
GIT For Real
David Keener
June 6, 2026
Version 1.2

Agenda

- **History of GIT:** Provides the context to understand what problems GIT was designed to solve and why it works the way it does
- **GIT Cheat Sheet:** What every developer needs to know to be “fluent” in GIT usage
- **Best Practices:** How processes can be built on top of GIT to support Agile Development Practices

David Keener at the Center for Innovative Technology in Northern Virginia.

Who Am I?

David Keener

Enterprise Solutions Architect

Current Presentation

- Software Engineering Practices
- SCM
- Git
- DecSecOps

Available Topics

- AI & LLMs in Practice
- Writing
- Cloud Architecture
- Federal Cloud Migrations
- Agile in Government

25+ year architect specializing in mission-critical federal systems for DHS, CISA, and defense contractors. Layered on top of Internet startup experience with AOL, Voxant, and Grab Media. A hands-on practitioner who has led teams from trade study to production – across zero trust, cloud migration, and high-throughput cybersecurity data platforms on **Azure, AWS, and GCP**.

25+	**3**	**$60M**
Years in software architecture	Tech conferences founded	DHS program scope

Founder: RubyNation, DevIgnition & NationJS


1. History of GIT

Configuration Management Made Easy

What we care about is **Software Configuration Management (SCM)**:

The practice of handling code changes so that a system maintains its integrity over time.

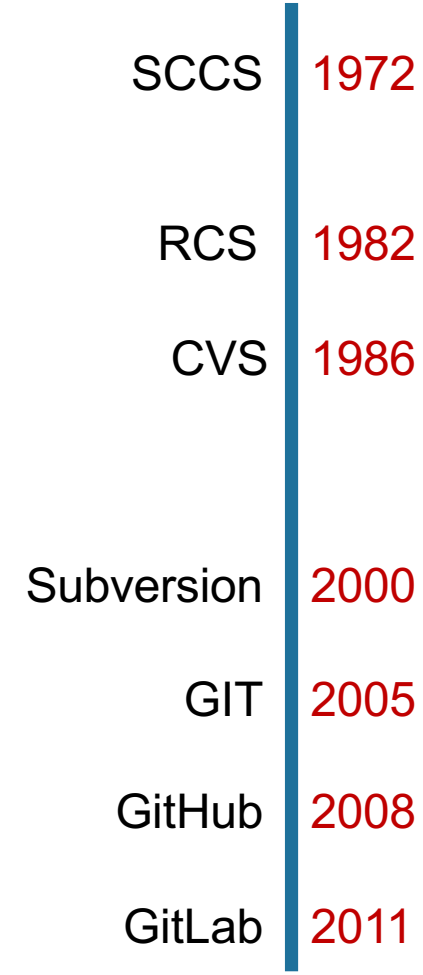
Who's Using GIT?

- Modern Software Engineering Practices require Software Configuration Management
- GIT is basically the default
- Usually in conjunction with GitHub, GitLab or a similar tool
- Usually via an IDE tool
  - Ex: Visual Studio Code
- Often integrated with a CI/CD Pipeline

A list like this was needed in 2017, when version 1.0 of this presentation was released…not so much, anymore

History of SCM Tools

- Largely driven by open source tools
- Commercial companies tended to accessorize around open source tools
- GitHub launched, making it easy and cheap to manage GIT repositories
- Competition like GitLab proved viability of online code repository model



Source Code Control System (SCCS)

- Came out in 1972
  - Not really available publically until 1977
  - First widely available SCM tool
- Had basic check-in/check-out features
- Difficult to use
  - Faded quickly when RCS came out

Revision Control System (RCS)

- Came out in 1982
- Provided simple check-in/check-out
  - Supported log messages
  - No atomic commits
- No concept of a central repository
  - Worked on single files only

Concurrent Versioning System (CVS)

- Came out in 1986
- Originally designed as a front-end for RCS
- Had a central repository
  - Usually accessible via shared storage
  - Later: accessible via Internet
- Primitive Branching/Merging
- Inflexible
  - Could not easily rename files
  - Could not easily move files/directories

Subversion

- Came out in 2000
- Basically an extension of CVS
- Featured
  - Centralized repository, web-accessible
  - Atomic commits
  - Easier branching/merging (debatable)
    - Branches are basically directories

GIT

- Came out in 2005
- Created by Linus Torvalds (of LINUX fame)
  - “git” = “unpleasant person” in British slang
- Features demanded by Linus:
  - Had to support distributed repositories
  - Super-fast performance on merges
  - When in doubt, make the opposite decision than the designers of CVS did…

What Makes GIT Different?

- Every commit has:
  - A committer
  - A timestamp
  - A SHA1 hash (unique identifier)
- Merging…
  - Branches are trees
  - Just zip the tree structures together
    - Only have to handle different commits
    - Can do diffs between commits as needed

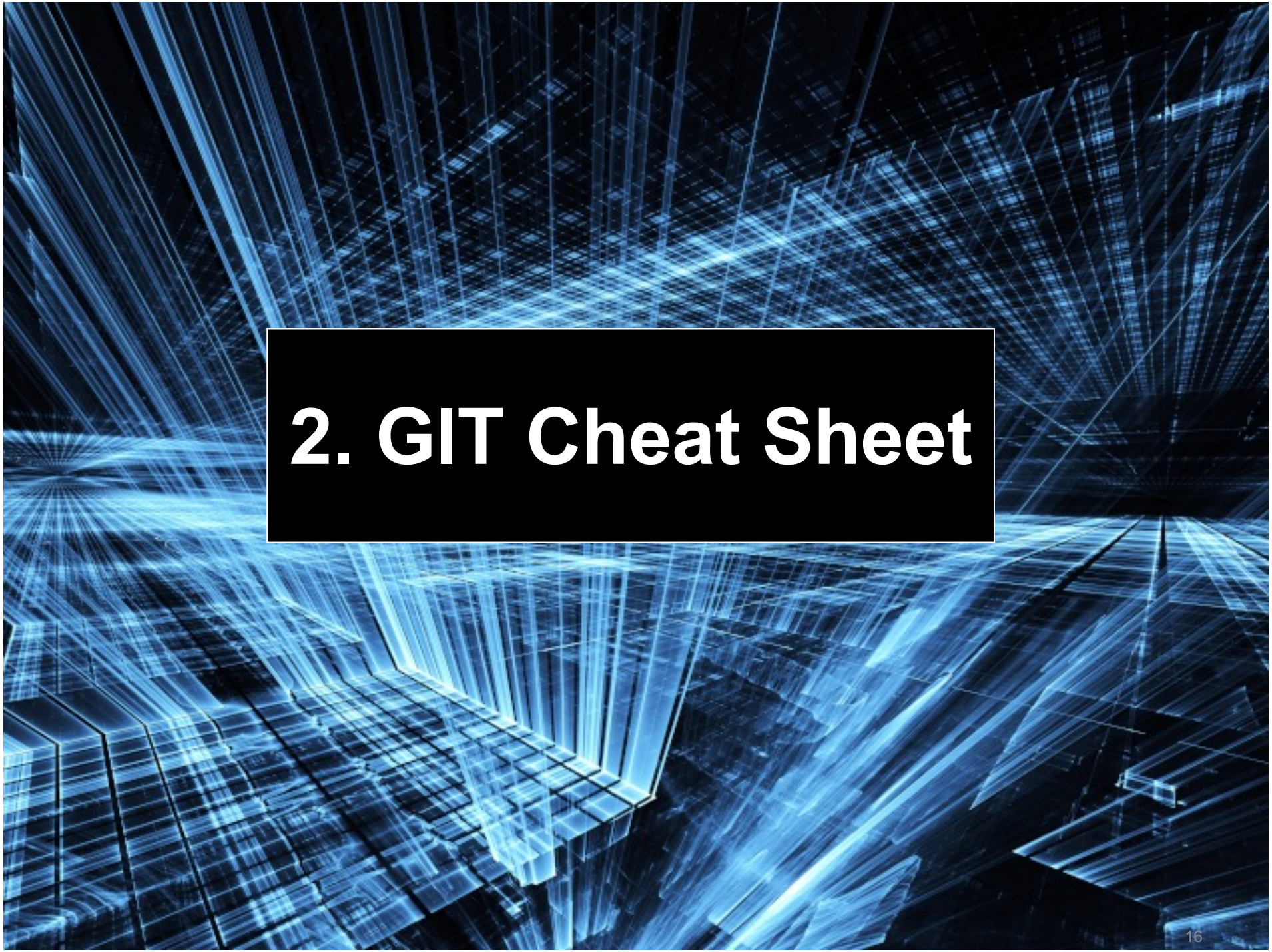
What Else Makes GIT Different?

- Every developer has a full copy of the repository
- Developers can check-in to each other's repositories
  - More useful for LINUX kernel developers…
- If the central repository crashes…
  - Nobody cares
  - Just make somebody else the new "origin"

Social Coding Platforms

GIT was given a huge lift by the rise of “Social Coding,” as exemplified by sites like GitHub, as well as its use for the LINUX kernel

- GitHub launched in 2008
- GitHub made GIT highly accessible
  - Offered free hosting for open source projects
  - Made money on commercial hosting
- By 2011, competition like GitLab emerged
  - Validated viability of “social” repository model

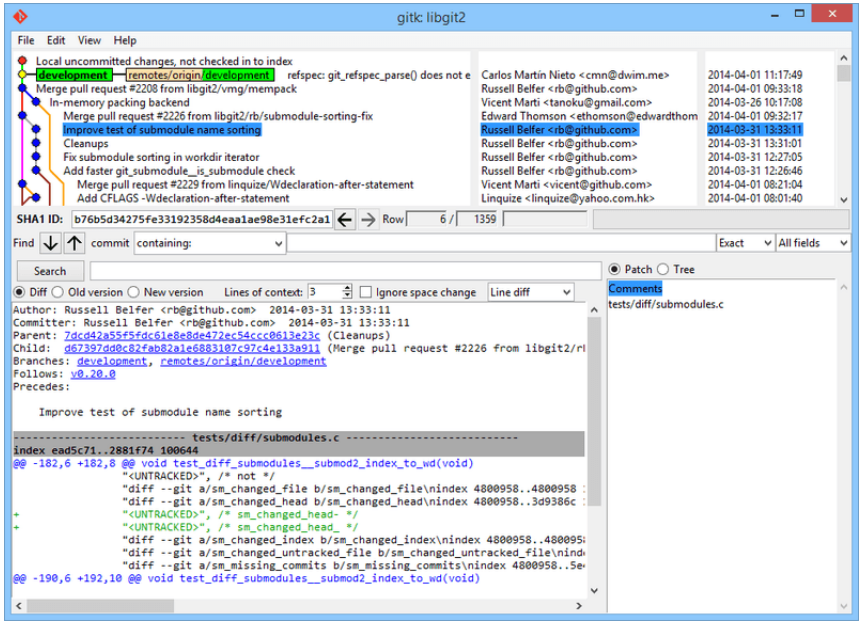

2. GIT Cheat Sheet

Git Teaching Approach

Most people will use GIT from a GUI of some sort, but we're going to show how to use it from the command line…

It's easier to learn what it's doing that way

And once you understand THAT, everything else falls into place…



Screenshot from gitk, a simple GUI tool that is bundled with the typical GIT installation

Cloning a Repository

- Via SSH

```
$ git clone git@github.com:dkeener/domain_name_validator.git
```

- Via HTTPS

```
$ git clone https://github.com/dkeener/domain_name_validator.git
```

- End result:
  - A directory called “domain_name_validator”
  - In your current directory

Er, What's a Repository?

- It's Your Codebase
  - An organized directory tree of software files

PLUS…

- The .git directory

```
dkeener@dkatlarge:~/sandbox/cyber-indicators$ ls -al .git
total 604
drwxr-xr-x  8 dkeener dkeener   4096 Mar 28 17:39 .
drwxr-xr-x 19 dkeener dkeener   4096 Mar 28 17:39 ..
drwxr-xr-x  2 dkeener dkeener   4096 Apr 23  2014 branches
-rw-r--r--  1 dkeener dkeener   1486 Mar 13 10:02 COMMIT_EDITMSG
-rw-r--r--  1 dkeener dkeener   1899 Dec 21 11:23 config
-rw-r--r--  1 dkeener dkeener     73 Apr 23  2014 description
-rw-r--r--  1 dkeener dkeener    104 Mar 28 17:39 FETCH_HEAD
-rw-r--r--  1 dkeener dkeener 328693 May 22 09:17 gitk.cache
-rw-r--r--  1 dkeener dkeener     23 Jan 23 17:15 HEAD
drwxr-xr-x  2 dkeener dkeener   4096 Apr 23  2014 hooks
-rw-r--r--  1 dkeener dkeener 168408 Mar 28 17:39 index
drwxr-xr-x  2 dkeener dkeener   4096 Dec 21 11:17 info
drwxr-xr-x  3 dkeener dkeener   4096 Dec 21 11:17 logs
drwxr-xr-x  7 dkeener dkeener   4096 Mar 13 10:02 objects
-rw-r--r--  1 dkeener dkeener     41 Mar 28 17:37 ORIG_HEAD
-rw-r--r--  1 dkeener dkeener  55313 Dec 21 11:17 packed-refs
drwxr-xr-x  5 dkeener dkeener   4096 Nov 21  2014 refs
```

Some Useful Customizations…

From within your new repository:

- Set up vi as your editor for commit messages

```
$ export EDITOR=/usr/bin/vi
$ export VISUAL=/usr/bin/vi
```

- Define Basic Repository Settings

```
$ git config --global user.name “dkeener”
$ git config --global user.email “david.keener@gd-ms.com”
```

Git Status

- Create a file called README.txt
- Check on the status of your repository

```
$ git status
# On branch master
#
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working dir)
#
# Untracked files:
#    (use "git add <file>..." to include in what will be committed)
#
#        README.txt
#
# no changes added to commit (use "git add" and/or "git commit -a")
```

Remember, “git status” is your friend

Git Diff

So, “git status” tells you which files changed, but how do you know what the changes were?

- Here’s how to find out

```
$ git diff your-file.java
```

  - Will show you what’s different

- Can also do diffs between branches

Git Add

- Stage README.txt to be committed…

```
$ git add README.txt
$ git status
# On branch master
#     (use “git push” to publish your local commits)
#
# Changes to be committed:
#    (use "git reset HEAD <file>…” to unstage)
#
#         new file:    README.txt
```

- Can stage as many files as needed

Git Commit

So, you've made changes, how do you "check in" your changes?

- Commit the file to your local repository

```
$ git commit your-file.java
```

- Can also use Unix wildcards
- Can use –a option for all uncommitted files

Git Push: Checking in Remotely

- In the last step, you checked the changes into your local repository only
- You still need to check in remotely…to "origin"
  - Defined as the repository you cloned
- Here's how to check in remotely…

```
$ git push origin master
```

- "origin" = repository, "master" = branch
- Your merge will fail if someone else has checked in more recently than you

Git Pull: Getting Remote Changes

- In the last step, your check-in FAILED because somebody else "beat you to the punch"
- You need to get the latest remote changes
  - And resolve conflicts, if any
- Get code from the remote "origin"

```
$ git pull origin master
```

- "origin" = repository, "master" = branch
- Really doing a merge
- Generally gets everything in sync…unless…

Conflict Resolution for Beginners

- "git pull" command shows files in conflict
- Can also get from "git status"
- Edit each file that is in conflict

```
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< HEAD
    Change 1
==============================
    Change 2
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> <file name>
```

- When done for each file, do "git add <file>"
- After all is complete, do "git commit"
- Now synced up, go push your changes again

Git Branch

- So, what branches do you have?

```
$ git branch
*   master
```

- An asterisk marks the branch you're on

Git Branch

- So, what branches do you have?

```
$ git branch
*   master
```

- An asterisk marks the branch you're on
- What if you have LOTS of branches?

```
$ git branch | grep '*'
```

Git Branch (2)

- Let's create a branch

```
$ git branch xyz
$ git checkout xyz
```

- Or you can do this in one step…

```
$ git checkout –b xyz
```

- Here's the cool part…
  - When you switch branches, git adjusts the directory tree you're in

Ramifications

- You have a full repository…
  - You can create any branches you need
  - Experimentation is EASY
- You can push only those changes worth pushing
  - Throw away failed experiments
- Merging takes thought, but only for conflicts
  - Git makes merging EASY
- You can even share branches

```
$ git push origin xyz
```

Git Tag

- Tag the Files in a branch (generally master)

```
$ git tag v6.5.0rc1
```

  - This tag is for version 6.5.0, release candidate 1
  - A tagging strategy is recommended

- Push the tags to the remote “origin”

```
$ git push origin master --tags
```

Git Help

- Help is always available
  - Brings up the man page for that command

```
$ git help add
```

- Get a summary of available commands

```
$ git --help
```

- Plus, git tries to provide helpful commentary in just about every context

Fluent in GIT

- This “Cheat Sheet” is basically everything you need to be fluent in git
- Yes, there’s more to know…
  - Use Google, Stack Overflow and git-scm.com as needed
- The rest is just practice…


3. Best Practices

.gitignore

- Lists patterns of files to be ignored
- Useful for ignoring local, non-source files
  - Swap files
  - IDE config files
- Rules look like:

```
/log/*.log
/tmp
*.swp
```

Master is Sacrosanct

- Master should be working code
- Developers work on branches
  - If a group is using an issue tracker, such as Jira, they may even name branches as:

    <Jira ID>_<brief name>
- Developers merge code into master only if…
  - Tests are passing
  - A Code Review has been done

Asynchronous Code Reviews

- Can be done using tools like GitHub & GitLab
- A Developer submits a “Pull Request” in GitHub, or a “Merge Request” in GitLab
  - To get Permission to merge into master
- The platform shows a diff between branch & master
  - Reviewers can vote up/down
  - Can enter comments
- Must have a policy on “passing”
  - Ex: 2 Up votes from Team developers needed to merge

Tagging Releases

- Release Builds should only be done from git
  - Based on a tag or a branch
  - Ensure nothing is accidentally included
- Tags should follow a naming convention
  - Should be well-defined, e.g. – “6.5.0rc1”
    - (Version 6.5.0, release candidate 1)
    - Can be anything reasonable that clearly identifies the version


Questions?

Revision History

- June 12, 2017 – Initial Version (1.0)
  - Presented at corporate Brown Bag session
- September 23, 2023 – Prettier & Tighter (1.1)
  - First version available on website
- June 6, 2026 – Even tighter (1.2)
  - General cleanup, new bio, improved look
  - Ready for presenting in any venue