“Best” Practices

- Standard and established procedures that aim to produce optimal results
- Evidence from research and experience support the use of the practice
- Suitable for widespread adoption (says who?)

Paraphrasing from Merriam-Webster dictionary

Non-software best practices

From life

Mechanical engineering

Electrical engineering

Be Careful!

- What is a “best” practice
  - Sometimes this is codified in a standard
  - Sometimes its more like a rule of thumb
  - Sometimes there is contradictory information
  - Use a reputable source and your own judgment
  - How serious it is depends on what you are doing

The Cost of a Defect

- While you are thinking/typing $0
- At compile time $1
- At runtime on your machine $20
- At runtime during testing $120
- In the customers home $1,000
- On Mars: $1,000,000,000
- When your automated cat caretaker robot fails while you are on a two-week vacation… (depends on if you like cats)

Software Best Practices

- The Cowboy Coder
- Hacks
  - Code whatever as long as it works
  - What about?
    - Documentation
    - Testing
    - Security
    - Maintainability
    - Reliability

Best Practices For Any Language

- Comments!
  - Do not write a function before commenting what it does and what arguments it takes
- Formatting – use a consistent formatting style
- Version Control (git)
- Build process (IDE, Makefile)
- Testing
- Use the right documentation

Use the right documentation

- Official TI documentation
- http://pubs.opengroup.org/onlinepubs/9699919799/
- TI wiki, TI forums (will often come up on google)
  - Tiva is similar to Stellaris, some stellaris info applies equally to Tiva
- www.cppreference.com
- www.cplusplus.com

Different Goals – Different Standards

- “You can’t fix every bug” - Anonymous employee at anonymous company
- Well, this is not entirely true, but
  - Constrained by resources (time/money)
- May be possible to prove that code is bug-free
  - Formal verification is still an ongoing research area

Buzz words in software best practices

- Agile
- Test Driven Development
- Xtreme programming
- Object oriented programming
- Functional programming
- Aspect oriented programming
- Procedural programming
- Generic programming
- Data-driven programming
- SCRUM
- Why do we have all these things?

Programming Languages that are “safer” than C

- ADA
- Go
- Java
- C#
- Haskell
- Lisp
- C++
- Active research into formal verification
  - Coq
  - ATS
  - TLA+

Problems with C?

- Memory leaks
- Stack overflow
- Heap overflow
- Buffer overflow
- Race conditions
- Deadlocks
- Error handling

Why use C?

- Easier and more portable than assembly
- Provides low-level control
- Everyone else is doing it!
  - Collective experience + tools + standardization
  - Static analysis (linter) can help

C Standard

- First there was K & R C
- Then ANSI C
- Then C89
- Then C99
- Then C11

Some C Coding Standards

- MISRA-C
- Jet Propulsion Lab C Standard
- https://lars-lab.jpl.nasa.gov/JPL_Coding_Standard_C.pdf
- Barr Group Embedded C Coding Standard

  https://barrgroup.com/Embedded-Systems/Books/Embedded-C-Coding-Standard

What do coding standards provide?

- Rules and guidelines for coding
- Justifications for those rules
- Formal standard that must be met for compliance purposes

Why?

- Help avoid human error
- Make it easier for a team to work on a project
- Eliminate problems before they occur
- Prevent undefined behavior

Compiler flags

For ti-cgt:

- -c99 (Use C99 standard)
- -pdr (enable remarks)

For gcc

- -std=c99 (Use C99 standard)
- -Wall -Wextra (more warnings!)

Why C99?

- By now it is widely supported
- Enables features that improve your life
  - <stdint.h> uint32_t uint8_t, etc
  - for(int i = 0; i < 100; ++i)

Example Time!

- 0 argument functions
- Initialize all variables
- Turn on all compiler warnings

Real Time Operating System

- Abstract interface to hardware (like any OS)
  - Portability
- Satisfy constraints due to timing requirements in the physical world
- Precise timing and directly specified task preemption

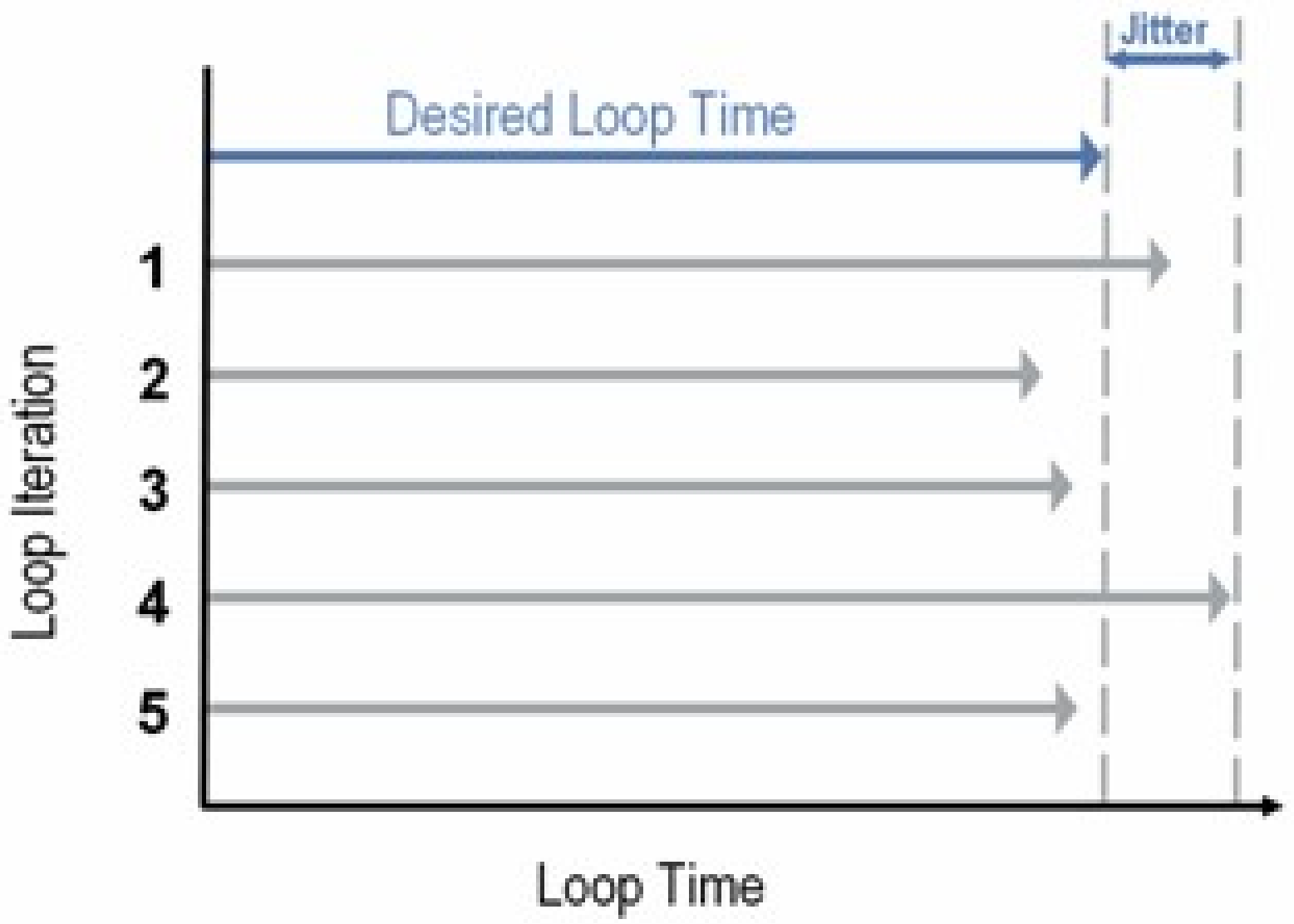
Features of Real-time OS's

- Tasks
- Scheduler
- Interrupt handling
- Inter-task communication
  - Mutexes, queues, etc.
- Memory allocation

Properties of RTOS Schedulers

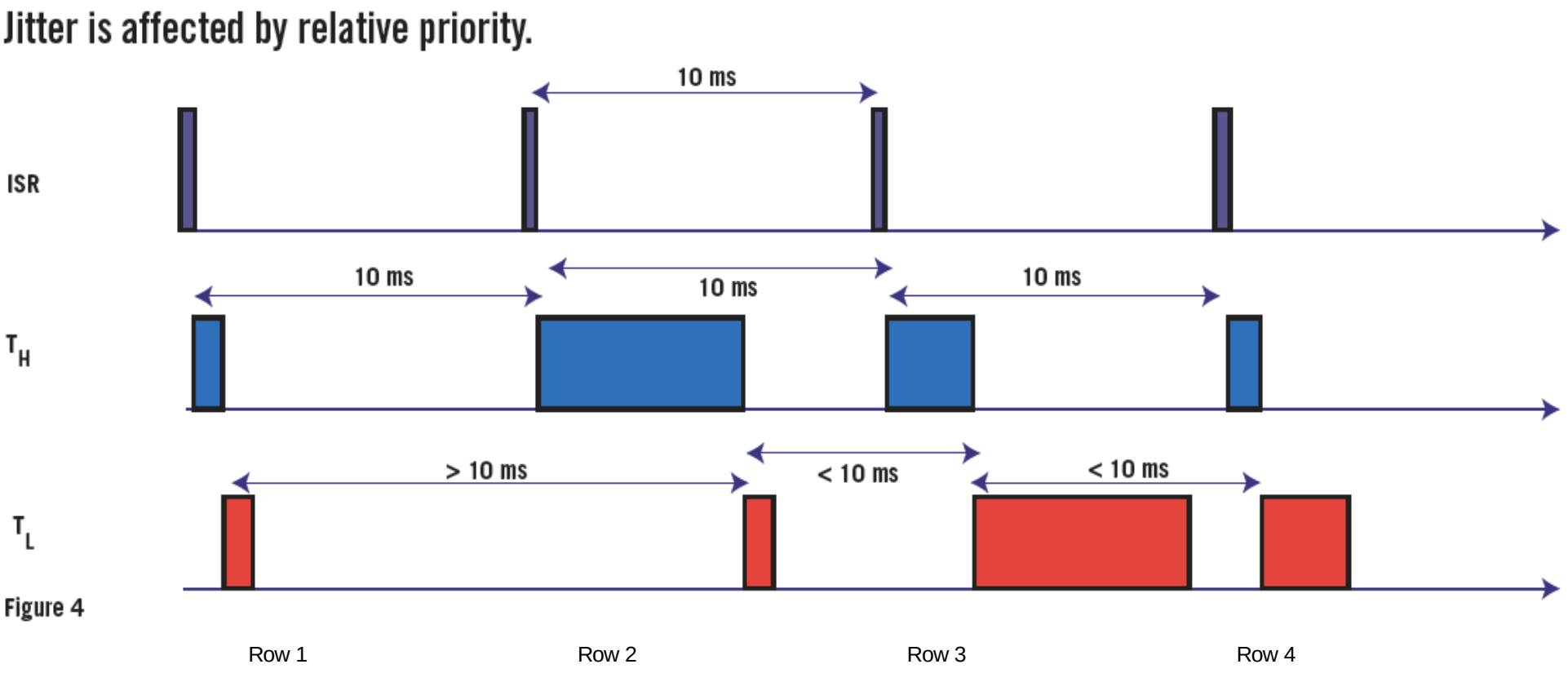
- Deterministic execution
- Timing Guarantees
  - Hard real-time
  - Soft real-time

Real Time Operating System Tasks



http://www.ni.com/cms/images/devzone/tut/3938_2.jpg

Jitter



http://eetimes.com/ContentEETimes/Images/Design/Embedded/2010/1110/1110esdBarr04.gif

When do we need good timing?

- Safety-critical
  - Pressing the brake shouldn’t wait for the GPS navigation system to load before slowing the car
- Hopping robot
  - Need to apply force in a tight window for optimal efficiency
- Robot arm
  - want joints to move synchronously
- Brushless motor
  - tracking a quickly varying input signal

Some RTOSs

- VxWorks
- QNX
- RTLinux

RTOS available for TM4C series

- TI proprietary RTOS
- FreeRTOS (probably)
- ChibiOS (maybe, unofficially)

But do we need an RTOS?

- Anything that can be done with an RTOS can be done coding on “bare-metal”
- RTOS take more on-chip resources
- Also adds initial complication
  - Learning the OS
  - Setting it up
- As your project gets more complicated the case for an RTOS becomes stronger

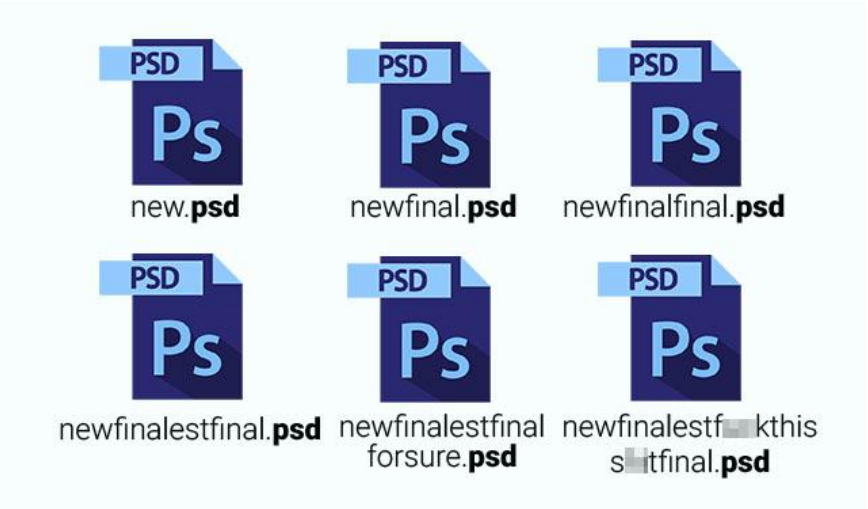
Git Intro and Best Practices Case Study

Robot Design Studio
1/31/2018

What is version control and why use it?

- Version Control System (VCS) records changes to a file (or files) over time so that you can recall specific versions later
- Uses
  - Collaboration
  - Back-ups
  - Debugging
  - Managing releases
  - Documentation
  - Sharing code publicly
- Vital tool in the life of a software developer

Avoid This:

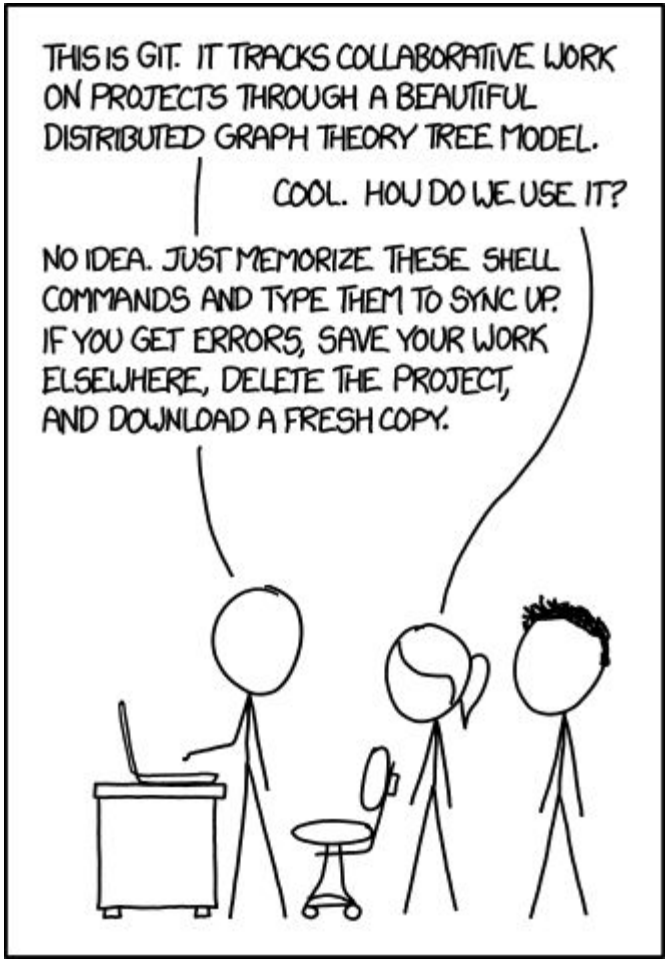


Why Git?

- There are many VCS available (Mercurial, Subversion, Rational ClearCase, Git, Perforce)... Why choose Git?
- Git is popular (in demand skill)
  - ~70% of Stack Overflow use Git (2017 Community Survey)
  - Many companies use Git (Google, Facebook, Microsoft, Twitter, Netflix, etc.)
- Git is fast (compared to other options)
- Git is distributed (not centralized)
  - Flexible and robust
- Git is free and open-source

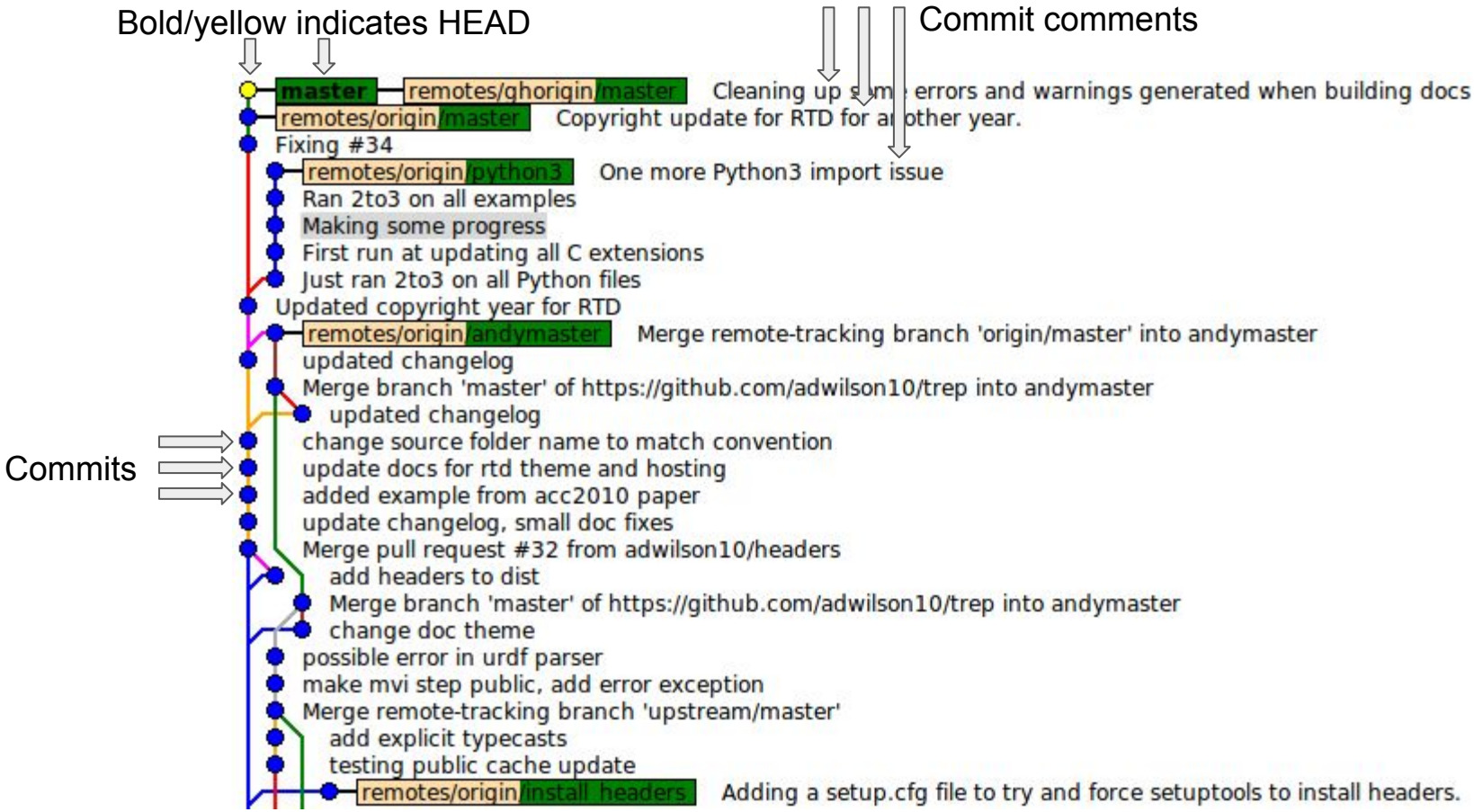
Why not Git?

- Git is challenging to learn!
  - Jargon heavy
    - Commits, branches, HEAD, remotes, staging, pulling, pushing, stashing, reverting, rebasing, checking, diffing, patching, reflogs, cherry-picks, clones, forks, squashing, blaming, bisecting, archiving, submodules, subtrees, repacking, pull-requests, fetching, resetting,...
  - Huge number of commands and options
  - Flexibility => complexity



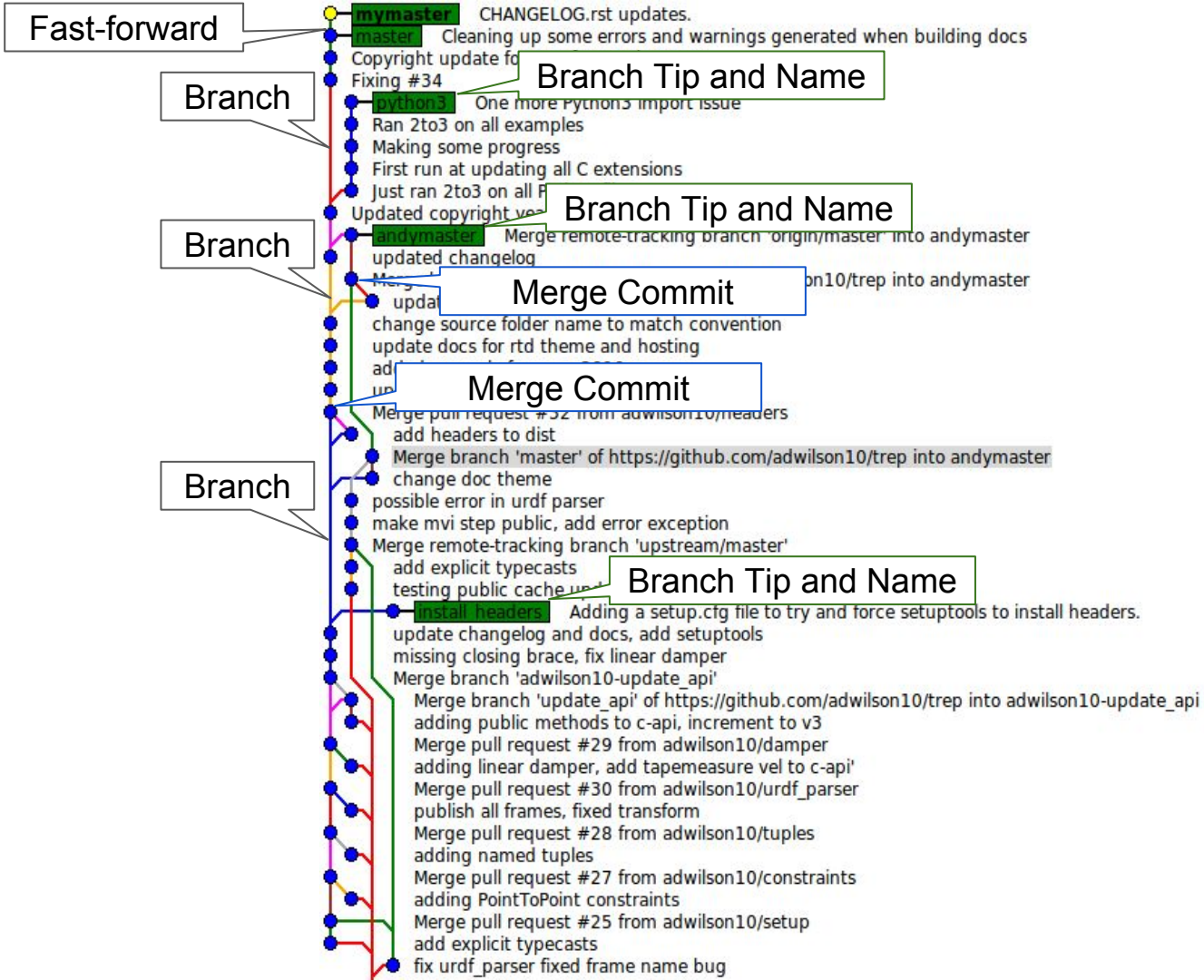
Important concepts: Commits, Checkout, HEAD

- Commit:
  - As a noun: A single point in the Git history; the entire history of a project is represented as a set of interrelated commits. The word "commit" is often used by Git in the same places other revision control systems use the words "revision" or "version".
  - As a verb: The action of storing a new snapshot of the project’s state in the Git history, by creating a new commit representing the current state of the index and advancing HEAD to point at the new commit.
- HEAD:
  - The current state of your repository (which commit you’re looking at)
- Checkout:
  - Update the state of your working tree by moving your HEAD (changes all files)


Bold/yellow indicates HEAD
Commit comments
master
remotes/ghoriqin/master
Cleaning up some errors and warnings generated when building docs
remotes/origin/master
Copyright update for RTD for another year.
Fixing #34
remotes/origin/python3
One more Python3 import issue
Ran 2to3 on all examples
Making some progress
First run at updating all C extensions
Just ran 2to3 on all Python files
Updated copyright year for RTD
remotes/origin/andymaster
Merge remote-tracking branch 'origin/master' into andymaster
updated changelog
Merge branch 'master' of https://github.com/adwilson10/trep into andymaster
updated changelog
Commits
change source folder name to match convention
update docs for rtd theme and hosting
added example from acc2010 paper
update changelog, small doc fixes
Merge pull request #32 from adwilson10/headers
add headers to dist
Merge branch 'master' of https://github.com/adwilson10/trep into andymaster
change doc theme
possible error in urdf parser
make mvi step public, add error exception
Merge remote-tracking branch 'upstream/master'
add explicit typecasts
testing public cache update
remotes/origin/install_headers
Adding a setup.cfg file to try and force setuptools to install headers.

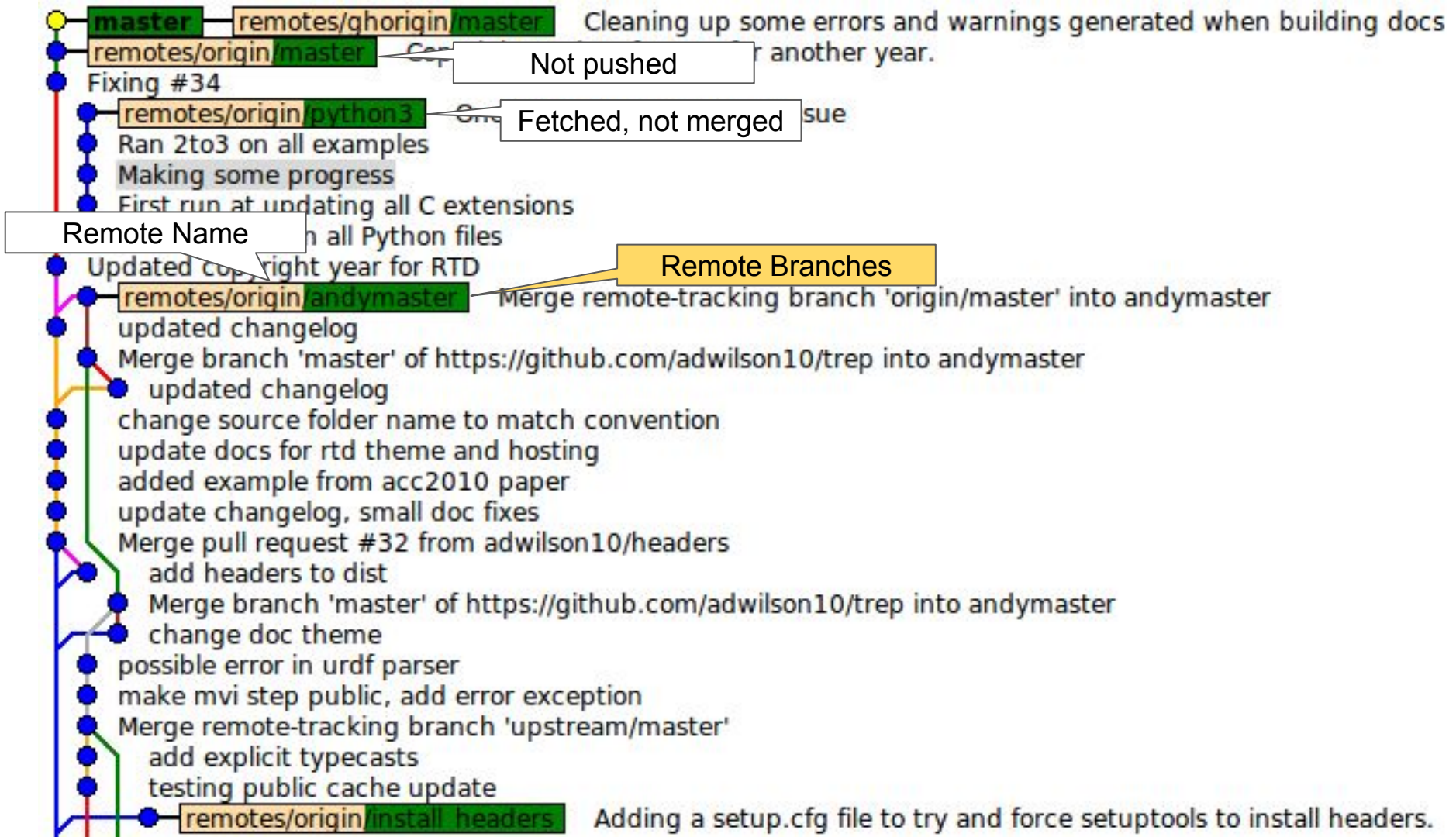
Important concepts: Branching and Merging

- Branch:
  - Active line of development; most recent commit on a branch is called the “tip”.
  - Also thought of as a movable pointer that points to commits.
  - Single repository handles arbitrary number of branches, but working tree is associated with one at a time (HEAD points to a single checked-out branch)
  - Default branch when creating a repository is called “master” (you don’t need a master branch).
- Merge:
  - Verb: To bring the contents of another branch (possibly from an external repository) into the current branch. Merging is performed by an automatic process that identifies changes made since the branches diverged, and then applies all those changes together. In cases where changes conflict, manual intervention may be required to complete the merge.
  - Noun: unless it is a fast-forward, a successful merge results in the creation of a new commit representing the result of the merge, and having as parents the tips of the merged branches. This commit is referred to as a "merge commit", or sometimes just a "merge".


Fast-forward
mymaster
CHANGELOG.rst updates.
master
Cleaning up some errors and warnings generated when building docs
Copyright update fo
Fixing #34
Branch Tip and Name
Branch
python3
Ran 2to3 on all examples
Making some progress
First run at updating all C extensions
Just ran 2to3 on all
Updated copyright ye
Branch Tip and Name
Branch
andymaster
Merge remote-tracking branch 'origin/master' into andymaster
updated changelog
Merge Commit
change source folder name to match convention
update docs for rtd theme and hosting
Merge Commit
add headers to dist
Merge branch 'master' of https://github.com/adwilson10/trep into andymaster
Branch
change doc theme
possible error in urdf parser
make mvi step public, add error exception
Merge remote-tracking branch 'upstream/master'
add explicit typecasts
Branch Tip and Name
install_headers
Adding a setup.cfg file to try and force setuptools to install headers.
update changelog and docs, add setuptools
missing closing brace, fix linear damper
Merge branch 'adwilson10-update_api'
Merge branch 'update_api' of https://github.com/adwilson10/trep into adwilson10-update_api
adding public methods to c-api, increment to v3
Merge pull request #29 from adwilson10/damper
adding linear damper, add tapemeasure vel to c-api'
Merge pull request #30 from adwilson10/urdf_parser
publish all frames, fixed transform
Merge pull request #28 from adwilson10/tuples
adding named tuples
Merge pull request #27 from adwilson10/constraints
adding PointToPoint constraints
Merge pull request #25 from adwilson10/setup
add explicit typecasts
fix urdf_parser fixed frame name bug

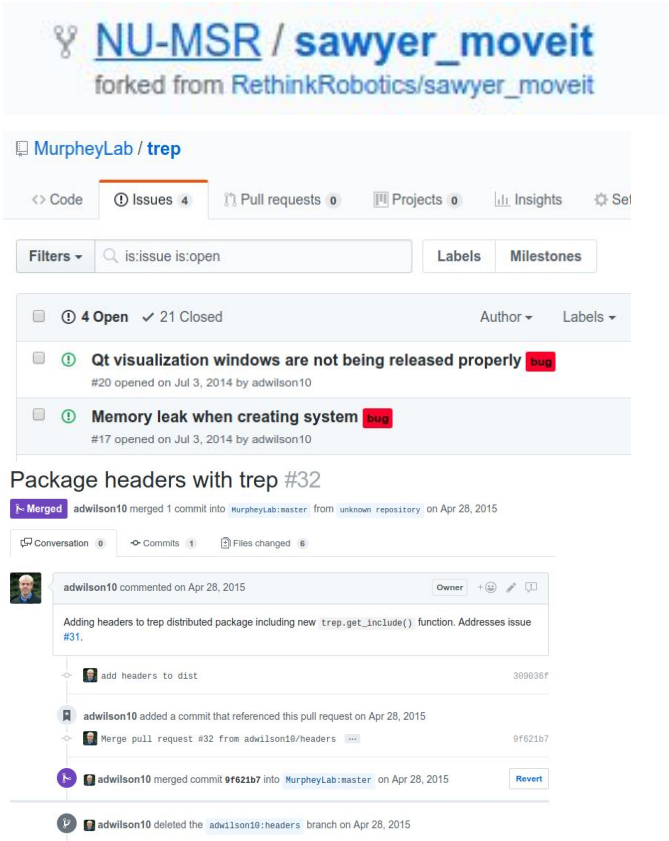
Important concepts: Pushing+Fetching+Cloning+Remotes

- Remote:
  - A repository which is used to track the same project but resides somewhere else (another directory, GitHub, BitBucket, GitLab, private server, Dropbox, etc.)
- Clone:
  - Copy a repository into a newly created directory
  - This is how you get a local copy of a repository off of GitHub (or another remote location)
- Push:
  - Move content from a local branch onto a remote branch
  - Syncs data from a local machine to GitHub
  - The remote’s branch tip needs to be an ancestor of the local branch tip
- Fetch:
  - Download information about remote branches/commits to local repository
  - Does not modify local work tree
  - A “pull” combines a fetch with a merge -- does modify local tree, not recommended for beginners


master
remotes/ghoriqin/master
Cleaning up some errors and warnings generated when building docs
remotes/origin/master
Not pushed
another year.
Fixing #34
remotes/origin/python3
Fetched, not merged
sue
Ran 2to3 on all examples
Making some progress
First run at updating all C extensions
Remote Name
all Python files
Updated copyright year for RTD
Remote Branches
remotes/origin/andymaster
Merge remote-tracking branch 'origin/master' into andymaster
updated changelog
Merge branch 'master' of https://github.com/adwilson10/trep into andymaster
updated changelog
change source folder name to match convention
update docs for rtd theme and hosting
added example from acc2010 paper
update changelog, small doc fixes
Merge pull request #32 from adwilson10/headers
add headers to dist
Merge branch 'master' of https://github.com/adwilson10/trep into andymaster
change doc theme
possible error in urdf parser
make mvi step public, add error exception
Merge remote-tracking branch 'upstream/master'
add explicit typecasts
testing public cache update
remotes/origin/install_headers
Adding a setup.cfg file to try and force setuptools to install headers.

Important Concepts: Git vs GitHub

- GitHub is probably the most common web-based Git repo hosting service
  - There are others (BitBucket, GitLab, etc.)
- You can use Git without GitHub
  - Due to popularity, the two are often conflated
- GitHub specific concepts
  - Forking: make a copy of a GitHub repo on another user's account (example)
  - Issue Reporting: Report bugs, comment on bugs, assign people to fix bugs, and track bugs (example)
  - Pull Requests: Recommend fixes to bugs; if accepted, commits on one fork will be merged into a branch on another fork (example)



Git Best Practices

1. Commit often, yet carefully (follow all other rules)
2. Commit related changes, write good commit messages, and use consistent email/name for your configuration (support good documentation)
3. Avoid committing broken or untested content (especially when pushing to a shared repo)
4. Never commit **any** files that are produced as part of a compilation or development process (executables, libraries, archives, local config files, autosaves, backups, etc.)
5. Avoid committing binary data files (images, videos, databases, etc.)
6. Avoid make unnecessary whitespace/formatting changes, especially in a shared repository (try to adhere to any standards set in place before, and what your team has agreed upon)
7. Avoid rewriting Git history such as `git reset --hard`, `git rebase`, and forcing pushing/pulling (especially important in a shared setting)
8. Work with your team to establish a set of rules that will govern team member branching, committing, pushing/pulling, and merging strategies (often called a workflow)
    - Atlassian Git Workflow, Pro Git Workflows, Git Flow model
9. When in doubt, try it out (easy to copy a repo, and test a procedure)
10. Design and manage where/what you push/pull to adhere to established workflow, but also support individual development

How do I learn Git?

- Practice, practice, practice!
  - Use Git for all projects, and it will soon become second-nature
  - More valuable than tutorials --- context provided
- When stuck, don’t develop workarounds
  - An operation you don’t know how to do, or an error message you don’t understand? Take the time to learn the basics and understand the *right* way to deal with your situation
  - This may be slower at first, but will pay off in the long run
- Do some background reading
  - There are thousands of great Git resources online
  - Strive to be a developer who understands how things work, and how to best utilize Git’s features

Recommended resources

- My Git tutorial: http://nu-msr.github.io/embedded-course-site/resources/git_intro.html
- GitHub guides: https://guides.github.com/
- Code School+GitHub tutorial: https://try.github.io
- Code School Try Git course: https://www.codeschool.com/courses/try-git
- Git IMMERSION: http://gitimmersion.com/
- Git documentation available online and in man pages (really good docs)
  - Git glossary: https://git.github.io/htmldocs/gitglossary.html
  - gittutorial and gittutorial-2
- Pro Git Book: https://git-scm.com/book/en/v2
  - Available as PDF, epub, mobi, and HTML
  - Hands down, the most thorough and detailed Git resource I've encountered. I often read sections/chapters from this when I want to understand something I don't know about

Case Study: Trep

- Main site: https://nxr.northwestern.edu/trep
- API documentation: http://trep.readthedocs.io/
- GitHub code: https://github.com/MurpheyLab/trep/
- Releases:
  - ROS Package: http://wiki.ros.org/python_trep/
  - PyPi: https://pypi.python.org/pypi/trep/
  - Macports: https://github.com/macports/macports-ports/blob/master/python/py-trep/Portfile