PUG CHALLENGE EXCHANGE AMERICAS

Git your source code under control, for free!

June 29, 2016 Chris Hawkins

git

About the Speaker

- Progress developer since V5 1988
- Director R & D at ASA Automotive Systems Inc.
- We develop software for Tire Dealers (POS/OE, Accounting, Inventory, many interfaces)
- Small team of developers, several products, including legacy ones. Millions of lines of code.
- Hundreds of customer systems on various releases of our products.
- Not a Git expert

About this Presentation

- A brief introduction to source control and Git
- Common Git usage
- Time travel
- Distributed Git and our workflow
- Tools
- Questions

What is Source Control?

- Commonly referred to as Version Control or Revision Control
- A secure location for source code
- Tracking changes to source files (and other artifacts)
- Allows controlled releases, typically with a release or revision number 10.2B or 11.6 etc.
- Safety net for programmers, can revert to older copies of code if needed
- Recreate the exact source code for previous releases. Allows patching of old releases (a necessary evil)
- Collaborate with other developers in a controlled way

What is Git?

- Git was initially designed and developed in 2005 by Linux Kernel developers including Linus Torvalds.
- Strong support for non-linear development (easy branching)
- Distributed development (each developer has a local copy of the full development history)
- Compatibility with existing systems/protocols (ssh, https)
- Efficient handling of large projects
- Cryptographic authentication of history
- Is both a repository and/or manager of local source code

Why Git?

- Git is free! And open source. With paid commercial hosted options.
- Very popular! Eclipse Foundation (May 2015) “Git is the most widely used source code management tool - 43% of professional developers use it”
- All my questions were answered with a Google search
- E.g. Stackoverflow has 70k questions and 55k answers
- It’s stable (11 years old), robust, fast, redundant
- GitHub and BitBucket are the go-to places for open source projects
- New developers more likely to be familiar with it

Why Git?

- Most IDE's, task tracking & build tools have Git support
- Everything can be done using the command line - easily scriptable
- Use with build tools like PCT Ant
- Also Java implementation
- Excellent free GUI tools for most activities
- There are even iOS client apps for Git
- Git can be tricky but keep in mind it gives you control over a complex process

Why Git?

- Complete confidence to make copies of your source anywhere
- For dev, test, support, on multiple machines?
- Every developer, wherever they are
- Each copy is the **full** history
- Access to code from any previous release
- Or any work any developer has made available
- Every copy becomes a backup, no single point of loss
- No “locking” of programs for team development

Git is not File Revisions

- Familiar with file based source control tools like CVS? Git is not file based, no file versions like 1.1, 1.2 etc.
- Git takes snapshots of the entire directory tree. This snapshot is uniquely identified by a SHA-1 key.
- Git tracks the changes to all files at each commit, very efficiently. Including file **deletes** & **renames**.
- You have access to these changes and can re-apply
- Changes your working directory to the code backwards/forwards in time very quickly
- Compare changes over time (and reapply for patches/hot-fixes)
- See who changed each line of code (git blame)
- It's difficult to loose anything (once committed)
- The past is immutable (like an accounting ledger, changes are always by adding)

Getting Started with Git

- Read - the (free) Pro Git book has a lot of detail
- Learn by experimenting - a lot
- Start fresh or import code history? How much history?
  - There is inbuilt support for SVN
- Decide on hosting, in house Gitolite or cloud GitHub, BitBucket etc.
- Not just hosting, they have code view/review
- It is distributed but there is likely one “origin” repository.
- Decide on workflow that works for your team(s)

Getting Started with Git

- Download Git - Windows, Mac, Linux, Solaris
- https://git-scm.com/downloads
- Download GUI tools
- Windows TortoiseGit https://tortoisegit.org
- Windows & Mac Source Tree https://www.atlassian.com/software/sourcetree
- For Eclipse install eGit http://www.eclipse.org/egit/

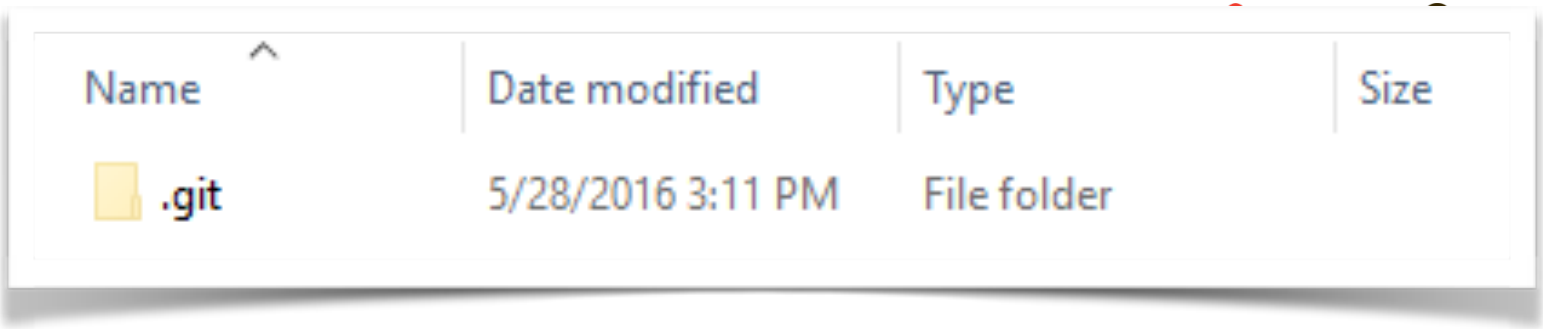
Identify Yourself

- Before using Git for the first time, you need to tell Git who are. It will use your name and email on each commit (see Git blame)
- Git config can be global, or local. Use global so all your repos (on this PC) have your id.

```
git config --global user.name "Chris"

git config --global user.email
chris@myemail.com
```

Create a Git Repo



- Create from scratch:

`git init`

- Or copy locally from another directory
- Or clone from a repo using Git protocols (locally,https,ssh)

`git clone`

- Look at the .git directory. That's where Git stores everything. Don't delete it.

`git status`

- Git says it's "On branch master". We'll discuss branches later
- There is also a "bare repository". This is where you have just the .git directory and no files.

Git Notices Changes

- Copy files from a local directory, into the directory. Then ask Git for a status
- Git notices the new files as “untracked”. All files are like this until you tell Git to manage them.
- Tell Git to ignore files using a .gitignore file.
  - Very important to ensure never add unwanted files e.g. Progress ABL .r code files
- Use wildcards e.g. ignore all r code *.r
- The .gitignore file is not ignored by default

Time to Commit

- Make our first commit to the Git repo

`git add .`

- Using the period wildcard is a quick shortcut but takes all files (unless in .gitignore file). Safe to use if you keep a clean directory.
- Are we done? No, the files are only “staged”.

`git status`

- Git says “Changes to be committed”
- Why the extra stage step?
- Very useful when working directory has many new or changed files but you want to commit them in separate groups.
- You can skip the stage step (using a different command) but I don't recommend it. A personal preference.

Make History

- To commit the changes to the repo, with a commit comment:

```
git commit -m "My first commit"
```

- Or, without the -m brings up a text editor to add the comment
- Git tells you what was committed but you can verify at any time using the git status command

```
git status
On Branch master
nothing to commit, working directory clean
```

- Use git log to see our newly created history and the SHA-1 key generated. The git log command has many options for viewing history - you will want to learn more about it.

```
git log
commit 1099e19124902281b936c5d436bdc45b17b8d00c
My first commit
```

The Stages of Git

- Files not managed by Git are “Untracked”
- Once managed by Git, there are 3 stages
  - Committed. The changes are in your local Git repo.
  - Staged. A snapshot of changes ready to be committed.
  - Modified. There are changes to the files managed by Git, compared with what is staged or committed.


Time Travel

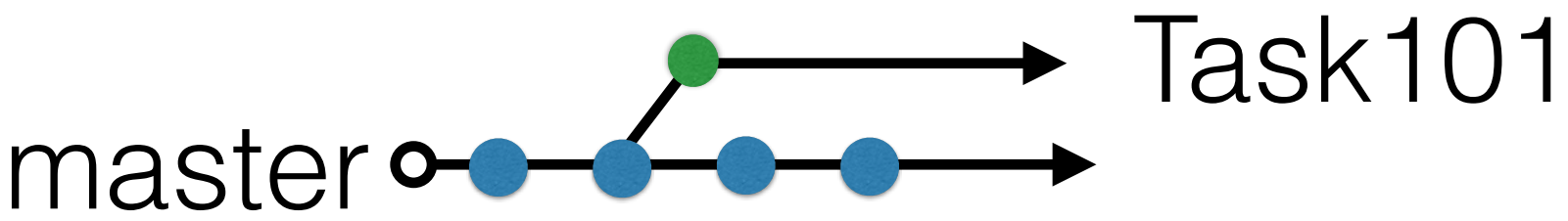
- All Git repos have the complete source history, so you can travel in time, locally.
- Just to look, or to create an alternate future from any previous time.

Time Travel

- Git lets you travel back and forwards in time. You can see the code as it was.
- Git log shows the commit messages and much more
- Git checkout allows you to access the code as of that time
- Like a time traveller, you cannot change the past, only create an alternate future. Using a Git “branch”.
- NOTE: Changing code without a branch (a “detached head” state), does not change the history, just your working copy of the files.

Branch to an Alternate Future

- To create an alternate future, you first create a branch
- Create branches for everything, small fixes, major features, experimenting
- **Never make changes on the master branch** - keep it open for merging
- Branch names are just names.
- The default Git branch is “master”. It’s purely a convention. What matters is the SHA-1.
- Creating a branch is instant, so create one anytime to start tracking changes.



Let's Branch

- Create a branch, from master

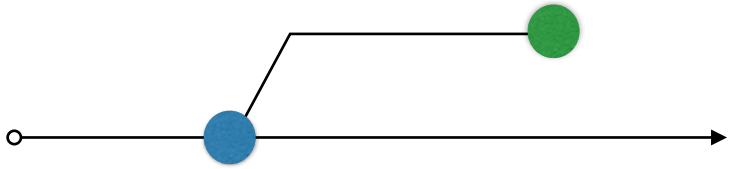
```
git checkout -b task101 master
```

- Make some changes to the files
- Commit

```
git add .
```

```
git commit -m"My first branch"
```

- Now have a branch with changes separate from master branch



Branches

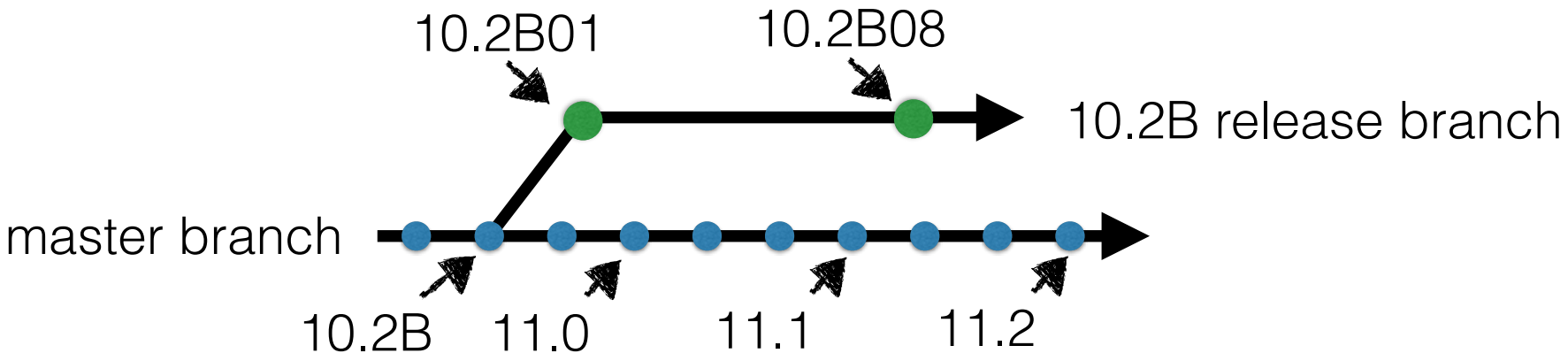
- The master named branch is the main branch by default,
  - Usually corresponds to release-ready code
  - Lives forever
- A feature branch exists until the work is completed and merged to the master branch
- A release or bug fix branch exists as long as needed
  - Usage depends on deployment situation - how many customers running older releases that you may have to patch
- Lots of discussions, examples online

Branch Maintenance

- Branches are intended to keep work separated
- Long lived branches often need to integrate work from other branches
- Two most common use cases:
  - Keeping a branch current with master - use rebase or merge
    - Rebase makes for a cleaner, simpler commit history
    - Merge adds (mostly) useless commits to the history
  - Applying bug fixes to release branches - use cherry-pick
    - Fix the bug in master, then cherry-pick to release branch

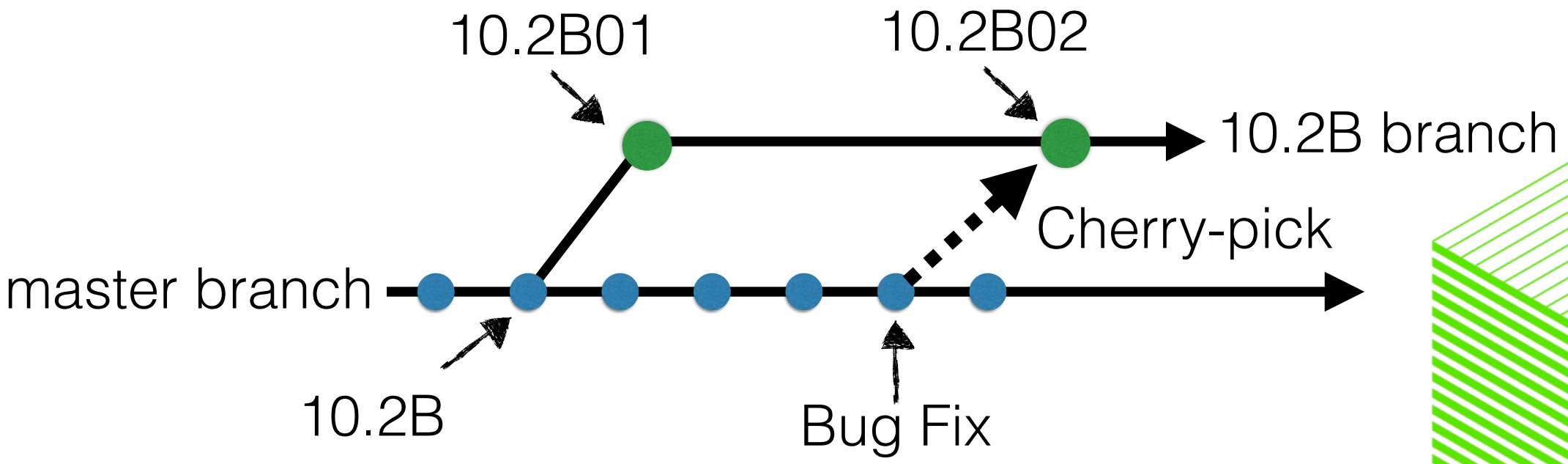
Release Branches

- A release branch is just a branch but is intended to live for the life of a release
  - E.g. OpenEdge releases (grossly simplified)
  - 10.2B released Dec **2009**, 10.2B08 Nov **2013**, 11.0 in **2011**
    - PSC must have worked on 11.0 and 10.2B at same time



Cherry Pick for Bug Fix Release

- Cherry-pick allows a “replay” of a change from one branch to another. E.g. OpenEdge releases
- 10.2B released Dec 2009, 10.2B08 Nov 2013,11.0 in 2011
- A bug found developing 11.0 (master branch) needs to also go in a 10.2B02 service pack (release branch)



Keep Commits Useful

- Git history shows changes over time. It's more useful if each commit means something
- No need to keep the "mini" commits; small changes you saved, just in case you needed to revert
- Before merging your branch, squash the commits - easy to do with a GUI tool
- Alternative to squash is amending a commit. Great for changing the commit message.
- Ideally, you use task software like Bugzilla or JIRA that generates an id for each task. You can use this id as the branch name to link work to task.
- Put the task id in the commit message as permanent connection to task

Distributed Git

- To share with other PCs, servers etc. you need a repo acting as a “server”. You can push changes to and pull changes from it
- Tools like Gitolite allow you to self host
- Easier to use a hosting company like GitHub or BitBucket
  - No firewall issues
  - Less setup and management
  - Offsite for redundancy, possibly faster
  - Great tutorials

Our Git Workflow

- We track/assign work using Bugzilla or JIRA
- Our master branch represents tested code, ready for release
- Developers create branches named after the Bugzilla task bz1234, usually from the master branch (unless it's a patch on a specific release branch)
- Developers push code branch to BitBucket for tester to retrieve and test
- Once code is approved, it is merged to master by gate keeper (senior developer) either using fast-forward merge or cherry-pick to avoid merge messages and keep clean history
- Task is closed and developer, gets Bugzilla email and sees commit appear in master branch.
- Developer deletes local and remote branch
- We keep release branches to patch old releases

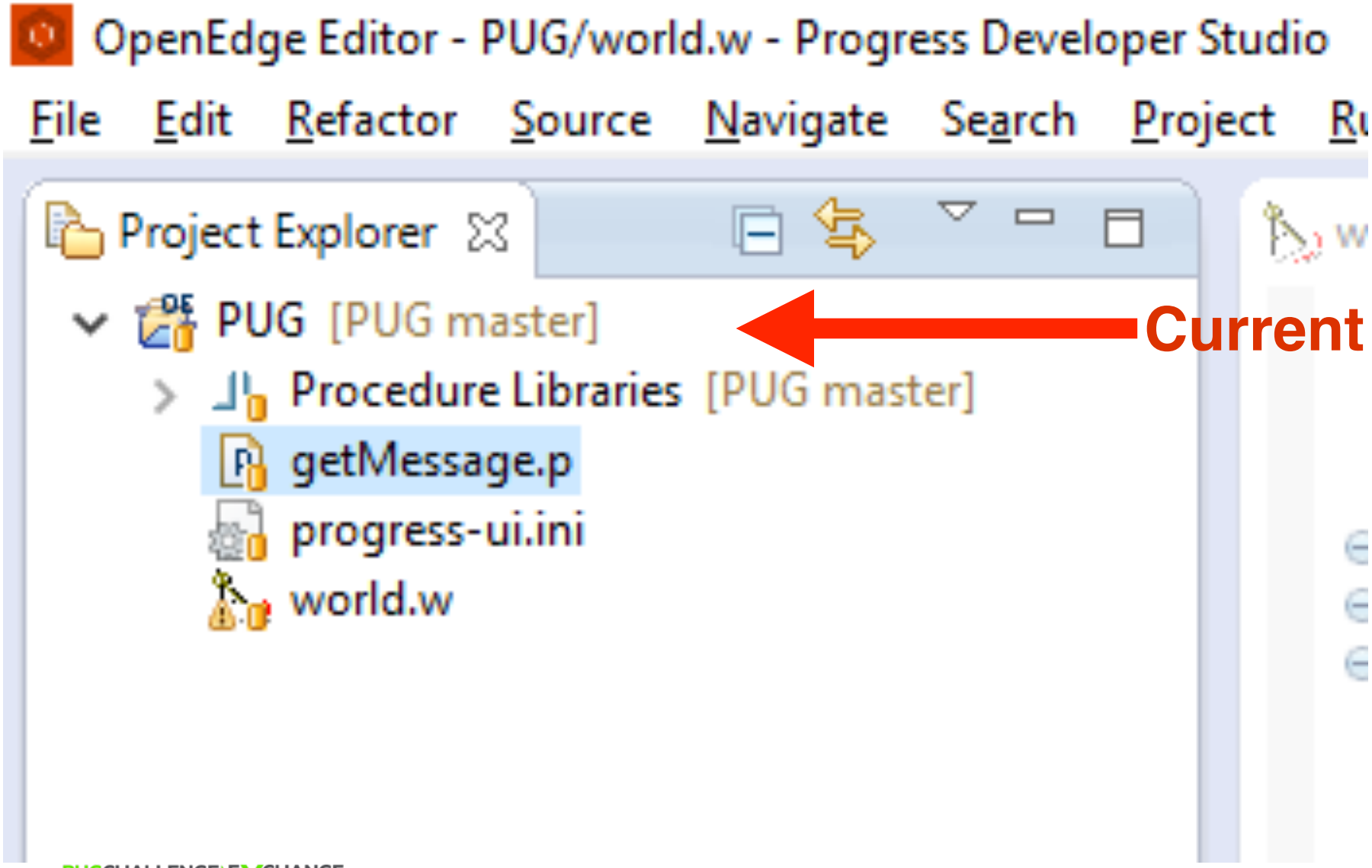
Your Git Workflow

- Decide on a workflow to control updates from developers to the important master and release branches
- Do you want a gatekeeper - control what gets merged to master or release branches?
- Many discussions online - search “Git workflow”
- Encourage good commit messages
  - They become your quick view of history
  - Not a good place for long descriptions, use a task tracking software for that.

Git GUI Tools

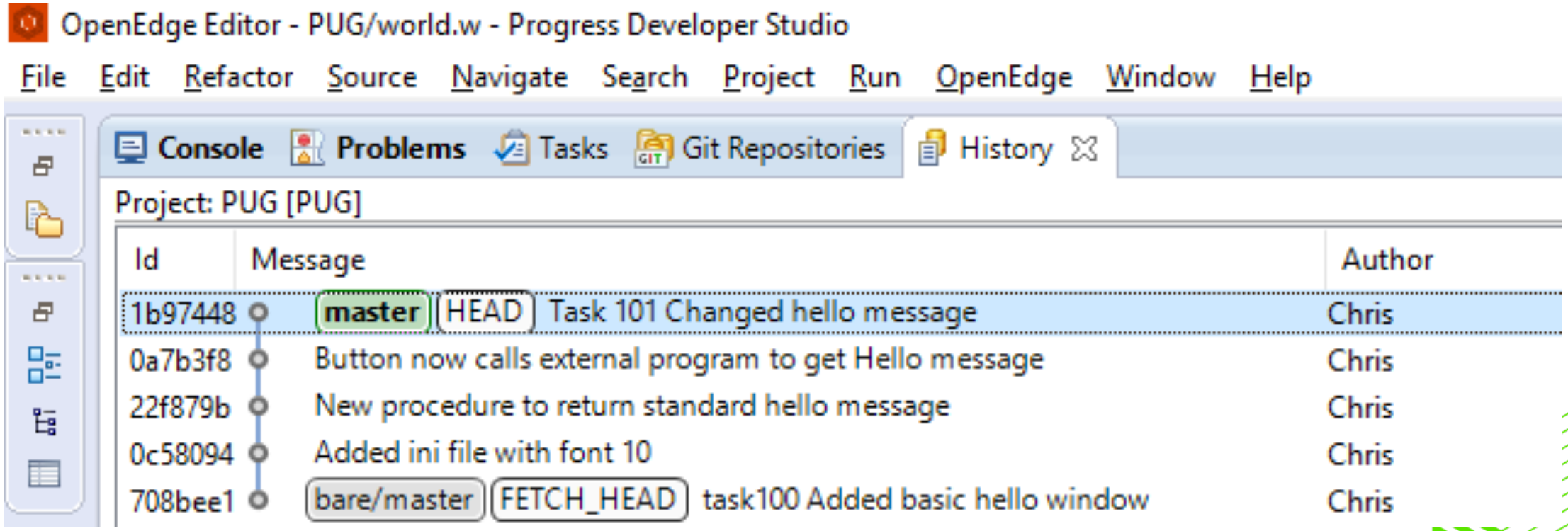
- Sometimes GUI tools are easier to use than command line.
  - File differences - side by side
  - Git log - also see the code snippets changed
  - Don’t have to memorize as many commands e.g. rename branch
  - Reset - just select the commit you want to reset to
- GUI tools I use
  - JGit/EGit - integrated into Eclipse
  - TortioseGit - Free but Windows only
  - SourceTree - From Atlasssian, free but requires registration. Mac or Windows

Git in Eclipse



Current branch is master

Git History in Eclipse



Git Compare in Eclipse

OpenEdge Editor - Compare getMessage.p 1b97448... and 0a7b3f8... - Progress Developer Studio

File Edit Navigate Search Project Run OpenEdge Window Help

world.w (AppBuilder) | getMessage.p | world.w | Compare getMessage.p 1b97448... and 0a7b3f8...

Text Compare

getMessage.p 1b97448... (Chris)

```
    Syntax      :

    Description : Define the message in one procedure.

    Author(s)   : Chris
    Created     : Sat Jun 25 11:26:42 PDT 2016
    Notes       :
  ----------------------------------------------------------------------*/

/* ***************************  Definitions  ************************** */

BLOCK-LEVEL ON ERROR UNDO, THROW.

/* ********************  Preprocessor Definitions  ******************** */


/* ***************************  Main Block  *************************** */
DEFINE OUTPUT PARAMETER pcMessage AS CHARACTER NO-UNDO.
pcMessage = "Hello Pug Challenge".
```

getMessage.p 0a7b3f8... (Chris)

```
    Syntax      :

    Description : Define the message in one procedure.

    Author(s)   : Chris
    Created     : Sat Jun 25 11:26:42 PDT 2016
    Notes       :
  ----------------------------------------------------------------------

/* ***************************  Definitions  ************************** 

BLOCK-LEVEL ON ERROR UNDO, THROW.

/* ********************  Preprocessor Definitions  ********** 


/* ***************************  Main Block  *************************** 
DEFINE OUTPUT PARAMETER pcMessage AS CHARACTER NO-UNDO.
pcMessage = "Hello Pug".
```

Bitbucket Teams Projects Repositories Snippets

Chris Hawkins / PUG

Commits

All branches

Author	Commit	Message
Chris Hawkins	1b97448	Task 101 Changed hello message
Chris Hawkins	0a7b3f8	Button now calls external program to get Hello message
Chris Hawkins	22f879b	New procedure to return standard hello message
Chris Hawkins	0c58094	Added ini file with font 10
Chris Hawkins	708bee1	task100 Added basic hello window

git

Bitbucket Teams Projects Repositories Snippets

Chris Hawkins / PUG

Source

master PUG /

getMessage.p	761 B	19 minutes ago	Task 101 Changed hello message
progress-ui.ini	5.9 KB	33 minutes ago	Added ini file with font 10
world.w	8.4 KB	22 minutes ago	Button now calls external program to get Hello message

Useful References

- The (free) Pro Git Book https://git-scm.com/book/en/v2 - everyone needs to read this book.
- Wikipedia has a good overview and many links https://en.wikipedia.org/wiki/Git_(software)
- A good description of why to use Git, with references to other good articles. http://www.netinstructions.com/the-case-for-git/ and https://colan.consulting/blog/business-case-switching-vcses-what-git-provides-over-subversion
- A useful cheat sheet PDF https://www.atlassian.com/dms/wac/images/landing/git/atlassian_git_cheatsheet.pdf
- A very good walkthrough of setup and usage. https://githowto.com/
- A popular Git branch workflow http://www.geekgumbo.com/wp-content/uploads/2011/08/nvie-git-workflow-commands.png

Thank You!

PUGCHALLENGE EXCHANGE
AMERICAS