SAGE III ISS GitHub Introduction

Git Gud

Dr. Kevin Leavor [1]
kevin.r.leavor@nasa.gov
Amy Rowell [2]
amy.rowell@nasa.gov
Dr. Charles Hill [2]
charles.a.hill@nasa.gov

[1]Science Systems and Applications, Inc.
1 Enterprise Parkway, Suite 200 Hampton, Virginia 23666

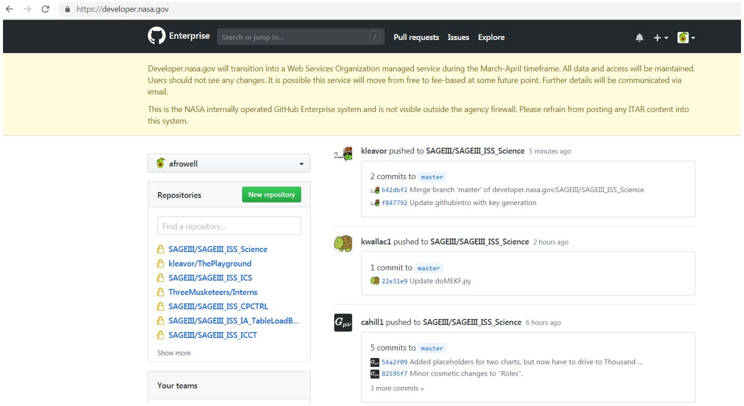
[2]NASA Langley Research Center

20 March 2019

Gitting Started

Log in to GitHub

If you have internal GitHub access: https://developer.nasa.gov



https://developer.nasa.gov

Developer.nasa.gov will transition into a Web Services Organization managed service during the March-April timeframe. All data and access will be maintained. Users should not see any changes. It is possible this service will move from free to fee-based at some future point. Further details will be communicated via email.

This is the NASA internally operated GitHub Enterprise system and is not visible outside the agency firewall. Please refrain from posting any ITAR content into this system.

afrowell

Repositories — New repository

Find a repository...

- SAGEIII/SAGEIII_ISS_Science
- kleavor/ThePlayground
- SAGEIII/SAGEIII_ISS_ICS
- ThreeMusketeers/Interns
- SAGEIII/SAGEIII_ISS_CPCTRL
- SAGEIII/SAGEIII_ISS_IA_TableLoadB...
- SAGEIII/SAGEIII_ISS_ICCT

Show more

Your teams

kleavor pushed to SAGEIII/SAGEIII_ISS_Science 5 minutes ago

2 commits to master
- 642dbf2 Merge branch 'master' of developer.nasa.gov:SAGEIII/SAGEIII_ISS_Science
- f847792 Update githubintro with key generation

kwallac1 pushed to SAGEIII/SAGEIII_ISS_Science 2 hours ago

1 commit to master
- 22e31e9 Update doMEKF.py

cahill1 pushed to SAGEIII/SAGEIII_ISS_Science 6 hours ago

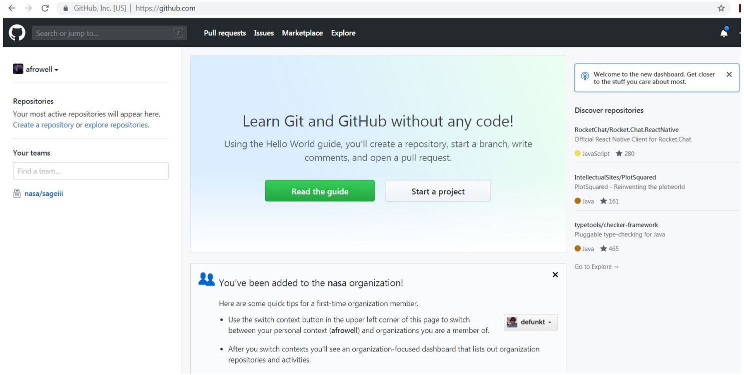
5 commits to master
- 54a2f09 Added placeholders for two charts, but now have to drive to Thousand ...
- 82595f7 Minor cosmetic changes to "Roles".

3 more commits »

Gitting Started

Log in to GitHub

If you **do not** have internal GitHub access: https://github.com


GitHub, Inc. [US] | https://github.com

Search or jump to... Pull requests Issues Marketplace Explore

afrowell

Repositories

Your most active repositories will appear here. Create a repository or explore repositories.

Your teams

Find a team...

nasa/sageiii

Learn Git and GitHub without any code!

Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.

Read the guide Start a project

You've been added to the **nasa** organization!

Here are some quick tips for a first-time organization member.

- Use the switch context button in the upper left corner of this page to switch between your personal context (**afrowell**) and organizations you are a member of. defunkt
- After you switch contexts you'll see an organization-focused dashboard that lists out organization repositories and activities.

Welcome to the new dashboard. Get closer to the stuff you care about most.

Discover repositories

RocketChat/Rocket.Chat.ReactNative
Official React Native Client for Rocket.Chat
JavaScript ★ 280

IntellectualSites/PlotSquared
PlotSquared - Reinventing the plotworld
Java ★ 161

typetools/checker-framework
Pluggable type-checking for Java
Java ★ 465

Go to Explore →

How to git a Public Key

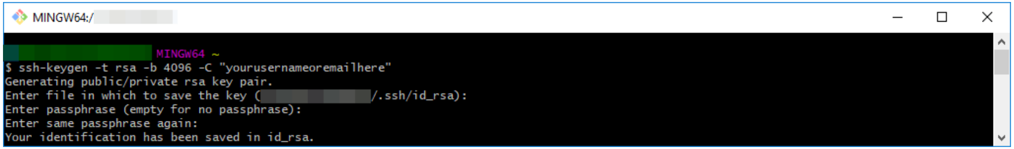
- The NASA internal GitHub requires SSH for repository access.
  - This allows for identity signing of all access.
  - SSH authentication is performed using key pairs as in RSA authentication using **public** and **private** keys.
- This section will walk through generating and registering a key pair in git-bash.
  - A brief translation section will be provided for those using PuTTY (e.g. TortoiseGit).

How to git a Public Key

Generating Your Public/Private Key Pair

Quick Reference:
https://help.github.com/en/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

- Generate SSH key
  `ssh-keygen -t rsa -b 4096 -C "YourUserNameOrEmailHere"`
  Press `<Enter>` to accept id_rsa for the filename
  Enter a passphrase if desired (you will be prompted for this when pushing)


MINGW64:/[redacted]

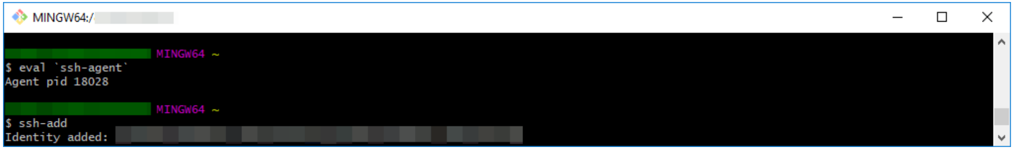
```
[redacted] MINGW64 ~
$ ssh-keygen -t rsa -b 4096 -C "yourusernameoremailhere"
Generating public/private rsa key pair.
Enter file in which to save the key ([redacted]/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in id_rsa.
```

How to git a Public Key

Adding your Key to SSH

Quick Reference: https://help.github.com/en/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

- (Optional) Add your SSH key to the ssh-agent
  If you saved a new key for some reason, we will add it now.
  `eval ‘ssh-agent‘` (This starts the ssh-agent on your machine)
  `ssh-add <path/to/your/keyfile/here>`
  (path is probably ∼/.ssh/WhateverYouNamedIt)



```
[redacted] MINGW64 ~
$ eval `ssh-agent`
Agent pid 18028

[redacted] MINGW64 ~
$ ssh-add
Identity added: [redacted]
```

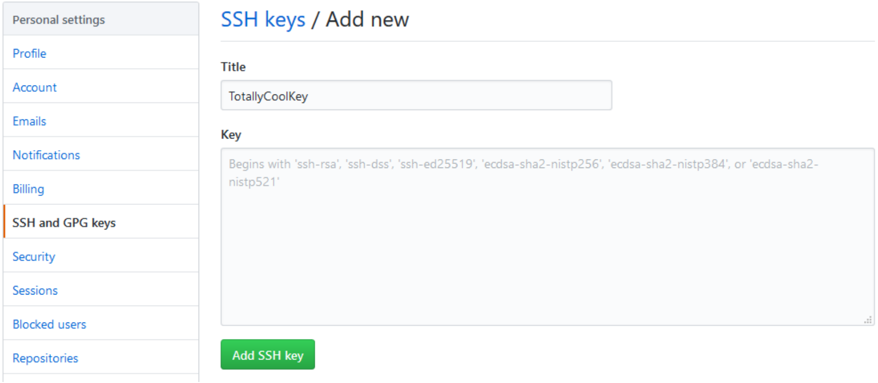
How to git a Public Key

Adding Your New SSH key to Your GitHub Account:
https://help.github.com/en/articles/adding-a-new-ssh-key-to-your-github-account

- Go to GitHub (`developer.nasa.gov` or `github.com`)
- Click your avatar in the top right and click "Settings"
- Click "SSH and GPG keys" on the left
- Click "New SSH Key"
- For "Title", give it a name you will recognize like "WorkLaptopKey"
- Copy the contents of the public section of your SSH key (e.g. id_rsa.pub) into "Key"
  - This key should include the "ssh-rsa" portion at the beginning.

How to git a Public Key

Adding Your New SSH key to Your GitHub Account:
The previous steps should have you looking at something like this
https://help.github.com/en/articles/adding-a-new-ssh-key-to-your-github-account



Personal settings
- Profile
- Account
- Emails
- Notifications
- Billing
- SSH and GPG keys
- Security
- Sessions
- Blocked users
- Repositories

SSH keys / Add new

Title: TotallyCoolKey

Key: Begins with 'ssh-rsa', 'ssh-dss', 'ssh-ed25519', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', or 'ecdsa-sha2-nistp521'

A Quick Aside for Those of You Using PuTTY

Generating a Key Using PuTTYGen:

- TortoiseGit and PuTTY can be configured to use OpenSSH keys.
- If you use the PuTTYGen format, the keys are saved into different files, such as mykey.ppk for "public-private-key."
- If you have an existing key, it can be loaded and converted to/from OpenSSH.
- To convert from OpenSSH to PuTTY, click "Conversions," "Import," then "Save public key" and "Save private key."
- To convert from PuTTY to OpenSSH, click "Load" if you are converting an existing key, "Conversions," "Export OpenSSH key."

Defining Version Control

What is **Version Control**?

- A vital component of software configuration management
- Aimed at supporting the tracking (by a number or letter code) of changes to a code base
  - or project documents, or markup ASCII files such as web sites or $\LaTeX$, or many other collections of information
    (This chart package was managed on GitHub.🤓)
- A logical way to control the reversal of changes when a developer has broken a build
- A method for enforcing annotations of changes introduced
- A powerful collaboration tool for software teams—when the process is understood and followed correctly

Like all engineering, Version Control is a **discipline** that must be learned and practiced.

Defining Version Control

What was life like **prior to** the introduction of Version Control Systems?

- Messy—life in the vacuum was oppressive and bleak.
- To introduce and roll out a change to a product, an individual developer would, for instance
    - retrieve archived version corresponding to a version in the production environment
    - proceed to make and test the change
    - assign a new version number to the release
    - add notes detailing the change and release back to the production environment
- To revert a change
    - match the release notes to specific files
    - reverse-engineer the actual changes introduced in those files
    - revert the changes manually and test
    - deploy the rectified version with release notes back to the production environment

Generations of Version Control Systems (VCS)

First generation — Local Version Control Systems (LVCS)

- One person on one file at one time.
- Checkout locks files from editing.
- Change management is accomplished by maintaining a revision history in each file.
- Examples:
  - Revision Control System (RCS)
  - Source Code Control System (SCCS)

Generations of Version Control Systems (VCS)

Second generation — Concurrent Version Control Systems (CVCS)

- Simultaneous modifications with the **merge-before-commit** philosophy.
- Conflicts are resolved when committing, but access to repository server is required before versioning.
- This generation introduced the notion of the *central repository* as a collaboration tool and Single Point Of Truth.
- Unit of change was tracked as the change on a set of files, not a single file.
- Examples:
  - CVS
  - Subversion (svn)
  - Perforce

Generations of Version Control Systems (VCS)

Third generation — Distributed Version Control Systems (DVCS)

- Simultaneous modifications with the **commit-before-merge** philosophy.
- Each developer keeps a copy of the remote repository alongside their working copy.
- Backups are maintained in a distributed manner, and users are required to merge differences and resolve conflicts before pushing changes.
- Example:
  - Git
  - Mercurial
  - BitKeeper

Version Control is Your Friend

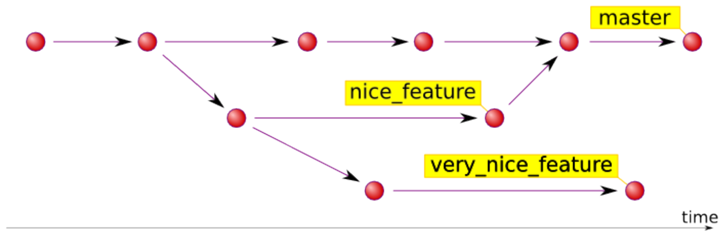
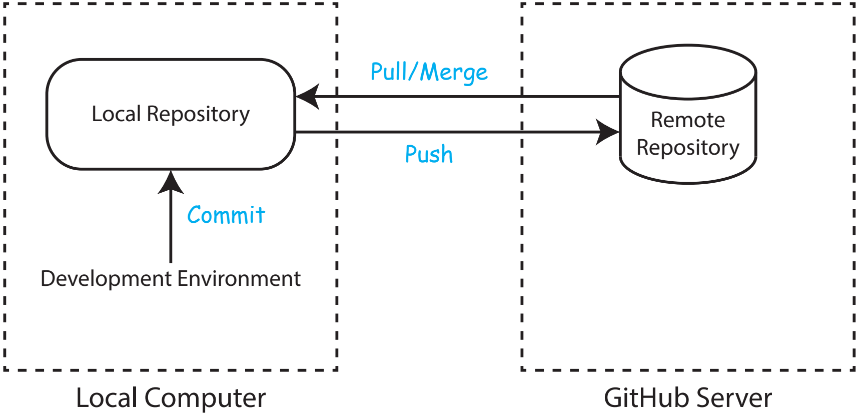


Image Taken Directly From hades.github.io

What are ...

- Repositories? Pulls?
- Commits/Pushes?
- Tags?
- Branches? (Master? Develop?)
- Conflicts?
- Merges?

GitHub Architecture



Git Commands

What is Git? — Distributed version control system

- `git init`
  Initialization of a repository — creates .git folder and contents
- `git clone /path/to/repository`
  Create a local copy of an existing repository
- `git remote add origin /path/to/repository`
  Add a new remote repository
- `git add <filename>`
  Add a file to be committed
- `git commit -m "This is my awesome commit message"`
  Commit in your local repository with a message
- `git push -u origin master`
  Updates remote repository with local repository changes
- `git pull`
  Fetch and merge changes from remote repository

Git Commands, tl;dr

Check out these cheat sheets:
https://education.github.com/git-cheat-sheet-education.pdf
https://www.git-tower.com/blog/git-cheat-sheet

And Now For The Discipline. . .

A Developer's **Checklist** for `pushing` to a Remote Repository.

1. Make Changes Locally to Code Base
2. Build Changes Error-Free in Working Directory
3. Commit Changes Locally with Detailed Commit Messages
4. `pull` from the Remote Repository
5. Merge and Reconcile Conflicts
6. Build Changes Error-Free in Working Directory
7. `push` to the Remote Repository

Easy when you get the hang of it!

Git Workflows

Workflows refer to the approach a team takes to introduce changes to a code base.

- A workflow is characterized by a distinct approach in the usage of branches (or lack thereof) to introduce changes to a repository. For example,
  - Gitflow Workflow
  - Centralized Workflow
  - Feature Branch Workflow
  - Forking Workflow

Git Workflows

Workflows refer to the approach a team takes to introduce changes to a code base.

- A workflow is characterized by a distinct approach in the usage of branches (or lack thereof) to introduce changes to a repository. For example,
  - Gitflow Workflow
    - Two branches are used: **master** and **develop**.
    - The master branch is used to track release history.
    - The develop branch is used to track the total history of feature integration.
  - Centralized Workflow
  - Feature Branch Workflow
  - Forking Workflow

Git Workflows

Workflows refer to the approach a team takes to introduce changes to a code base.

- A workflow is characterized by a distinct approach in the usage of branches (or lack thereof) to introduce changes to a repository. For example,
  - Gitflow Workflow
  - Centralized Workflow
    - » The **master** branch is the default development branch too.
    - » All changes are committed to the master branch.
    - » This is a suitable workflow for small teams or those transitioning from SVN (trunk = master).
  - Feature Branch Workflow
  - Forking Workflow

Git Workflows

Workflows refer to the approach a team takes to introduce changes to a code base.

- A workflow is characterized by a distinct approach in the usage of branches (or lack thereof) to introduce changes to a repository. For example,
  - Gitflow Workflow
  - Centralized Workflow
  - Feature Branch Workflow
    - » Feature development is carried out in a dedicated branch.
    - » That branch is then merged to the **master** once the intended changes are project-approved.
  - Forking Workflow

Git Workflows

Workflows refer to the approach a team takes to introduce changes to a code base.

- A workflow is characterized by a distinct approach in the usage of branches (or lack thereof) to introduce changes to a repository. For example,
  - Gitflow Workflow
  - Centralized Workflow
  - Feature Branch Workflow
  - Forking Workflow
    - The developer seeking to make a change creates a copy of the desired repository in their individual GitHub account.
    - The changes are made in the copy of the source repository.
    - It is then merged to the source repository through a "pull request".

Git Workflows

Workflows refer to the approach a team takes to introduce changes to a code base.

- A workflow is characterized by a distinct approach in the usage of branches (or lack thereof) to introduce changes to a repository. For example,
  - Gitflow Workflow
  - Centralized Workflow
  - Feature Branch Workflow
  - Forking Workflow

tl;dr Teams must make process decisions before development starts!

GitHub

What is GitHub? — Git repository hosting service

Why use GitHub? What are some of the features, tools, and advantages?

- Collaborative software development
- Issue tracking
- Web-based GUI
- Project management

Limitations

- No files >150MB
- Will receive a warning for any files >50MB

GUIs

- TortoiseGit (https://tortoisegit.org/)
    - Pushing of large amounts of data is known to be slow.
- Git Extensions (https://sourceforge.net/projects/gitextensions/)
- GitHub Desktop (https://desktop.github.com/)
    - Currently does not play well with ssh.
- GitKraken
    - Requires login paired with GitKraken account; Paid to interface with GitHub Enterprise.
- Fork
    - Visually-oriented GUI with significant-praise.

Profiles and Personalization

Profile

- Add your email address
- Add and name your public key

Personalization

- To start with, you will be issued an "Identicon"
- These are "simple 5×5 "pixel" sprites that are generated using a hash of the user's ID. The algorithm walks through the hash and turns pixels on or off depending on even or odd values. These generated patterns, combined with hash-determined color values, ensures a huge number of unique Identicons."
- You can easily select your own avatar and upload a picture of your choosing.

For this training we will have a competition for best avatar!

Organizations and Teams

Organizations — A group of users that share ownership of projects, repositories

- The SAGE III organization has been created for our use. As people gain access to GitHub Kevin, Charles, and Amy have the ability to add people to the organization.
- When you create a new organization you become the owner (essentially admin) by default but you can invite people to be co-owners.
- Repositories that are created under an organization can be utilized by all members of the organization.

Teams — A grouping of users within an organization

- Can be used to define the level of access to each repository. (admin, read, write)
- Can be used to mention a group of people in a comment, issue, or pull request. Similar to our Ops, Science, SCF, and GS email addresses

Roles

Roles — Define the level of access to your organization, its settings, and data

Within an organization

- Repository permissions include:
  - Admin — able to clone, pull, push, and add new collaborators to all repositories
  - Write — able to clone, pull, and push repositories
  - Read — able to clone and pull repositories
  - None — able to clone and pull public repositories
- Roles include:
  - Owner — Full administrative access to the entire organization
  - Member — Can see every member and non-secret team in the organization, and can create new repositories

Within a team

- Maintainer — Can add and remove team members
- Member — Member of the team

Special Files

GitHub recognizes a number of special files with specific roles.

- README[.md;.rst;etc]
  - A summary that will be interpreted and displayed on the repository main page.
- LICENSE
  - A file describing the legal responsibilities of the contents of the repository.
  - GitHub recognizes and summarizes certain formats (Apache, GNU, etc.).
- CONTRIBUTING
  - Guidelines for contributing. Will be linked by GitHub during ticketing and pull requests.

Special Files

- CODE_OF_CONDUCT
  - A description of how users are expected to handle themselves.
- CODEOWNERS
  - A list of who owns what in the repository. Accepts wildcards (*).
  - Higher levels take priority in lower levels until contradicted.
- .gitignore
  - A list of files (can include wildcards) git should ignore for versioning.
  - Useful for final large files produced by runs (e.g. pdfs).
- Empty folders
  - git will ignore empty folders by default.
  - To maintain directory structure, add a blank ".gitignore" file. An empty ".gitkeep" is also an accepted unoffical standard.

An Overview of VCS Philosophy

What this section is

- A ***BRIEF*** overview of versioning philosophy.
- A means to get on the same page with terminology.
- An introduction to systems, best practices, and industry standards.
- A look at what we do right as an organization and places we can improve.

And what it isn't

- An in depth user guide on a system of choice.
- An answer to all of your problems.
    - In fact, expect more questions before we get to the end.

Software Management

Readings from the Book of Joel, Chapter 1, Verse 1

On Software Developers (paraphrasing)

Good software *developers* do not necessarily make for good software *managers*.

[1] **spolsky:2004**.

Software Management

Readings from the Book of Joel, Chapter 1, Verse 1

On Software Developers (paraphrasing)

Good software *developers* do not necessarily make for good software *managers*.

Software Management is about *process*, not code slinging.

[1] **spolsky:2004**.

Software Management

Readings from the Book of Joel, Chapter 1, Verse 1

On Software Developers (paraphrasing)

Good software *developers* do not necessarily make for good software *managers*.

Software Management is about *process*, not code slinging.

- "Managing software projects requires a completely different set of skills and techniques than writing code does; they are two radically different and unrelated fields."
- "Writing code is as different from management as brain surgery is from pretzel-baking."[1]

[1] **spolsky:2004**.

Software Management

The Joel Test

(for assessing a software team)

1. Do you use source control?
2. Can you make a build in one step?
3. Do you make daily builds?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
6. Do you have an up-to-date schedule?
7. Do you have a spec?
8. Do programmers have quiet working conditions?
9. Do you use the best tools money can buy?
10. Do you have testers?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

Philosophy — Specs and Schedule

What should the code do, and when should it be completed?

- Feature Tracking and Bug Tracking are indistinguishable.
  - "Missing features" are tantamount to bugs.
  - Features have request dates, descriptions, and should be tracked to completion.
- Every build should have a set completion date.
  - SAGE as a mission worked by following schedule, maintaining critical path, and defining milestones and their completion date.
  - Software should fundamentally be no different.
- If an undefined feature isn't necessary for a critical feature or bugfix, it's tabled unless
  1. Sufficient schedule slack or personnel are identified.
  2. Another feature is slipped to the next build.
- There is no code TARDIS (though all code is definitely bigger on the inside...)

Philosophy — Bug Database

What is broken, How is it broken, Who is fixing it, and When is it getting fixed?

- Feature Tracking and Bug Tracking are indistinguishable!
- Track ***everything***.
  - "Never" is an acceptable answer to when. (Soon^{TM} is also acceptable in some circles...)
  - You are tracking decisions as much as deficiencies.
- Link to your Version Control System (VCS).
  - TODO is common and supported in most Interactive Development Environments (IDEs).
  - Ticket number in VCS commits!
- Increases collaboration and usability.
  - Meeting suggestions can be tracked.
  - All parties can view progress.
  - Becomes an item to quickly cover closure, progress, and new items.
  - Ticket discussions and decisions can be linked into official documentation.

Philosophy — Building Daily, Nightly, and Simply

The scheduled build is a tool to manage code quality.

- Ensures maintainability.
- If the code doesn't compile from scratch it's out of spec.
- Can be handled by `cron` in a clean folder. (or Windows$^{\text{task}^{\text{scheduler...}}}$)
  - Link to VCS, Rollback on errors
  - Assign blame (through VCS) for all breaks.
- Maintain a test directory covering known edge cases.

The one-line build

- Compilation should be a script which is versioned.
- Enables automated scheduled builds.
- If it's scripted, it's been considered.
- Hallway Usability! (Can Jamie build your trending code?)

Philosophy — Source Control

A VCS — Like GitHub — Enables

- Collaboration. VCSs are ultimately collaboration tools first and foremost.
- Freedom to make updates. Nothing is permanent, and history is forever.
- Distributed backups where nothing can ever be lost forever.
- Tracking to completion with comments detailing who, what, why, when, and how.

Commit Early. Commit <u>Often</u>. Leave detailed commit messages.
Upcoming versions should be in branches with maintenance in `master`.
Every version producing a product release should be tagged.

VCS Tools

Many of these tools can integrate with one another

- For instance, Trac supports both svn and git integration natively with minimal configuration.

The tools themselves do not matter!

- What matters is the group decides on the tools that meet their needs.
- Also, that the group actually uses the tools.
- **How a decision is made can be more important than the decision being made.**

Treat the repository as the Single Point of Truth. If it's not in the repository, it's not in the build. If it is, it can be cross-referenced, regex'd, parsed, and used. **This includes the commit logs!**

- Release notes and changelogs can be generated straight from commits!

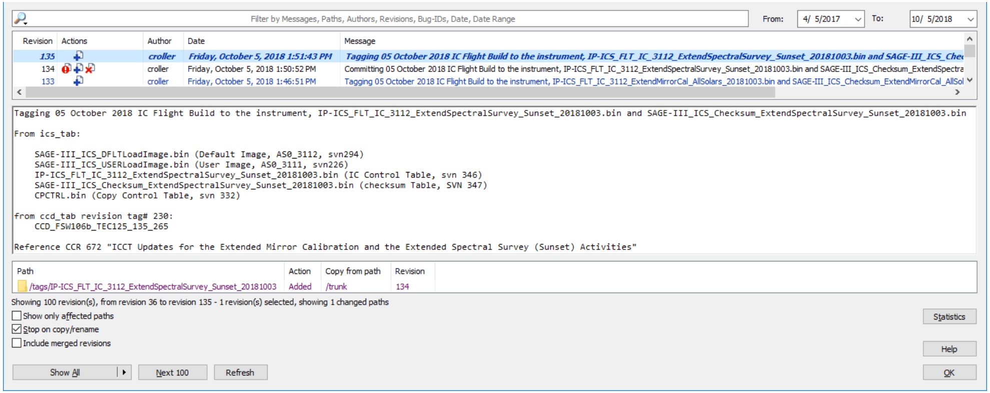
Examples — Things the Mission Does Right

The IC Flight Builds Repository

This repository maintains a record of the complete configuration of the Instrument Controller software build loaded to the in-flight instrument.

- Designed by Kevin and Sudha after proposal to mission management.
- Solved a problem knowing which versions of flight software components maintained separately were loaded at a given time.
- Checksums/CRCs cannot be generated without the full builds and push back on operational mnemonics.
- Commit logs were designed at repository creation to be uniform, meet QA needs, and link back to component repositories.
- Creates a single place where operators can get full builds with associated checksums, reducing potential opportunities for human error.

Examples — Things the Mission Does Right



Filter by Messages, Paths, Authors, Revisions, Bug-IDs, Date, Date Range   From: 4/ 5/2017   To: 10/ 5/2018

Revision	Actions	Author	Date	Message
135		croller	Friday, October 5, 2018 1:51:43 PM	Tagging 05 October 2018 IC Flight Build to the instrument, IP-ICS_FLT_IC_3112_ExtendSpectralSurvey_Sunset_20181003.bin and SAGE-III_ICS_Che
134		croller	Friday, October 5, 2018 1:50:52 PM	Committing 05 October 2018 IC Flight Build to the instrument, IP-ICS_FLT_IC_3112_ExtendSpectralSurvey_Sunset_20181003.bin and SAGE-III_ICS_Checksum_ExtendSpectra
133		croller	Friday, October 5, 2018 1:46:51 PM	Tagging 05 October 2018 IC Flight Build to the instrument, IP-ICS_FLT_IC_3112_ExtendMirrorCal_AllSolars_20181003.bin and SAGE-III_ICS_Checksum_ExtendMirrorCal_AllSol

```
Tagging 05 October 2018 IC Flight Build to the instrument, IP-ICS_FLT_IC_3112_ExtendSpectralSurvey_Sunset_20181003.bin and SAGE-III_ICS_Checksum_ExtendSpectralSurvey_Sunset_20181003.bin

From ics_tab:

    SAGE-III_ICS_DFLTLoadImage.bin (Default Image, AS0_3112, svn294)
    SAGE-III_ICS_USERLoadImage.bin (User Image, AS0_3111, svn226)
    IP-ICS_FLT_IC_3112_ExtendSpectralSurvey_Sunset_20181003.bin (IC Control Table, svn 346)
    SAGE-III_ICS_Checksum_ExtendSpectralSurvey_Sunset_20181003.bin (checksum Table, SVN 347)
    CPCTRL.bin (Copy Control Table, svn 332)

from ccd_tab revision tag# 230:
    CCD_FSW106b_TEC125_135_265

Reference CCR 672 "ICCT Updates for the Extended Mirror Calibration and the Extended Spectral Survey (Sunset) Activities"
```

Path	Action	Copy from path	Revision
/tags/IP-ICS_FLT_IC_3112_ExtendSpectralSurvey_Sunset_20181003	Added	/trunk	134

Showing 100 revision(s), from revision 36 to revision 135 - 1 revision(s) selected, showing 1 changed paths

☐ Show only affected paths
☑ Stop on copy/rename
☐ Include merged revisions

Examples — Ways We Can Improve

- SAGE III Science and Ops have shared needs for software for processing instrument data. Repositories (until recently) have separate and sometimes redundant development.
- SCF needs access to algorithms and processing developed by both Science and Ops. Currently sharing development and collaborating is difficult.
- Science has multiple instances of redundant processing, e.g. data readers, analysis, etc. that can be shared among members to allow for less time spent creating low-level processing tools and more time producing new analyses through shared and collaborative efforts.
- Parts of the released data product are not tracked alongside the algorithm. Attempts are being to remedy this using GitHub and tagging versions, but no process exists as yet going forward.

Examples — Documentation

Sphinx

A powerful documentation generator written in and for Python, though its coverage has expanded through tools such as Breathe. Using defined markup in reStructuredText, documentation including HTML, LaTeX, ePub, and more can be generated from the source.

- Projects, such as Numpy and Google, provide versions of comment markup that are supported by Sphinx and well documented
  - Numpydoc
  - Google Python Style Guide
- Similar tools exist for other languages (these are links!)
  - Michael Galloy's IDLdoc.
  - Doxygen — Does basically everything.
  - Breathe — Merges Doxygen and Sphinx capabilities.

Laziness is a programmer's virtue – Write your documentation once, ideally while writing the code, and both code commenting and end-user information are done.

Resources

- Joel on Software — https://www.joelonsoftware.com/
- A Visual Guide to Version Control — https://betterexplained.com/articles/a-visual-guide-to-version-control/
- Sphinx — http://www.sphinx-doc.org/en/master/
- JIRA — https://www.atlassian.com/software/jira
- GitHub — https://github.com/
- Trac — https://trac.edgewall.org/
- Virtues of the Perl Programmer — https://www.oreilly.com/library/view/programming-perl-4th/9781449321451/ch24s03.html

Projects

Project Boards

GitHub can be used for project management and/or task tracking using project boards. (kanban board) Projects provide organization and prioritization.

- Can be specific to a user, organization, or repository and track the issues, pull requests, and/or notes.
- GitHub provides several board templates as well as allows for custom designs.
- The boards are setup with a series of columns.
- Workflow automations can be configured to automatically add new or existing items to columns.
- Repositories can be linked to existing projects.

Issues/Ticketing

What are Issues?

System to track code bugs, features, and enhancements.

- Labels — Facilitate grouping, categorizing, and color-coding of issues. Issues can have multiple labels.
- Milestones — Used to tie issues to a project phase, specific feature, time period, or build (e.g. ICSv3.1.13)
- Assignee — Person or team that is working on resolving the issue
- Comments — Allow for discussion about an issue similar to how NCRs can have tracking comments

Issues/Ticketing

- Notifications — Can be used to notify users of new issues or forward motion on an issue
- @mentions — Use @username to reference a GitHub user or team
- References — Used to relate issues or show dependency between issues. Issues can also be referenced in commits.

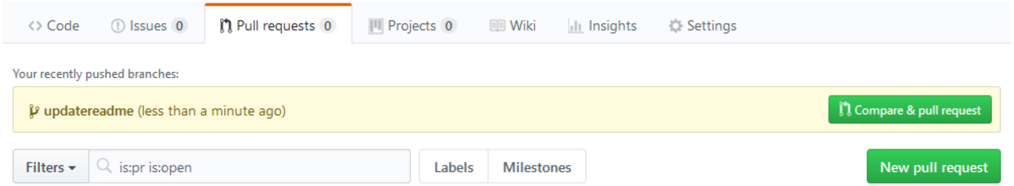
Filter to Find Important Issues

Milestones, labels, and assignees can all be used to filter and track a group of issues. You can search by keyword, state (i.e. open, closed), assignee in the search bar.

Pull Requests

Pull requests are a way to have a mini-CCR/MRB.

- A pull request is a **request** for a repository maintainer to **pull** (merge) concurrent changes into the master branch.
- Pull requests are generated from branches or forks on GitHub.



Your recently pushed branches:

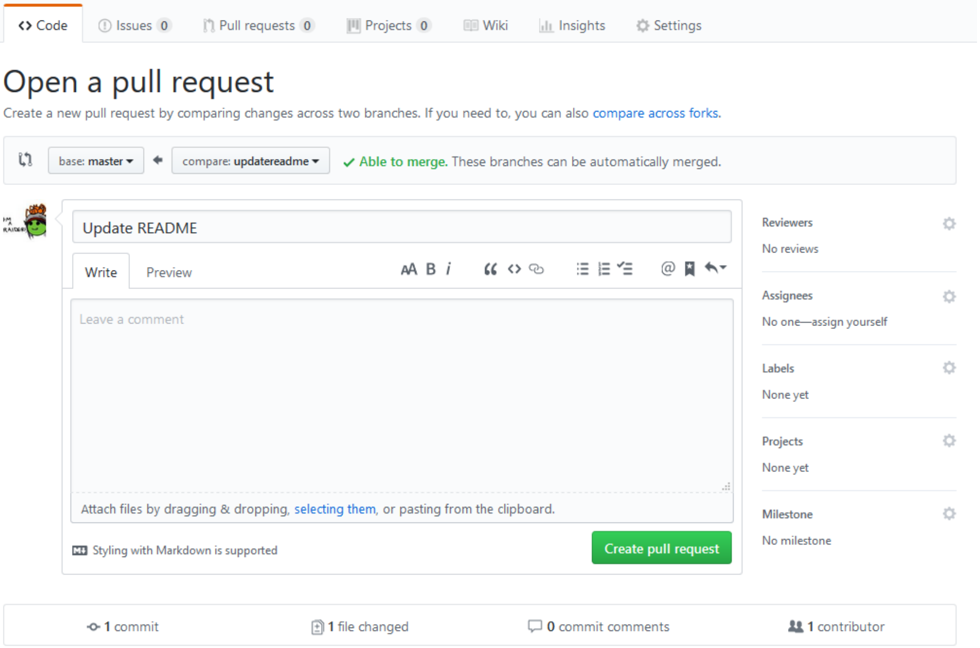
updatereadme (less than a minute ago)

Pull Requests

To start a Pull Request, click "New Pull Request"

- You will choose a base branch (left) and the branch to merge from (right — Note the arrow)
- The interface provides a quicklook at the changes.
- When ready, click "Create Pull Request"
    - For new branches, GitHub will also give you an option to jump to this step from the Repository dashboard.

Your Pull Request must follow rules for merging.



Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

base: master ← compare: updatereadme ✓ Able to merge. These branches can be automatically merged.

Update README

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

Styling with Markdown is supported

Reviewers	No reviews
Assignees	No one—assign yourself
Labels	None yet
Projects	None yet
Milestone	No milestone

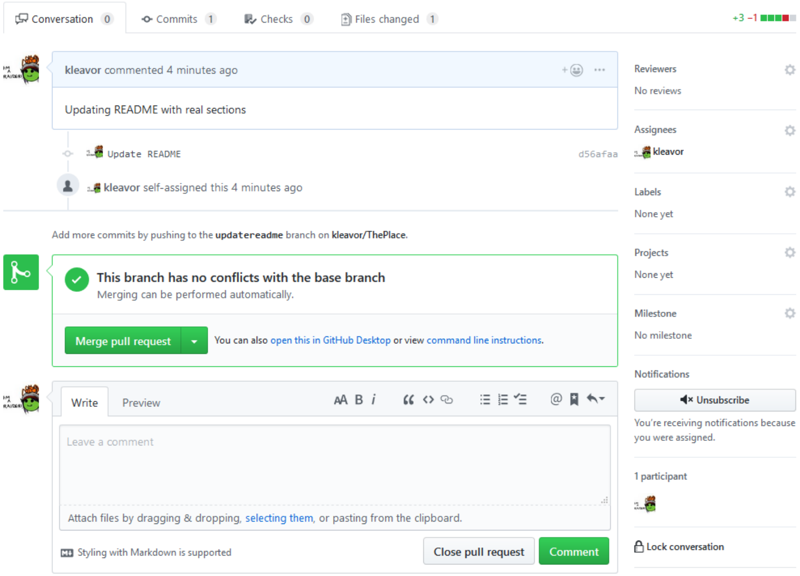
1 commit · 1 file changed · 0 commit comments · 1 contributor

Pull Requests

But what can I do with Pull Requests?

- Individuals can be assigned as Reviewers or Assignees
    - Reviewers can be anyone with `READ` access.
    - Reviewers can be re-added once suggested changes are made to re-review.
    - Assignees are placed as developers expected to perform the merge.
- Pull Requests can be given labels, placed under a project, or be associated with a milestone.
- Specific users can additionally be mentioned using @USERNAME. Assigned and mentioned users are subscribed to discussion.
- The Pull Request becomes a record of the conversation and decisions that led to a change being incorporated.
    - The Pull request contains the commit history and a complete changelog.
- When the change is ready to merge, click “Merge Pull Request”

Pull Requests



Conversation 0 · Commits 1 · Checks 0 · Files changed 1 · +3 −1

kleavor commented 4 minutes ago

Updating README with real sections

Update README d56afaa

kleavor self-assigned this 4 minutes ago

Add more commits by pushing to the **updatereadme** branch on **kleavor/ThePlace**.

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request — You can also open this in GitHub Desktop or view command line instructions.

Write | Preview

Leave a comment

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

Styling with Markdown is supported

Close pull request | Comment

Reviewers: No reviews

Assignees: kleavor

Labels: None yet

Projects: None yet

Milestone: No milestone

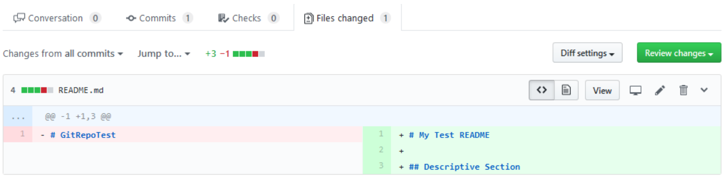
Notifications: Unsubscribe

You're receiving notifications because you were assigned.

1 participant

Lock conversation

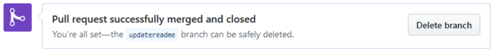
Pull Requests



```
4 README.md
@@ -1 +1,3 @@
1 - # GitRepoTest
1 + # My Test README
2 +
3 + ## Descriptive Section
```

In Closing...

- When you merge a pull request, you are given the usual commit log
  - Enter a short (≤50 char) description. By default this is "Merge ..."
  - Provide a more descriptive commit log for the merge. If this closes a ticket, inserting language like "Closes #432" will allow GitHub to automatically close the ticket.
- After merging, GitHub will also give you the option to delete the feature branch (if created from a branch, not a fork). It can be restored!


Pull request successfully merged and closed

You're all set—the `updatereadme` branch can be safely deleted.

Create a Repository

- On the main GitHub page (either https://developer.nasa.gov or https://github.com) click “New Repository” or “New”.
- Give your repository a name and description.
- Select Public or Private (your choice).
- Click “Create repository”.
- Create a folder on your computer to store the repository.
- In the new folder right click and select “Git Bash Here”
- In the command line perform:

```
echo "# GitRepoTest" >> README.md
git init
git add README.md
git commit -m "first commit"
```

Create a Repository

For Internal GitHub:

```
git remote add origin git@developer.nasa.gov:<YourUserName>/<Name of your repo>.git
git push -u origin master
```

For Public GitHub:

```
git remote add origin git@github.com:<YourUserName>/<Name of your repo>.git
git push -u origin master
```

Enter your password when prompted.

The SSH username before @ is always "git" not your user name.

You did it!