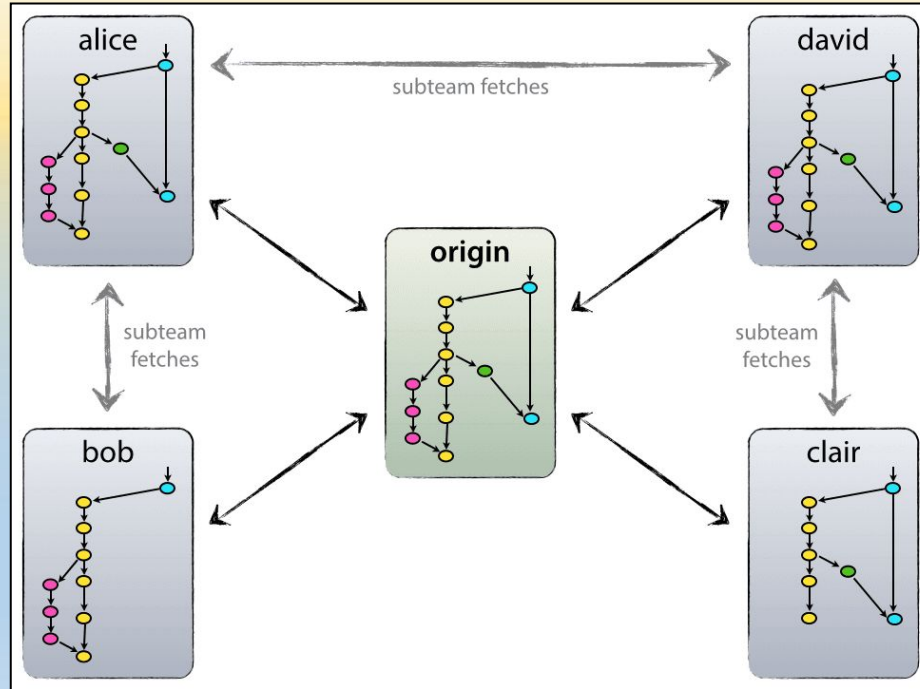
- Repository – area where code is kept
- Baseline – development or product most recent stable state of code
- Merge
 - Integration (between baselines)
 - Resolution of changes between two states
 - Several different algorithms to accomplish this
- Commit
- Tag / Label / Branch
 - “in development” work
 - “releases”



Example Workflow

- Create or join (clone) a project
- Contribute (make changes)
 - Branch, checkout, work in the branch
 - Stage changes, commit with informative messages
- Complete work & save (push / pull request)
- Transitionary phase (peer reviews, layers of testing, reverts if needed)
- Merge with master – release – tag is created (locked)

Example Workflow: The Git Interactions



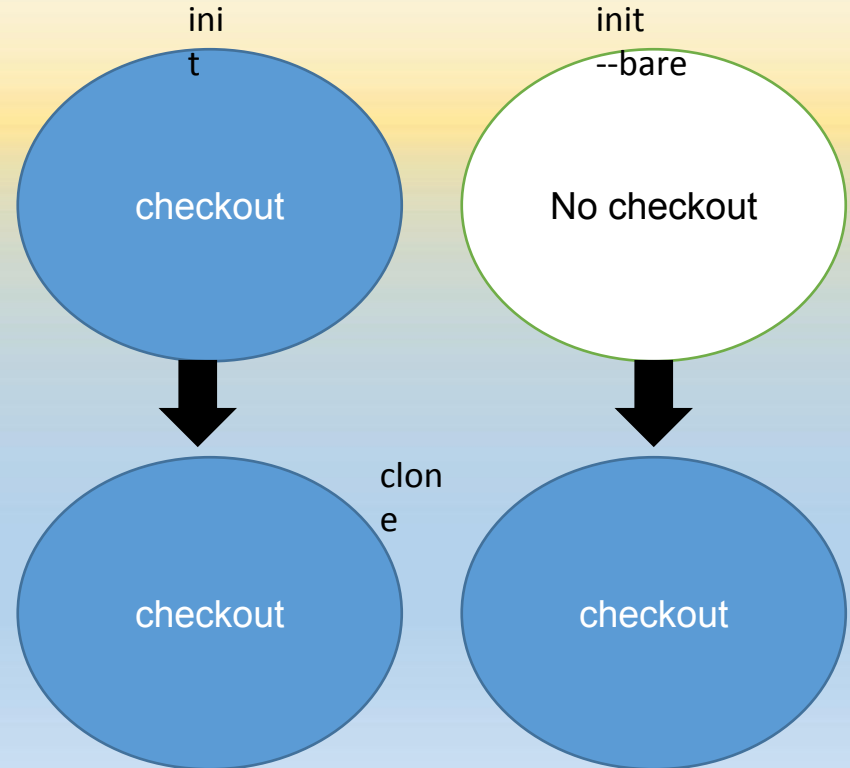
I want to make a project

(You already have Git Installed)

- Repository creation
 - Initialize a repository
 - `git init <repository name>`
 - `git init myProject`
 - Bare repositories
 - `Git init myProject.git --bare`
 - No “checkout” exists in a bare repo
 - No one can actually work here (this might be good)
 - Typically named with a `.git` on the end
- Permission control
 - Cmdline – Unix permissions / user access
 - Web-based Interfaces – in project settings or group / user settings

Make a Repository / Project

- `git init`
 - has a checkout
- `git init --bare`
 - No checkout
 - Locks others out from working there
- `git clone`
 - “checkout” cmd
 - Works with either



Clone Existing Repository / Project

- Concept of an origin (the remote, most often trunk)
 - Clone: similar to an SVN checkout; your copy
 - Track: `git remote -v` lists information about the repo
 - Syntax is “`git clone <location> <(optional) clone name>`”

Examples:

```
git clone /myPath/GitRepo/rabbit.git myRabbit
```

```
git clone https://github.com/Shopify/draggable.git
```

Add to the Repository / Project

- If bare repo, make a clone (your copy)

```
git clone /dir/myRepo.git bobsRepo
```

- Add items and stage

```
cd bobsRepo
```

```
cp file.txt .
```

```
git add file.txt    - this stages the file to be committed
```

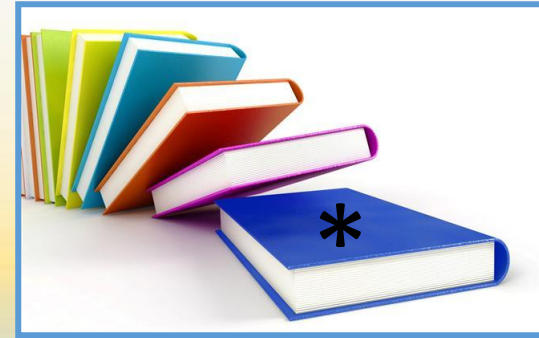
- Commit code

```
git commit -m "added the file"
```

- (if needed) push

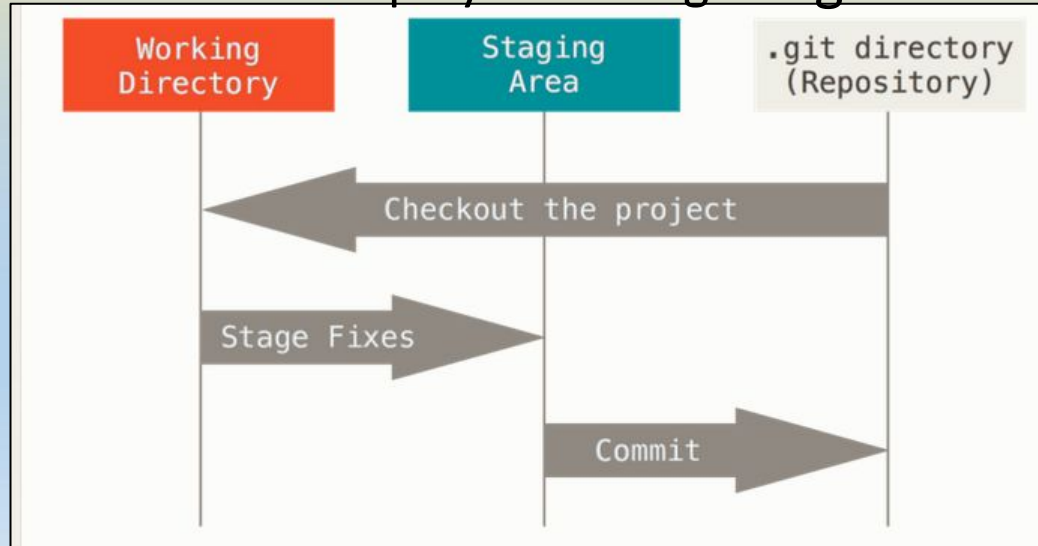
Branching

- A branch is basically your workspace, copied from another branch (usually master)
- Every repo comes with a default master
- Branching is a “safe workplace” – you aren’t changing master
 - `git branch` – list all branches, * branch is active
 - `git branch <name>` – create a branch, does NOT switch to it
 - `git branch -a` – track a remote branch
 - `git -d <name>` – delete a branch (similar with tag, must push)
 - `git checkout <branch>` – switch to a branch
- Branches are disposable and cheap (size-wise)
- Most workflows involve a branch rather than working out of “master”



A “Checkout” Does not require a “Check-in”

- Your initial copy is a clone of (or is itself) a master repository
- “checkout” switches between branches or tags (or revisions)
- It can also reset a file to wipe your changes – `git checkout file`



Retrieving Saves / Revisions

- `git pull`
- `git fetch`
- `git pull --rebase`
- `git revert <commit number>`
- `git reset --hard <file>` or `git checkout <file>`

What is the “HEAD”?

- HEAD is the most recent commit in the current branch
- HEAD can be used in commands instead of the hash of the most recent commit
- Every branch has a HEAD
- What happens if you checkout a specific revision?



Detached HEAD

- You can get here by checking out a revision
- It's called "detached" because your changes don't belong to a specific branch
- Best thing to do is create a branch for your changes
- You may see this in:
 - Use of submodules
 - Doing rebases
 - Checking out a tag



Saving Locally (Commit)

Do these things

first!

- `git diff`
 - Useful to see changes made to a specific file (GUIs have an interface with this)
 - Can be configured to your choice of tool (Diff, Meld, BeyondCompare, etc.)
- **`git status` – important command!**
 - Check the staging status of files
- stage changes
 - `git add <file or folder>` - if you've added something new or changed something
 - `git rm <file or folder>` - if you've removed something
 - Move (`mv`) and Copy (`cp`) also
- **`git commit -<option>`**
 - `git commit -m "message"`
 - `git commit -F <path to file with message>` - useful for tying in AGILE backlog story numbers

Sharing Changes: Git Push

- `git push origin master`
 - You can also set the default remote to have a different name or use a different branch
- `git merge <branch>` and push
- Rejected?
 - You might need an update! (do a pull)
 - You might have unstaged changes (do a commit or checkout or reset or a stash for these changes)
 - You might have an address / syntax issue

There are no permanent mistakes*



Tags, Sharing Changes

- The .gitignore files
 - A .gitignore file is a file that lists files that git should ignore!
 - Add & commit this
 - Git will automatically ignore executables
 - If you need to add executables, you must use a `-f` option (in general not a good idea)
- Branching vs. tagging
 - Tags are basically “locked” branches
 - They can both be deleted
- Sharing changes between branches
 - Merges between branches unless only certain changes are needed (cherry-pick)
- Sharing changes between remotes
 - Add the remote path / address and pull from it or push to it

A Useful Tool: Git Stash

- Sometimes you may want to grab updates (git pull) without running the risk of conflicts to your local changes
- Git stash is an option
 - Your local changes are saved but not committed until you apply and officially commit them
 - This is a good “quick fix” to get updates
- Syntax
 - git stash - saves off your changes
 - git stash apply - applies to branch (no commit)
 - git stash drop - deletes the stash




Quick Notes on Remotes

- It is possible to add and share changes with other repositories
 - Remember: the “master” is where the team designates it to be
`git remote add <name> <address>`
Ex. `git remote add mario https://mushroomKingdom.com`
- Once added, you can interact with the repo (based on its permissions) like you would with the master
- This permits collaboration on a large item in the workflow


What's a “Pull Request”?


- A request to push! (share publicly to the origin repo)
- This is a chance to have your changes reviewed by someone else
- If granted, a merge occurs (automatically unless there are conflicts)
 - Git prompts you with difference files to resolve the conflicts

GitHub



[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)


[+](#) 




Ashley Hunt

ashleypepita

Python is my favorite language :)

 Huntsville, Alabama

 Joined 3 hours ago

[Overview](#) [Repositories 1](#) [Stars 0](#) [Followers 0](#) [Following 0](#)

Popular repositories

[git_talk](#)

For hsv.py demo, 10/6/17


[Customize your pinned repositories](#)

5 contributions in the last year

[Contribution settings](#)

	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Mon												
Wed												
Fri												

[Learn how we count contributions.](#)

Less  More

GitLab

The screenshot shows the GitLab.org group page for @gitlab-org. The page features a header with the GitLab logo, a search bar, and navigation tabs for Group, Activity, Labels, Milestones, Issues (8,501), Merge Requests (701), Members, and Contribution Analytics. The main content area displays the group name @gitlab-org, a description "Open source software to collaborate on code", and buttons for "Leave group" and "Global". Below this, there is a section for "All Projects" and "Shared Projects" with a filter by name and a "Last updated" dropdown. A list of projects is shown, including "GitLab Development Kit", "kubernetes-gitlab-demo", "omnibus-gitlab", "GitLab Enterprise Edition", "gitlab-shell", and "gitlab-ci-multi-runner".

GitLab.org

This group Search

Group Activity Labels Milestones Issues 8,501 Merge Requests 701 Members Contribution Analytics

@gitlab-org

Open source software to collaborate on code

Leave group Global

All Projects Shared Projects

Filter by name Last updated

- GitLab Development Kit**
Get started with GitLab Rails development
- kubernetes-gitlab-demo**
Idea to Production GitLab Demo running on Kubernetes
- omnibus-gitlab**
This project creates full-stack platform-specific downloadable packages for GitLab.
- GitLab Enterprise Edition**
GitLab Enterprise Edition
- gitlab-shell**
SSH access and repository management app for GitLab
- gitlab-ci-multi-runner**
GitLab Runner

Other interfaces worth mentioning

- Atlassian's Bitbucket
 - Option to use Git or Mercurial
- A good example of an “assembly of tools” with JIRA, Confluence, Bamboo...



Some more advanced stuff

- Cherrypicking changes
 - The ability to be in one branch and choose certain changes out of another branch (or remote) and merge them in to your branch
 - Example: `git cherry-pick <commit>`
- Git squash (rewriting history)
 - “squashes” several of your commits together, so that they appear as one commit
 - This rewrites selected history
 - Why do it? Reduces the git log
 - Example using git pick:
 - `pick f7f3f6d` changed my name
 - `squash 310154e` updated README formatting
 - `squash a5f4a0d` added cat-file
 - You will be prompted to edit the new commit message upon commit

Advanced Stuff: Submodules

- What is a submodule?
 - It's basically a mini-repository, remotely tied to another repository
 - Great for separate projects that must be inter-linked
 - Example: Browser release (main repo) with an ad-blocker addon (submodule repo)
- How to set one up
 - `git submodule add <address/path> <place in the main repo>`
 - `git submodule add /mnt/mySub.git lib/mySub`
- How to clone within an existing repo
 - `git submodule --init`
 - `git submodule --update`

It's Intimidating at First: Don't Worry



Some Good Books and References

- **Git Pocket Guide** by Richard E. Silverman (O'Reilley)
- **Pro Git** by Scott Chacon and Ben Straub
 - Available online for free at <https://git-scm.com/book/en/v2>
- **Git for Teams** by Emma Jane Hogbin Westby
- <https://git-scm.com/book/en/v2>
- <https://try.github.io>
- <https://learngitbranching.js.org>
- <https://www.codecademy.com/learn/learn-git>
- <http://git.rocks>
- <https://www.codeschool.com/courses/git-real>

Group Exercise: Lean Coffee

1. Each table has a facilitator.
2. The facilitator has a short introduction to an aspect of continuous experimentation and improvement.
3. Everyone write down questions on the subject. Place them in the middle of the table.
4. The group votes on questions by placing a dot on the card. 3 votes per person.
5. Question with most dots goes first. Set a timer for 5 minutes and discuss.
6. After 5 minutes, either vote (thumbs up/down) to keep going or move on to the next question.



Agenda: Finish with a Lean Coffee

1. DevOps Transition:
Fear, Uncertainty &
Doubt
2. Lean Startup: Rapid
Prototyping
3. Configuration
Management Best
Practices & Lessons
Learned



DevOps Resources

<https://www.meetup.com/DevOps-for-Defense/>
<https://github.com/jondavid-black/DevOpsForDefense>
devopsfordefense@gmail.com

Books / Publications:

- The Phoenix Project
- The DevOps Handbook
- Continuous Delivery
- Lean Enterprise
- Lean Startup
- The State of DevOps Report
- Turn This Ship Around!

Conference Presentations (YouTube):

- DevOps Enterprise Summit (DOES)
- Velocity
- GoTo

